CARL VON OSSIETZKY universität OLDENBURG

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Projektgruppenabschlussbericht

LSOP - Liveanalysen im Sport mit Odysseus P2P

vorgelegt von

Michael Brand, Tobias Brandt, Carsten Cordes, Simon Eilers, Simon Küspert, Torben Pehlke, Marc Preuschaft, Thomas Prünie, Pascal Schmedt, Thore Stratmann, Chris Tönjes-Deye, Marc Wilken

Gutachter:

Prof. Dr. Dr. h.c. H.-J. Appelrath
M. Sc. Timo Michelsen

Oldenburg, 1. April 2015

Zusammenfassung

Taktische Entscheidungen charakterisieren Mannschaftssportarten wie Fußball oder Basketball nachhaltig. Die Analyse von Trainingseinheiten und Spielen (z.B. Laufstrecken oder Passwege) formen mehr und mehr eine entscheidende Basis für diese taktischen Entscheidungen. Viele dieser Analysen sind video-basiert, was in der Regel zu hohen Betriebskosten führt. Zusätzlich wird ein hoch spezialisiertes System mit entsprechenden Ressourcen wie Prozessor und Arbeitsspeicher benötigt. Typischerweise werden die Ergebnisse einer solchen Videoanalyse in Spielunterbrechungen, wie einer Halbzeitpause beim Fußball, von Analysten vorgestellt.

Dieser Bericht stellt mit *Herakles* ein System vor, das Liveanalysen im Sport ermöglicht. Anders als video-basierte Systeme verwendet es Daten von Echtzeit-Sensoren (z.B. Chips im Ball und in Schuhen). Außerdem kommt für die Verarbeitung der Sensordaten ein Peer-to-Peer-Netzwerk aus herkömmlichen, kostengünstigen Maschinen wie Tablets, Notebooks oder Raspberry Pis zum Einsatz. Weil diese Sensordaten in der Regel sowohl ein hohes Volumen als auch eine hohe Datenrate aufweisen, basiert *Herakles* auf einem verteilten Datenstrommanagementsystem (DSMS).

Die Ergebnisse der Datenstrom-Verarbeitung sind für Trainer gedacht. Aus diesem Grund ist die Nutzerschnittstelle von *Herakles* eine Anwendung für mobile Endgeräte (z.B. Smartphones oder Tablets). Jedes Endgerät ist mit dem verteilten DSMS verbunden, empfängt Aktualisierungen der Statistiken und präsentiert sie in Echtzeit. Dadurch ermöglicht es Herakles dem Trainer, sein Team während eines laufenden Spiels zu analysieren und umgehend mit taktischen Entscheidungen zu reagieren.

Im Wesentlichen lässt sich *Herakles* in drei große Bereiche unterteilen: (1) Sensorik, (2) verteiltes DSMS und (3) mobiles Endgerät. Bei der Sensorik abstrahiert *Herakles* sowohl von bestimmten Sensoren als auch bestimmten Schemata, in denen die Sensordaten vorliegen. Dies macht es möglich, das Sensorsystem zu wechseln ohne wesentliche Änderungen an *Herakles* vornehmen zu müssen.

Die Verarbeitung der Sensordaten zu Statistik-Werten innerhalb des verteilten DSMS ist die Kernaufgabe von *Herakles*. Dabei wird auf ein vorhandenes System zur Datenstrom-Verarbeitung aufgebaut: *OdysseusP2P*. *Herakles* erweitert *OdysseusP2P* allerdings um Mechanismen zur Wahrung der Zuverlässigkeit. Mit einem dynamischen Load-Balancing ist es möglich, Peers mit starker Auslastung zu identifizieren und zu entlasten. Ein Recovery stellt Anfragen, die auf einem ausgefallen Peer ausgeführt wurden, auf einem anderen Peer wieder her. Neben diesen beiden zentralen Mechanismen zur Wahrung der Zuverlässigkeit, wurde eine Anwendung zur Analyse, Wartung und Fernsteuerung von Peers, die sich zu *Herakles* zusammengeschlossen haben, entwickelt.

Zur Visualisierung der Statistiken wurde eine Android-Anwendung für mobile Endgeräte entwickelt und in *Herakles* integriert. Sie ist in erster Linie für Trainer zur Verwendung während des Spiels am Spielfeldrand konzipiert. Außerdem wurde sie, wie der Rest von *Herakles*, so entwickelt, dass sie unabhängig von Sportarten ist (z.B. Fußball und Basketball).

Inhalt

1 Einleitung

Sport ist weltweit ein häufig diskutiertes Thema. Die Medien, Fans, Mannschaften und Sportler diskutieren über die Leistungen, Entscheidungen und Fehler aller Akteure auf und neben den Spielfeldern. Statistiken spielen in diesen Diskussionen häufig eine wichtige Rolle. Sie geben den Meinungen eine fundierte Grundlage und lassen Vergleiche zwischen verschiedenen Mannschaften und Sportlern zu. Das Ziel von *Herakles* ist es, einen Beitrag zu den Möglichkeiten der statistischen Auswertung von unterschiedlichen Sportarten zu leisten.

1.1 Motivation und Hintergrund

Im professionellen Sport ist der Bedarf an spiel- oder trainingsbezogenen Daten und daraus gewonnenen Auswertungen sehr hoch. Allein die Menge an Statistiken, die in sogenannten „Datencentern“ auf den Internetseiten von Tageszeitungen angeboten werden, verdeutlichen das Ausmaß des Bedarfs an Zahlen und Grafiken zu sportlichen Leistungen. Ein Beispiel eines solchen Datencenters der Süddeutschen Zeitung ist in Abbildung 1.1 zu sehen. Die Statistiken sind nicht nur für Fans der Sportarten interessant, sondern auch für Trainer. Aus den Statistiken können diese Rückschlüsse auf die Leistung des Teams und einzelner Spieler ziehen, sodass Strategieoptimierungen abgeleitet werden können.

Abbildung 1.1: Beispiel von Fußballstatistiken auf der Internetseite der Süddeutschen Zeitung [Zei15]

Um solche Statistiken zu erstellen, gibt es unterschiedliche Möglichkeiten. Diese reichen von manueller Erfassung, bei der ein Mensch das Spiel betrachtet und die Statistiken in eine Datenbank einträgt bis hin zu automatischen Systemen. Die automatisierten Systeme basieren meistens entweder auf Videoanalysen oder Sensorsystemen. Bei ersterem werten Computer mit Bildverarbeitungsalgorithmen Videoaufzeichnungen des Spiels aus und erstellen daraus Statistiken. Systeme auf Basis von Sensordaten erhalten die Position der Spieler von Sensoren, die an den Spielern und anderen Spie-

lobjekten angebracht sind und errechnen daraus die Statistiken. Manuelle Auswertungen sind sehr zeit- und kostenintensiv, sodass kommerzielle Hersteller wie bspw. Synergy Sports Technology[1] Produkte anbieten, welche die Aufzeichnung, Aufbereitung und Präsentation von Spielen automatisch übernehmen. Diese Produkte sind jedoch selbst wiederum kostenintensiv und benötigen eine ausgebaute Infrastruktur (bspw. Kameras, die das Spielfeld voll überblicken). Weiterhin liefern sie die Ergebnisse häufig nicht in Echtzeit, sondern erst in Spielunterunterbrechungen oder nach Spiel- bzw. Trainingsende, sodass eine unmittelbare Rückkopplung mit den Spielern nicht möglich ist. Andere Projekte im Rahmen der Forschung wie bspw. RedFIR [GFW+11] erfassen die Daten mit Sensoren, werten diese jedoch mit spezialisierter und damit häufig teurer Hardware aus.

Für die Trainer der Mannschaften wäre es von Vorteil, die Statistiken nicht erst nach dem Spiel zu erhalten, sondern direkt während des Spiels mit diesen arbeiten zu können. Taktische Entscheidungen könnten so direkt mit den Statistiken abgestimmt werden, um die richtigen Maßnahmen noch während des Spiels durchführen zu können. Genau dies wird von Herakles ermöglicht.

1.2 Ziele

Das grundlegende Ziel von Herakles ist es, Sportstatistiken live während laufender Spiele zu errechnen und diese so zu visualisieren, dass der Trainer einer Mannschaft komfortabel mit diesen arbeiten kann. Wird z.B. im Fußball eine Ecke gegeben, soll dies kurz nach der Ausführung in der Statistik angezeigt werden. Dabei soll Herakles auf günstige Hardware aufbauen, sodass es als System auch für kleinere Vereine einsetzbar ist. Um dies zu erreichen, soll das System möglichst mit vorhandener Hardware funktionieren, also z.B. auf Notebooks, welche die Trainer eines Vereins bereits besitzen.

Da ein einzelnes Notebook eventuell nicht leistungsfähig genug ist, um alle Statistiken zu berechnen, soll die Berechnung in einem P2P-Netzwerk durchgeführt werden. In einem solchen Netzwerk sind unterschiedliche Rechner, sogenannte Peers, zusammengeschaltet und lösen eine Aufgabe gemeinsam. Um die Statistiken live erstellen zu können, soll Herakles auf dem Prinzip von Datenströmen aufbauen und nutzt zur Berechnung dieser das DSMS Odysseus in der Variante Odysseus P2P. Die Berechnung in einem P2P-Netzwerk ist jedoch vergleichsweise instabil: Es ist möglich, dass Peers das Netzwerk unvorhergesehen verlassen, z.B. dann, wenn ein Trainer sein Notebook ausschaltet, während ein anderer weiter analysieren möchte. Ein weiteres Ziel ist also, die Berechnung im P2P-Netzwerk robuster zu machen, indem Techniken zur Lastverteilung und zum Wiederherstellen ausgefallener Peers eingebaut werden.

Um dem Trainer die Nutzung der Statistiken am Spielfeldrand zu ermöglichen, soll zudem eine Anwendung für ein Tablet erstellt werden. Auf diesem sollen die berechneten Statistiken übersichtlich und in einer für einen Trainer nützlichen Weise dargestellt werden.

1.3 Struktur der Dokumentation

Das vorliegende Dokument beschreibt die Ergebnisse der Projektgruppe. Dazu werden in Kapitel 2 die Grundlagen erklärt. Das beinhaltet z.B. eine Erklärung des DSMS Odysseus. In Kapitel 3 werden die Anforderungen an die Projektgruppe beschrieben. Das folgende Kapitel 4 gibt einen Überblick über das entstandene System Herakles und die einzelnen Komponenten, sodass ein Verständnis über

[1] In der amerikanischen Basketballliga NBA großflächig eingesetzt (`http://corp.synergysportstech.com/`)

das Zusammenspiel dieser entsteht. Im 5. Kapitel werden anschließend die für die Umsetzung von Herakles entstandenen Konzepte erläutert. Dazu gehören unter anderem die Konzepte für die Lastverteilung (engl. Load-Balancing) und die Wiederherstellung ausgefallener Peers (engl. Recovery).

Details zur Implementierung werden in Kapitel 6 gegeben. Dort wird z.B. auch die Implementierung der Coach Application, der Trainer-Anwendung für ein Tablet, erklärt. In Kapitel 7 werden die Ergebnisse der Evaluation gezeigt und erläutert. Hier wird z.B. die Geschwindigkeit von verteilter und nicht verteilter Berechnung verglichen.

Um das Projektmanagement der Gruppe und das gesamte Vorgehen im Projekt geht es in Kapitel 8. Hier wird unter anderem erklärt, wie die Gruppe Scrum umgesetzt hat. Die zum Entwickeln genutzte IT Infrastruktur und die Tools werden in Kapitel 9 beschrieben. In Kapitel 10 ist im Installationshandbuch beschrieben, wie Herakles installiert und eingerichtet werden kann. Das Benutzerhandbuch folgt in Kapitel 11. Am Ende wird in Kapitel 12 ein Fazit gezogen und ein Ausblick gegeben.

2 Grundlagen

Das folgende Kapitel stellt die wesentlichen Themen vor, die für das weitere Verständnis der Dokumentation wichtig sind. Dabei handelt es sich größtenteils um Seminararbeiten, die während des Projekts von einzelnen Teilnehmern angefertigt worden sind.

2.1 Sportanalyse

Heutzutage werden im Sport durch das Data Mining große Datenmengen gesammelt, die weiter verarbeitet werden müssen. Dazu soll im weiteren Verlauf dieses Teilkapitels ein Überblick über Möglichkeiten gegeben werden, wie aus einfachen Daten Informationen generiert werden können. Als erstes wird das Thema Data Mining sehr kurz angerissen, ein Überblick gegeben, wo das Data Mining im Sport entstand und wo es angewandt wird. Im Anschluss werden eine Reihe von Statistiken besprochen. Wobei hinzuzufügen ist, dass es in Mannschaftssportarten grundsätzlich zwei unterschiedliche Aspekte der Analyse gibt. Zum Einen muss die Mannschaft als Ganzes und zum Anderen die einzelnen Akteure betrachtet werden. Da jede Sportart spezielle Regeln und Spielmechaniken hat, wird sich hier auf Fußball und Basketball fokussiert. Im Folgenden wird zunächst auf Statistiken und die Schlussfolgerungen aus diesen für den einzelnen Spieler und im Anschluss auf Mannschaftsebene eingegangen.

2.1.1 Data Mining

Data Mining umfasst Werkzeuge und Techniken, um aus einer großen Menge von Daten Wissen zu extrahieren. Dabei geht es hier nicht um das klassische Analysieren von Statistiken, sondern um das algorithmisch gestützte Erkennen neuer Muster oder Beziehungen innerhalb des Datensets, die sonst verborgen geblieben wären. Das Data Mining ist i.d.R. in den Knowledge Discovery in Databases (KDD)-Prozess eingebettet, bei dem es darum geht, neue, gültige, nützliche und verständliche Muster in Daten zu erkennen.[Gra]

2.1.1.1 Data Mining im Sport - Geschichte

Sucht man im deutschen Sprachraum nach Literatur zur Analyse von Sportdaten, fällt auf, dass kaum Veröffentlichungen zu finden sind. Dieses Feld ist in Nord-Amerika wesentlich weiter verbreitet. Zurückzuführen ist dies auf die „Statistikrevolution“ in der in Amerika beliebten Sportart Baseball aus den 70er Jahren. Bill James begann 1977 mit der Herausgabe des *Bill James Baseball Abstracts*. In diesen jährlichen Ausgaben hat er die traditionell erfassten Statistiken des Sport hinterfragt und eigene damals unorthodoxe Formeln für die Leistungsbewertung aufgestellt. Er nannte diese neuen Statistiken *sabermetrics*. Mit der Zeit wuchs zwar das Interesse an seinen Texten und die Leser fingen an sich für die neuen Inhalte zu interessieren, in den Vereinen fanden sie allerdings lange Zeit keinen Anklang. Erst 2002, als sich der Manager der Oakland Athlectics, Billy Bean, auf die Suche nach Möglichkeiten machte, sein Team zu verbessern, fing er an, den *sabermetrics* anstelle der Meinung von Experten und Scouts bei der Auswahl neuer Spieler zu vertrauen. Bean hatte mit diesem Ansatz verhältnismäßig großen Erfolg und andere Manager folgten seinem Vorbild [SSC10].

Ähnliches geschah im Basketball. Wie Bill James hat auch Dean Oliver traditionelle Statistiken hinterfragt. Er fokussierte sich anders als James allerdings auf Teammetriken anstatt auf Einzelspieler. Sogar im subjektiven Feld der Team-Chemie, also wie gut die einzelnen Spieler miteinander harmonieren, versuchte Oliver Metriken zu erstellen. Er publizierte seine Überlegungen 2002 im Buch *Basketball on Paper: Rules and Tools for Performance Analysis* [SSC10].

Diese Art der Revolution hat es im Fußball bisher so nicht gegeben. Dies liegt daran, dass der Fußball schwieriger zu quantifizieren ist. Im Baseball sind die Aktionen im Spiel sequenziell und damit sehr leicht zu erfassen. Im Basketball werden eine viel größere Anzahl Aktionen ausgeführt. Dies ist bereits an den Resultaten ersichtlich - eine Basketballmannschaft kann in einem Spiel mehr als 100 Punkte erzielen. Dazu kommen noch weitere Würfe, die den Korb verfehlen, ständige Ein- und Auswechslungen, etc. Somit kann eine größere Anzahl Daten erfasst und verarbeitet werden. Ein Fußballspiel ist im Vergleich dazu Aktionsarm und es dauert länger bis eine aussagekräftige Datenbasis gesammelt ist. Allerdings wird die Entwicklung durch die britische Firma *Opta* forciert, die in Zusammenarbeit mit Manchester City alle gesammelten Daten zur Premier League Saison 2011/2012 für die Öffentlichkeit bereitgestellt haben, damit Nutzer neue Maßzahlen für den Sport entwickeln können.

2.1.1.2 Data Mining im Sport - Anwendungen

Ein sehr wichtiger Bereich, in den das Data Mining herein spielt, ist das Scouting von Spieler. Durch die Datenbanken, die über die letzten Jahre entstanden sind, können sich Vereine gezielter und tiefgreifender mit potenziellen Neuzugängen beschäftigen. Dies ist besonders für Vereine wichtig, die mit einem kleinen Etat auskommen müssen. Dies soll am folgenden Beispiel illustriert werden. Beim Scouting eines Stürmers, der primär für das Erzielen von Toren zuständig ist, reicht es nicht aus, auf die Torquote des Vorjahres zu blicken, da nicht garantiert werden kann, dass er im nächsten Jahr die Quote beibehält. Darüber hinaus verrät diese Zahl nicht, ob die Tore nur zufällig und mit Glück entstanden sind. Hier unterstützt das Data Mining die Suche. Untersucht man nun die relevanten Daten, stellt sich heraus, dass sich Stürmer, die in einer Saison Torchancen mit hoher Erfolgswahrscheinlichkeit erarbeitet haben, dies in der nächsten Saison meist reproduzieren können. Demgegenüber steht die Erkenntnis, dass die Quote der Umwandlung der Torchancen in Tore nicht mit der Quote der Folgesaison korreliert. Mit anderen Worten ist es zu einem großem Teil vom Glück abhängig, ob der Stürmer viele oder wenige Tore schießt, solange er jedoch konstant zu hochwertigen Abschlüssen kommt, ist eine gewisse Torquote garantiert. Ist einer der finanzschwächeren Vereine nun auf der Suche nach einem Stürmer, muss er sich geschickt anstellen und nach Spielern suchen, die noch nicht durch eine große Anzahl an Toren auf sich aufmerksam gemacht haben und somit sehr teuer wären. Stattdessen schaut dieser Verein, ob es Spieler gibt, die zwar gute Torgelegenheiten hatten, diese jedoch nicht verwandeln konnten und entsprechend günstiger sind [elf14].

Ein weiterer Anwendungsfall für die Analyse von Daten ist das sogenannte *Milan Lab* des AC Milan, der 2002 in einem Pilotprojekt damit begonnen hatte, eine Software einzusetzen, die Verletzungen der Spieler, auf Basis der Werte die in den Fitnesstests gesammelt werden, voraussagt, um sie dann zu vermeiden. Sobald bei einem Spieler Fitnesswerte unterhalb der normalen Werte sinken, signalisiert das Programm, dass eine Verletzung vorliegt oder das eine Verletzung schlimmer geworden ist. In einem Testlauf mit Daten aus der vorangegangenen Saison erzielte die Technologie eine Trefferquote von 70%. Das System war nach seiner Einführung sehr erfolgreich, denn in den nächsten Jahren sank die Anzahl der Verletzungen um fast 90 Prozent. Den Mitarbeitern des Labs ist es

außerdem möglich anhand einer eigens kreierten „Norm der Wellness“, auf Basis der mentalen und physiologischen Werte, zu sagen wann ein Spieler in Topform ist. [Man11] Zu dieser Zeit hat Milan auch dank ihres Labs viele Erfolge gefeiert. Jedoch ist dies anderen Vereinen nicht verborgen geblieben, so dass vielerorts die Technologie Einzug erhalten hat und der Vorsprung den der AC Milan hatte schnell wieder geschmolzen war.

Es existieren sicherlich weitere Anwendungsgebiete, jedoch sollte die beiden genannten Beispiele ausreichen, um zu erkennen welche Wichtigkeit das Data Mining für Sportvereine hat.

2.1.2 Mannschaftsstatistik

Aus den gesammelten Daten werden Statistiken extrahiert, die im Folgenden untersucht werden. Zunächst wird eine Übersicht über die populärsten Fußballstatistiken gegeben, die meist bei Fernsehübertragungen zu sehen sind und jeweils darauf eingegangen wie diese weiter verfeinert werden können, um eine bessere Aussagekraft zu erhalten. Im Anschluss werden Kennzahlen für die Mannschaftsleistungen im Basketball erläutert. Sowohl beim Fußball als auch beim Basketball gibt es noch zahlreiche weitere Aspekte, die betrachtet werden könnten.

- Torschüsse
- Passzahlen (gespielte Pässe, Fehlpässe, Passquote)
- Ballbesitz
- Zweikampfwerte

2.1.2.1 Torschuss

Die Anzahl der Torschüsse als Ganzes sagt nichts darüber aus, ob die Torschüsse vielversprechend waren. Nun kann man diese Torschüsse weiter aufschlüsseln und angeben, ob der Schuss inner- oder außerhalb des Strafraums abgegeben wurde oder ob es sich nicht um einen Schuss handelte, sondern um einen Kopfball. Dennoch reicht dies ebenfalls nicht aus, denn auch innerhalb des Strafraums gibt es ungünstige Positionen aus denen geschossen werden kann. Eine wesentlich genauere Angabe ist der Expected Goals (ExpG)-Wert. Dieser Wert quantifiziert die Chancenqualität einer Mannschaft, basierend auf den Positionen aus denen auf das gegnerische Tor geschossen wird. Aktuell existiert hierfür keine allgemeine Berechnungsweise, sondern jeder Anbieter ermittelt den ExpG-Wert auf eine andere Art, indem verschiedene Einflussfaktoren einbezogen werden. Für eine übersichtliche Darstellung können die Werte in einen Graphen, wie Abbildung 2.1 zeigt, eingetragen werden. In dieser Abbildung ist ebenfalls zu sehen, dass Dortmund das Spiel unverdient gewann, da Wolfsburg aufgrund der Chancenqualität vier Tore hätte schießen können. Somit ist diese Grafik ein Indiz für den Trainer, ob seine Mannschaft andere Wege gehen muss um zu Torgelegenheiten zu kommen oder ob die Mannschaft zu viele gefährliche Chancen zulässt.

2.1.2.2 Passzahlen

Der nächste Punkt sind die Passwerte. Auch hier sagt die absolute Zahl eher wenig aus, denn es wird nicht klar, wo und über welche Distanz die Pässe gespielt wurden. Um in einem Fußballspiel genauer

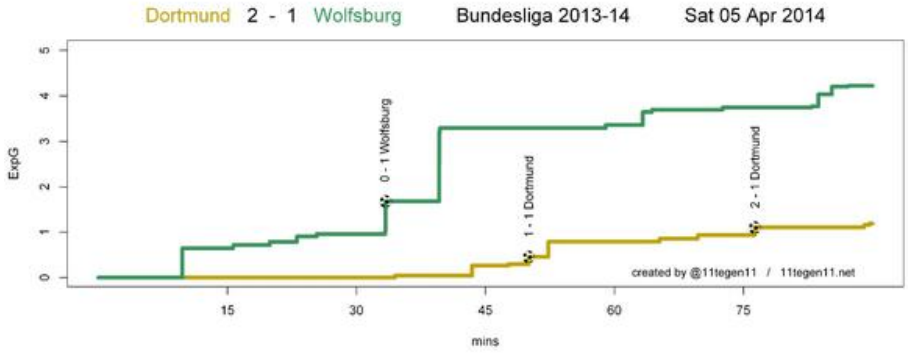


Abbildung 2.1: ExpG Plot für das Spiel Dortmund - Wolfsburg [1]

zu betiteln, wo Aktionen stattfinden, wird das Spielfeld in drei Drittel aufgeteilt. Das dritte Drittel ist dabei das Angriffsdrittel, das erste das Verteidigungsdrittel und das zweite liegt dazwischen. Wird bei einer Mannschaft der Ball oft im ersten Drittel zwischen den Verteidigern zirkuliert, um eine Lücke im gegnerischen Block zu suchen, steigt die absolute Passanzahl ohne das Gefahr für den Gegner entsteht. Da es gleichzeitig in den meisten Fällen einfache, sichere Pässe sind steigt die Quote ebenso. Im Gegensatz dazu steht eine Mannschaft, die das Spiel nicht über Flachpasskombinationen aufbauen möchte, sondern eher versucht mit langen Bällen aus der Abwehr heraus, einen Zielspieler im Angriffsdrittel anzuspielen. Eine solche Spielweise resultiert in weniger gespielten Pässen und eine niedrigere Passquote. Eine Mannschaft sucht im Angriffsdrittel oft nach einer anderen Passart als zuvor. Hier kann es lohnenswert sein risikoreicher zu passen, um zu Torchancen zu gelangen. Diese werden in Schnittstellen- und Schlüsselpässe, sowie Flanken unterteilt. Somit sollten Pässe zusätzlich zu den Risikopassarten in Ort und Passdistanz klassifiziert werden. Neben den Passzahlen ist für einen Trainer eine grafische Darstellung der Passrichtungen interessant. Wenn er in dieser Anzeige z.B. sieht, dass seine Mannschaft gegen eine sehr tief stehende Mannschaft kaum in den Strafraum eindringt, sondern nur um den Strafraum herumspielt, kann er geeignete Anpassungen vornehmen. Eine solche Darstellung kann ebenfalls helfen Passmuster im gegnerischen Spiel zu ermitteln, um diese dann durch passende Maßnahmen für sich zu nutzen.

2.1.2.3 Ballbesitz

Beim Ballbesitz verhält es sich ähnlich wie beim Passspiel. Die beiden Werte hängen auch in gewissem Maße zusammen, da je häufiger Pässe eine Mannschaft spielt, desto mehr muss sie auch den Ball in den eigenen Reihen haben. Dementsprechend ist auch hier wichtig, wo der Ballbesitz gesammelt wird. Ballbesitz kann auf verschiedene Arten berechnet werden. Zum einen in dem die Ballkontakte einer Mannschaft durch die gesamten Ballkontakte geteilt wird und zum anderen durch das Stoppen der Zeit einer Mannschaft am Ball geteilt durch die gesamte Zeit am Ball.

2.1.2.4 Zweikampfwerte - Defensive

Das Thema der Zweikampfwerte soll nun zum Anlass genommen werden, die Defensivarbeit einer Mannschaft zu besprechen. Diese fängt bereits beim Großteil der Mannschaften beim Stürmer an. Der Grund liegt darin, dass der Spielaufbau der gegnerischen Mannschaften so gut wie möglich gestört

[1] Quelle: `https://twitter.com/11tegen11/status/452884430579707904`

werden soll. Wichtig in diesem Zusammenhang ist ebenfalls das Pressing. Die Idee beim Pressing liegt darin, den Gegner soweit unter Druck zu setzen, dass er Abspielfehler begeht oder den Ball direkt verliert. Eine einfache Form des Pressing läuft wie folgt ab. Ein Spieler läuft den Ball-führenden Spieler an, während seine Mitspieler an mögliche nahe Passoptionen heranrücken. Dadurch kann der Spieler am Ball keinen sicheren Pass spielen, da seine Mitspieler sollte ein Pass bei ihnen ankommen ihrerseits direkt wieder unter Druck stehen und Gefahr laufen den Ball zu verlieren. Ein möglicher Messwert ist der Abstand zu den Gegenspielern in einer solchen Pressingsituation, um zu analysieren wie gut das Pressing ausgeführt wird.

2.1.2.5 Basketball

Da die Anwendung von Statistiken gerade in der National Basketball Association (NBA) bereits seit längerer Zeit praktiziert wird, sind Schlüsselelemente des Spiels analysiert. Dean Oliver hat in seinem Buch die vier Faktoren für Erfolg im Basketball identifiziert. Alle vier Faktoren können sowohl offensiv als auch defensiv angewendet werden und bekamen eine Gewichtung zugeordnet. Diese Faktoren sind:

Werfen Dieser Faktor wird über die Effective Field Goal Percentage beziffert und bezieht die Tatsache ein, dass Drei-Punkte-Würfe 50% mehr Wert sind als Zwei-Punkte-Würfe. Die Formel dafür lautet:

$$(Koerbe + 0,5 * 3er)/Korbversuche \tag{2.1}$$

Turnovers Dies sind Ballverluste. Die Formel für die Turnover Percentage lautet :

$$Turnovers/(Korbversuche + 0,44 * Freiwurfversuche + Turnovers) \tag{2.2}$$

Rebounding Dieser Faktor wird durch offensive und defensive Rebound Percentage (ORB% und DRB%) ausgedrückt:

$$ORB\% = ORB/(ORB + DRBGegner) \tag{2.3}$$

$$DRB\% = DRB/(DRB + ORBGegner) \tag{2.4}$$

Freiwurfrate Beschreibt wie Effektiv ein Team ist, an die Freiwurflinie zu kommen und diese Möglichkeiten in Punkte umzuwandeln. Hier ist die Formel:

$$Freiwurfrate = Freiwuerfe/Korbversuche \tag{2.5}$$

2.1.3 Einzelspielerstatistik

Auch wenn es im Fußball vier Mannschaftsteile gibt (Angriff, Mittelfeld, Abwehr und Torwart) geht die Entwicklung der Taktik längst in eine Richtung, die es nötig macht, dass gerade Feldspieler in allen Bereichen gut ausgebildet sind. So ist der Stürmer oft nicht mehr nur für das erzielen von Toren zuständig, sondern stellt bei gegnerischem Ballbesitz die erste Verteidigungslinie dar. Somit sind die meisten Statistiken für alle Spieler von Bedeutung, wenn auch mit unterschiedlicher Gewichtung. Die Abbildung 2.2 zeigt eine Grafische Aufarbeitung vieler Saisonstatistiken von Arjen Robben in einer Radardarstellung. Zuerst wurde diese Art der Darstellung von Roger Pimentel, dem Chefanalysten von *howtowatchsports*[2] erfunden, um Basketballspieler miteinander zu vergleichen[Pim09]. Später

[2] www.howtowatchsports.com

hat Ted Knutson, einer der Autoren von *statsbomb*[3], die Adaption auf Fußballstatistik erstellt. Diese Statistiken sind normiert pro 90 Minuten um sie vergleichbar zu machen, lassen sich aber ebenso innerhalb eines Spiels anwenden. Ein Großteil dieser Werte sind in Tabelle 2.1 aufgelistet, inklusive einer Einteilung der Wichtigkeit für die Mannschaftsteile. Hier ist offensichtlich, dass für das Mittelfeld alle Werte wichtig sind. Das liegt daran, dass in der Tabelle die Punkte nur grob zugeordnet werden können, da es innerhalb des Mittelfeldes wiederum viele verschiedene Spielerrollen gibt, bei denen auch wieder Gewichtungen vorgenommen werden müssten, was hier zu weit ins Detail gehen würde.

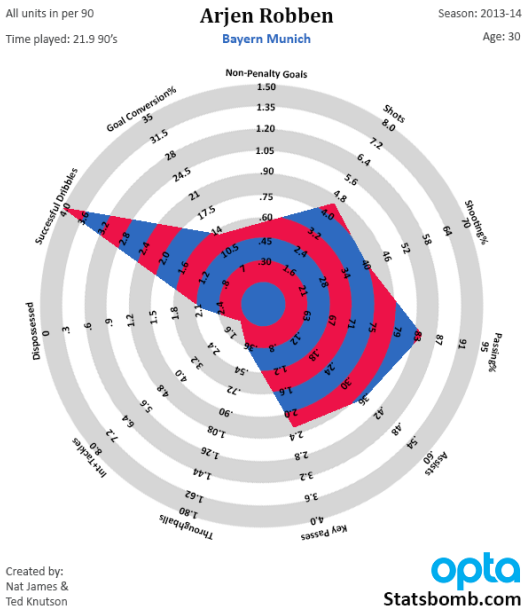


Abbildung 2.2: Radardarstellung: Arjen Robben - [Knu14]

Analog funktioniert dies mit Basketball-Statistiken. Hier sind die typischen Zahlen des Sports sinnvoll. Diese sind unter anderen gespielte Minuten, Punkte gesamt, Rebounds, Assists, Steals. Blocks und Fouls. Außerdem existieren Formeln zur Berechnung der Performance eines Basketballspieler, die die genannten Größen verwenden. Die Formel zur Effektivität eines Spielers wurde von Martin Manley erfunden und lautet: *Effektivität = (Punkte + Rebounds + Assists + Steals + Shotblocks) - ((Anzahl Wurfversuche - Anzahl verwandelte Würfe) + (Anzahl Freiwurfversuche - Anzahl verwandelte Freiwürfe) + Turnover)*. An dieser Formel kann kritisiert werden, dass weder die Zeit, die ein Spieler auf dem Platz war, noch eine Gewichtung der Unterschiedlichen Größen stattfindet. Dies beachtet wiederum einen weiteren Wert, das Player Efficiency Rating. Hier werden sehr viele Einflussfaktoren und Gewichtungen beachtet, was zu einer sehr komplexen Formel führt, die z.B. in [Kub] aufgeführt ist. Eine weitere Formel ist dem Offensive Rating zuzuordnen. Er beziffert, den Beitrag einzelner Spieler zur Offensive und berechnet sich aus (Punkte produziert / Individueller Ballbesitz) * 100. Beide benötigten Werte werden aus Formeln berechnet, in denen die Größen erzielte und verfehlte Körbe, Freiwürfe, Assists, offensive und defensive Rebounds, sowie Turnovers einbezogen werden. Anders als beim Fußball gibt es beim Basketball die Unterteilung zwischen den Mannschaftsteilen

[3] www.statsbomb.com

Statistik	Offensive	Mittelfeld	Abwehr	Torwart
Schüsse	x	x		
Schuss%	x	x		
Schusserfolg%	x	x		
Torschussvorlagen	x	x		
Pass%	x	x	x	x
Lange Pässe		x	x	x
Schlüsselpässe	x	x	x	
Steilpässe	x	x	x	
Flanken	x	x		
Abgefangene Bälle	x	x	x	x
Ballkontakte	x	x	x	x
Pass pro Ballkontakt	x	x	x	
Zweikampf%		x	x	
Kopfball%	x	x	x	
Verlorene Bälle	x	x	x	
Ausgedribbelt worden		x	x	
Erfolgreiche Dribblings	x	x		
Fouls		x	x	

Tabelle 2.1: Relevanz versch. Feldspielerstatistiken für die Mannschaftsteile

nicht, da sich nur fünf Spieler pro Team auf dem Spielfeld befinden und somit alle Spieler die Defensive und Offensive bilden.

Auch in der Fußballwelt werden Spielerleistungen mit Noten beziffert. Bekannte Online-Fußball-Manager wie *Communio*[4] basieren auf solchen Spielernoten. Problematisch wird es allerdings dann, wenn diese nicht aus reinen objektiven Gesichtspunkten ermittelt werden, sondern letztendlich subjektiv von Menschen vergeben werden. In solchen Fällen können Spieler nie objektiv miteinander verglichen werden, da immer zweifel an Einzelnoten gegeben sind[Kös08]. Deshalb muss es, wie im Basketball, kriterien geben anhand dessen Spieleraktionen auf dem Platz bewertet werden. Wie die Berechnungen im genauen geschehen geben die Anbieter allerdings nicht preis. Eine der bekanntesten Statistik-Portale für Fußball ist *WhoScored*[5]. Auch hier wird nur vage erklärt wie es zu den Bewertungen kommt. Es gehen über 200 Statistiken in die Notenvergabe ein, die nach dem Einfluss auf das Spiel gewichtet werden und jedes Ereignis auf dem Feld wird inklusive der Wichtigkeit, Ort auf dem Spielfeld und dem Resultat einbezogen. Dabei wird von einem Startwert von 6.0 auf einer Skala von 1.0 bis 10.0 ausgegangen und jeweils für die Ereignisse Punkte addiert oder subtrahiert[who]. Für Trainer sollte ein genauer Darstellung über Statistiken allerdings wichtiger sein, da er seine Spieler genauer analysieren kann.

Zusätzlich zu den genannten Werten, ist es sinnvoll Tracking-Statistiken, wie gelaufene Distanz, Anzahl schneller/intensiver Läufe und Anzahl Sprints aufzunehmen [SSC10]. Neben den bisherigen Statistiken die auf numerischen Werten basieren, gibt es die Möglichkeit die Position und Laufwege, sowie die Positionen bestimmter Ereignisse, wie Torschüsse, der Spieler auf dem Feld, zu visuali-

[4] www.comunio.de
[5] www.whoscored.com

sieren. Abbildung 2.3 zeigt die Heatmap von Würfen, die ein Spieler in einem NBA-Spiel getätigt hat.

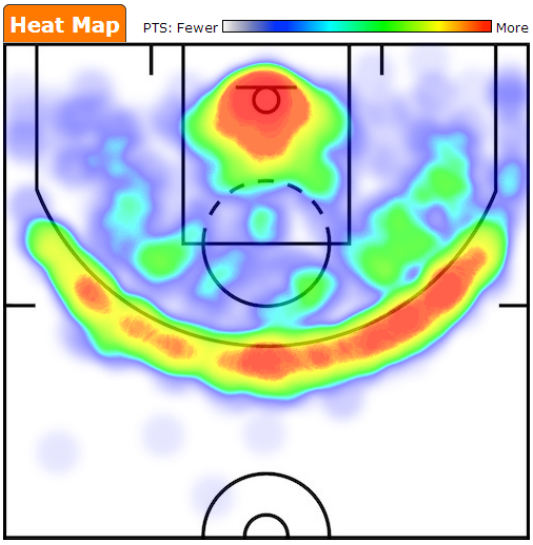


Abbildung 2.3: Beispiel einer Heatmap mit Korbwürfen von einem Spieler - [ref15]

2.1.4 Zusammenfassung

Abschließend kann gesagt werden, dass es eine große Zahl an Statistiken gibt, die es zulassen, dass jeder taktische Aspekt eines Spiels sich abbilden lässt, so dass es einfach wird zu überprüfen ob die Vorgaben des Trainers sinnvoll sind oder nicht. Natürlich müssen die Umstände bei all diesen Statistiken beachtet werden. Weist ein Spieler schwache Werte auf, muss das nicht zwangsläufig heißen, das er schlecht spielt. Es ist ebenso möglich, dass die gegnerische Mannschaft sehr gut gegen ihn verteidigt. Ein weiteres Beispiel für eine falsche Bewertung kann die Gesamtdistanz sein. Selbst wenn ein Spieler sehr viel gelaufen ist, heißt dies zwar wahrscheinlich, dass er großen Einsatz zeigt, allerdings nicht ob er die Kilometer sinnvoll genutzt hat, oder ob er viele Wege umsonst gelaufen ist. Für einen Trainer, der an der Seitenlinie steht, ist es wichtig eine ungefähre Vorstellung davon zu haben, welche Werte er bei seinen Spielern sehen möchte.
Damit Statistiken in irgendeiner Art und Weise zustande kommen können, werden die Positionen der Spieler und des Balls in Echtzeit lokalisiert und ausgewertet. Das folgende Teilkapitel befasst sich dabei mit dem Thema der Echtzeitlokalisierung.

2.2 Sensorik und Echtzeitlokalisation

Im Bereich der Logistik stehen viele Unternehmen unter hohem Kostendruck und müssen daher ihre internen Prozesse optimieren. Mithilfe einer Lokalisierung von Waren, Betriebsmitteln oder Personen lassen sich die internen Prozesse exakt aufzeichnen und dementsprechend optimieren. Aber nicht nur in der Logistik kann eine Lokalisierung hilfreich sein. So kann diese Technik auch im Sportbereich eingesetzt werden, um z.B. Schwächen und Stärken einer Mannschaft zu analysieren. Im nächsten Schritt kann die Analyse dabei helfen, spezielle Trainingspläne oder Taktiken zu entwickeln, um die

Schwächen zu beseitigen bzw. die Stärken zu nutzen. Unter Lokalisierung versteht man im Allgemeinen die Ermittlung des Ortes von Objekten zu einem Zeitpunkt [Grü13b]. Ein bekanntes Verfahren ist hierbei das satellitengestützte Global Positioning System (GPS), das in Navigationssystemen sowie in vielen Smartphones oder Tablets genutzt wird [TS08]. Zur Bestimmung des Ortes eines Objektes gibt es unterschiedliche Techniken. Im nachfolgenden Abschnitt werden die drei verschiedenen Verfahren Triangulation, Schauplatzanalyse und Näherungsanalyse näher erläutert. Im Abschnitt 2.2.1 werden anschließend verschiedene Techniken zur Positionsbestimmung beschrieben. Hierbei können die Lokalisierungssysteme einzelne oder verschiedene Kombinationen aus den einzelnen Techniken nutzen.

2.2.0.1 Triangulation

Bei der Triangulation werden die geometrischen Eigenschaften von Dreiecken genutzt, um die Positionen von Objekten zu bestimmen [Röh13]. Hierbei können die Postionen einmal durch Angulation (Berechnung der Position mittels Winkeleigenschaften) oder Lateration (Berechnung der Position mithilfe von Abständen) bestimmt werden, welche in den folgenden Abschnitten genauer beschrieben werden [Jar06].

Angulation

Angulation verwendet zur Positionsbestimmung Winkelangaben. Soll die Position eines Objektes festgestellt werden, ist dies also durch die Messung der Entfernung von zwei bekannten Positionen und deren Winkel zur unbekannten Position möglich. Dieses Verfahren wurde bereits im Altertum zur Landvermessung eingesetzt und wird heute noch, abgesehen von technischen Neuerungen, dafür eingesetzt. Abbildung 2.4 zeigt ein Dreieck mit den Postionen A,B und C. Angenommen, es sind die Positionen A und B bekannt und die Position von C soll bestimmt werden. So müssen zunächst die Winkel α und β ermittelt werden. Diese Winkel können z.B. mit einem Theodolit aus der Vermessungstechnik ermittelt werden [Bil10]. Anschließend lässt sich γ wie folgt berechnen [Jar06]:

$$\gamma = 180 - (\alpha + \beta)$$

Die Strecken zwischen AC und BC werden durch folgende Formeln ermittelt:

$$AC = \frac{(AB * sin(\beta))}{sin(\gamma)}, BC = \frac{(AB * sin(\alpha))}{sin(\gamma)}$$

Zum Schluss lässt sich die Stecke zwischen dem Mittelpunkt M zur Position C berechnen [Jar06]:

$$MC = \sqrt{((\frac{AB}{2}) - BC * cos(\beta))^2 + (BC * sin(\beta))^2}$$

Lateration

Bei der Lateration wird die Position eines unbekannten Objektes mithilfe der Entfernungen zu bekannten Objekten bestimmt. Dies bedeutet, dass für eine zweidimensionale Positionsbestimmung

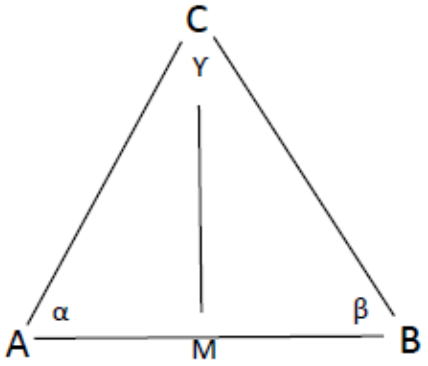


Abbildung 2.4: Triangulation

mindestens drei Referenzpunkte bekannt sein müssen. Soll zusätzlich die Dimension bestimmt werden, benötigt man insgesamt vier Referenzpunkte. Um die Position eines unbekannten Objektes zu berechnen, müssen die Entfernungen zu den drei bekannten Referenzpunkten gemessen werden. Aus diesen Entfernungen lässt sich dann anschließend die Position des Objektes bestimmten. Abbildung 2.5 zeigt hier drei Punkte (z.B. Wireless Access Points) mit folgenden Koordinaten: A (4,4), B(30,10) und C(15,30) [NAMS13], sowie die empfangende Signalstärke in Punkt X. Des Weiteren wird davon ausgegangen, dass die Punkte A,B und C jeweils eine maximale Reichweite von 30 Metern besitzen. Um den Abstand von dem Objekt X zu den einzelnen Punkten zu berechnen, kann z.B. die empfangende Signalstärke der einzelnen Punkte (A,B,C) im Punkt X genutzt werden (siehe Kapitel 2.2.1.3 bzw. Kapitel 2.2.1). Anschließend kann durch folgende Formel der Abstand zu den einzelnen Punkten berechnet werden [NAMS13].

$$d_i = p - (1 - m_i)$$

$m =$ Empfangende Signalstärke $p =$ Maximale ausbreitung des Signales

$$d_1 = 30 - (1 - 0,8) = 6$$
$$d_2 = 30 - (1 - 0,7) = 9$$
$$d_3 = 30 - (1 - 0,85) = 4,5$$

Anschließend stellt man die Gleichungen für die drei Knoten wie folgt auf [NAMS13]:

$$\sqrt{(x_i - x)^2 + (y_i - y)^2} = d_i \text{ for i = 1,2,3}$$

X und Y bezeichnen hierbei die unbekannten Positionen von Objekt X. Zieht man Gleichung 3 von 1 & 2 ab und stellt diese anschließend um, erhält man folgende Matrix:

$$2 * \begin{pmatrix} x_3 - x_1 & y_3 - y_1 \\ x_3 - x_2 & y_3 - y_2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} (r_1^2 - r_3^2) - (x_1^2 - x_3^2) - (y_1^2 - y_3^2) \\ (r_2^2 - r_3^2) - (x_2^2 - x_3^2) - (y_2^2 - y_3^2) \end{pmatrix}$$

Diese Gleichung mit zwei Unbekannten, lässt sich eindeutig lösen. So erhält man für $x = 14.21$ und $y = 15.30$. Somit befindet sich das gesuchte Objekt X an den Koordinaten (14.21, 15.30) [NAMS13].

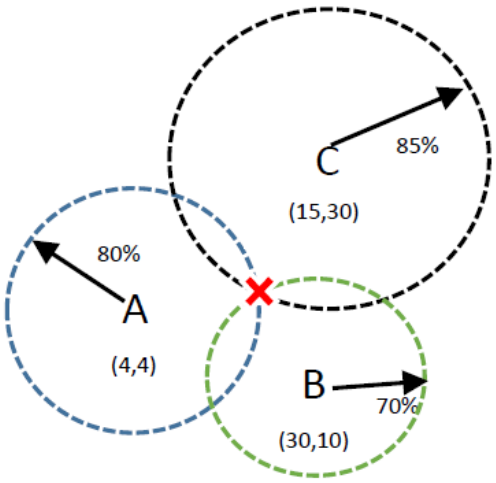


Abbildung 2.5: Ortung durch Distanzmessung (in Anlehnung an [NAMS13])

2.2.0.2 Schauplatzanalyse

Bei der Schauplatzanalyse wird die Positionierung eines Objektes anhand von Referenzdaten berechnet. Um diese Referenzdaten zu erhalten, wird zunächst das entsprechende Gebiet, in dem später Objekte geortet werden sollen, analysiert. Bei dieser Analyse können z.B. Umgebungsbilder oder Signalstärken aufgezeichnet werden. Damit ein späterer Vergleich der Daten einfacher ist, werden z.B. die Umgebungsbilder in Umrisslinien umgewandelt. Somit wäre es z.B. möglich, Umgebungsbilder, die von einem autonomen Roboter aufgenommen wurden, mit den Umrisslinien zu vergleichen und somit eine Positionsbestimmung vorzunehmen [Sch12].

2.2.0.3 Näherungsanalyse

Mit Hilfe der Näherungsanalyse kann festgestellt werden, ob sich ein unbekanntes Objekt in der Nähe von einem bekannten Objekt befindet. Dies kann z.B. durch Berührungssensoren oder anderen Detektoren festgestellt werden. In der Praxis aber stellen diese Sensoren lediglich fest, dass ein Objekt in ihrer Nähe ist, aber nicht, um welches Objekt es sich handelt. Eine weitere Möglichkeit Objekte zu Orten ist die Verwendung von drahtlosen Access Points. Befindet sich ein Objekt innerhalb eines Funknetzes und empfängt von mehreren Drahtlosen Access Points ein Signal, ist über eine Schnittpunktbildung eine Eingrenzung der Position möglich.

2.2.1 Techniken zur Positionsbestimmung

Abhängig davon, ob die Positionsbestimmung mit dem Triangulations- oder Trilaterationsverfahren bestimmt wird, gibt es unterschiedliche Techniken, um diese zwei Verfahren umzusetzen. Abbildung 2.6 zeigt die verschiedenen Techniken, sowie die benutzten Informationen um eine Positionsbestimmung durchzuführen [Röh13].

Im folgenden Abschnitt werden Positionsessverfahren wie Laufzeitmessung, Laufzeitdifferenzen und Winkelmessung vorgestellt, die bei einer bestehenden Sichtverbindung eingesetzt werden kön-

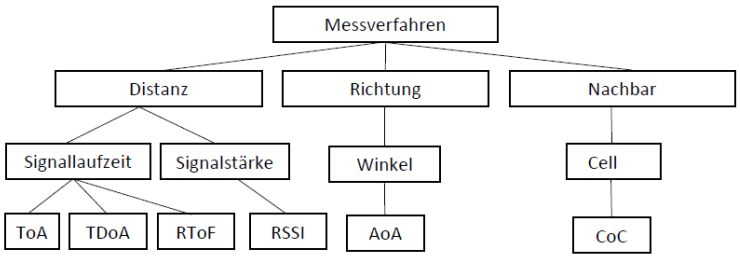


Abbildung 2.6: Überblick der Messverfahren

nen. Im Anschluss wird auf Messverfahren wie Feldstärkedämpfung und Mehrwegimpulsantwort eingegangen, die bei fehlender Sichtverbindung an gewannt werden können.

2.2.1.1 Laufzeitmessung

Die „Time of Arrival (ToA)“ Methode basiert auf der Messung der Distanz zweier Punkte (Sender und Empfänger) [RDS11, TS08]. Alle Signale breiten sich mit einer endlichen Signalgeschwindigkeit aus. Kennt man also Sende- und Empfangszeit sowie die Signallaufzeit, so kann die Entfernung zwischen Sender und Empfänger berechnet werden. Abbildung 2.7 a) verdeutlicht das ToA Verfahren. Der Sender schickt einen Frame an den Empfänger, der die exakte Sendezeit beinhaltet. Der Empfänger kann nun aus der Sendezeit, Empfangszeit und der bekannten Signalausbreitungsgeschwindigkeit c (z.B. Lichtgeschwindigkeit) die Distanz d berechnen. Da auf Grundlage der Sende- und Empfangszeit die Entfernung berechnet wird, ist eine exakte Zeitmessung von Nöten. Tritt ein Messfehler von einer Sekunde auf, würde dies eine Abweichung von 300 Metern bedeuten. Dies hat zur Folge, dass die Uhren der Sender und Empfänger eine Uhrengenauigkeit von $< 3{,}0$ ns benötigen, um eine Genauigkeit von < 1 Meter zu gewährleisten.

2.2.1.2 Laufzeitdifferenzen

Beim Time Difference of Arrival (TDoA) werden vom Sender zeitgleich zwei verschiedene Signale mit unterschiedlichen Signalausbreitungsgeschwindigkeiten versendet [RDS11, TS08]. Durch die unterschiedlichen Empfangszeiten beim Empfänger kann die Distanz d zum Sender berechnet werden. Abbildung 2.7 b) zeigt das TDoA Verfahren. Angenommen vom Sender wird ein Ultraschallsignal und ein Funksignal zeitgleich versendet. Diese Signale kommen zu unterschiedlichen Zeiten (t_0 und t_1) beim Empfänger an. Dadurch, dass die Lichtgeschwindigkeit c deutlich höher ist, als die Schallgeschwindigkeit c_0, kann die Laufzeit des Funksignals bei der Berechnung der Distanz d vernachlässigt werden [RDS11, TS08].

2.2.1.3 Eingangswinkel eines Signals

Das „Angle of Arrival (AoA)“ ist ein Verfahren, das auf einer Winkelmessung basiert, um die Position des Senders zu ermitteln [RDS11, TS08]. Hierbei wird der Eintrittswinkel der Signale ermittelt und über die gewonnenen Winkelinformationen anguliert. In welchem Winkel ein Signal empfangen wird, kann z.B. mit Hilfe eines Antennen Arrays festgestellt werden. Diese Technik wird unter anderem

in der Schiff- und Luftfahrt verwendet, um Objekte zu orten. Oftmals wird das AoA Verfahren in Verbindung mit TDoA eingesetzt, um die Messgenauigkeit weiter zu erhöhen. Abbildung 2.7 c) zeigt das AoA Verfahren mit drei Referenzpunkten (Empfänger) und einem Objekt (Sender), das geortet wird.

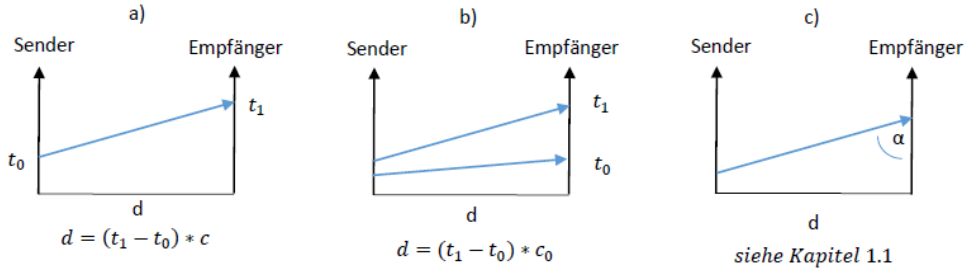


Abbildung 2.7: ToA

Signalstärke

Mit dem Verfahren „Received Signal Strength Indication (RSSI)“ wird die Position eines Objektes auf Basis der empfangenen Signalstärke berechnet [RDS11, TS08]. Der Vorteil an diesem Verfahren ist, dass dies eine reine Softwarelösung ist und keine extra Hardware benötigt. So ist der RSSI-Wert in allen handelsüblichen mobilen Geräten verfügbar und gibt eine Aussage darüber, wie groß die aktuelle empfangende Signalstärke ist. Damit eine möglichst genaue Ortung vorgenommen werden kann, benötigt auch dieses Verfahren mehrere Referenzpunkte. Die einzelnen Entfernungen zu den Referenzpunkten können leicht mit der „Frii’schen Wellengleichung“ berechnet werden [SRKZ12]. In der Praxis aber dämpfen z.B. Mauern, Bäume oder sonstige Objekte die Signale. Hierdurch wird es schwierig die aktuelle korrekte Position zu ermitteln.

2.2.1.4 Zellenreichweite eines Senders

Das „Cell of Origin (COO)“ Verfahren ermittelt auf Basis der empfangenen Netzwerkkomponenten den Standort [RDS11, TS08]. Dieses Verfahren wird unter anderem im Mobilfunk eingesetzt [TS08]. So nutzt der Mobilfunkanbieter o2 zurzeit dieses Verfahren, um dem Nutzer eine sogenannte „Homezone“ zuzuweisen, in der er vergünstigt telefonieren kann. Damit eine Ortung erst möglich ist, muss eine flächendeckende Zellstruktur vorhanden sein. Dieses Verfahren ähnelt dem RSSI-Verfahren, da auch hier die abnehmende Signalstärke bei größerer Distanz ausgenutzt wird. Ausschlaggebend für eine exakte Ortung ist bei diesem Verfahren aber die Zellgröße: Desto kleiner sie ist, desto genauer kann die Position des Objektes bestimmt werden.

2.2.2 Echtzeit-Lokalisierung

Mithilfe von sogenannten „Real-Time Locating System (RTLS)“, zu Deutsch „Echtzeit-Lokalisierungssysteme“, kann die Ortung von Personen und Objekten in abgegrenzten Arealen in Echtzeit vorgenommen werden. Diese Systeme sind in internationalen Standards definiert und durch ein Gre-

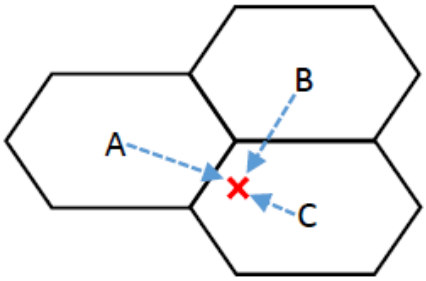


Abbildung 2.8: Cell of Origin

mium wie ISO/IEC JTC1/SC31 veröffentlicht. Im Allgemeinen besteht so ein System aus speziellen Funkkomponenten und einer Software, die es ermöglicht die Daten in Echtzeit auszuwerten. Zurzeit gibt es fünf verschiedene Technologien, auf die ein RTLS aufgebaut werden kann. Im nachfolgenden Abschnitt werden diese Technologien vorgestellt sowie deren Vor- und Nachteile erläutert.

2.2.2.1 Radio Frequency Identification

Mit der Radio Frequency Identification (RFID) Technologie ist es möglich, Daten von Gegenständen berührungslos und ohne Sichtkontakt zu lesen und zu speichern. Hierbei besteht ein RFID-System aus einem Transponder (auch Tag genannt) und einem Lesegerät (Tag-Reader). Die Lesegeräte sind dabei in der Lage, die Daten mehrerer Transponder per Funksignal zu lesen. Die Reichweite kann hierbei zwischen wenigen Zentimetern bis hin zu mehreren hunderten von Metern reichen [TS08].

Die Datenübertragung vom Transponder zum Lesegerät erfolgt über Induktion oder elektromagnetischen Wellen. Die Transponder können in zwei Kategorien eingeteilt werden: aktive und passive Transponder. Aktive Transponder erzeugen ein eigenes Signal und besitzen eine eigene Stromversorgung. Passive Transponder hingegen erhalten die notwendige Energie zur Datenübertragung aus dem Energiefeld des Lesegerätes. Hierdurch beschränkt sich die Reichweite dieser passiven Transponder auf wenige Zentimeter [TS08]. Die Genauigkeit der Ortung per RFID kann je nach technischen Aufwand bis auf wenige Zentimeter reduziert werden. So bietet die Firma Ubisense[6] zum Beispiel ein System an, dass das eine Genauigkeit von bis zu 15 cm erreichen soll.

2.2.2.2 Drahtloses lokales Netzwerk

Wireless Local Area Network (WLAN) ist eine sehr verbreitete Technologie, die vor allem im privaten oder geschäftlichen Bereich genutzt wird, um Computern, Tablets, Smartphones oder Laptops einen drahtlosen Zugang zum Intranet/Internet zur Verfügung zu stellen. Basierend auf dem vorhanden WLAN, kann ein „Real Time Locating Systems“ eingerichtet werden. Hierbei werden die Positionen der Sender z.B. über die Signalstärke (RSSI) der einzelnen WLAN-Stationen berechnet [RDS11] [Röh13]. Die Genauigkeit der Ortung wird durch die Qualität des WLAN beeinträchtigt. Liegt eine sehr gute WLAN Abdeckung vor, kann eine sehr genaue Ortung vorgenommen werden.

[6] http://ubisense.net/en/products-and-services/products/research-packages/ubisense-research-package.html

Solche Systeme werden z.B. von der Firma Ekahau[7] angeboten. Ein Vorteil solcher Systeme ist, dass keine extra Hardware angeschafft werden muss, sondern die vorhandene WLAN-Infrastruktur genutzt werden kann. Des Weiteren funktioniert die Ortung der Clients verbindungslos, sodass sich die Clients nicht an dem jeweiligen Access Point authentifizieren müssen. Die Genauigkeit der Ortung kann bis zu einem Meter genau realisiert werden [8]. Wird eine genauere Ortung gewünscht, verweist Ekahau auf die Verwendung von aktiven RFID Tags.

2.2.2.3 Sensornetzwerke

Sensornetze bestehen aus einem Zusammenschluss von mehreren Sensorknoten. Diese Sensorknoten kommunizieren untereinander, z.B. durch ein aufgebautes Ad-Hock Netzwerk. Ein Wireless Sensor Network (WSN) ist ein Sensornetzwerk, das die Kommunikation der einzelnen Sensorknoten über ein aufgespanntes WLAN realisiert. Diese Art von Sensornetzwerken werden unter anderem bei Umweltbeobachtungen, in medizinischen Systemen oder in der Robotik eingesetzt. Im Bereich der Umweltbeobachtungen werden die Sensornetzwerke z.B. eingesetzt, um Umwelteinflüsse wie Temperatur, Lichteinfall oder Strahlungen zu messen.

2.2.2.4 Global Navigation Satellite System

Mithilfe von Satelliten können Positionsbestimmungen sowie Navigationen durchgeführt werden. Global Navigation Satellite System (GNSS) ist hierbei ein Sammelbegriff für verschiedene Satellitensysteme wie [TS08]:

- GPS der Vereinigten Staaten von Amerika
- GLONASS (Global Navigation Satellite System) der Russischen Föderation
- Galileo der Europäischen Union

Letzteres befindet sich zurzeit noch im Aufbau und soll in wenigen Jahren zur zivilen Nutzung noch genauere Positionsdaten als die System GPS oder GLONASS liefern. Die einzelnen Satelliten teilen über Funkcodes ihre genaue Position sowie Uhrzeit mit. Damit ein Empfänger eine Positionsbestimmung durchführen kann, muss dieser mindestens vier Satelliten gleichzeitig empfangen. Im Empfangsgerät werden dann die Signallaufzeiten gemessen und daraus die aktuelle Position ermittelt [TS08].

2.2.3 Zusammenfassung

Zusammenfassend wurden in diesem Abschnitt verschiedene Techniken vorgestellt, mit denen eine Positionsbestimmung vorgenommen werden kann. Neben den einzelnen Techniken zur Positionsbestimmung wurden auch einzelne Techniken erläutert, mit denen ein RTLS aufgebaut werden kann. Je nach Einsatzgebiet eignen sich unterschiedliche Techniken und erfordern eine gründliche Evaluation. Dadurch, dass die Satellitensysteme GPS, GLONASS oder Galileo auf den Outdoor-Bereich ausgelegt sind, ermöglichen diese Systeme keine bzw. eine sehr schlechte Indoor-Ortung [TS08]. Mit

[7] http://www.ekahau.com/real-time-location-system/solutions
[8] http://www.ekahau.com/userData/ekahau/documents/case-studies/SAMC_case_study_letterpdf

Bezug zur Liveanalyse im Sport ist ein Einsatz dieser Systeme aus diesem Grund nicht sinnvoll und würden sehr ungenaue Messergebnisse liefern. Des Weiteren wurden die beiden Triangulationsverfahren Angulation und Lateration vorgestellt. Das Laterationsverfahren hat gegenüber dem Angulationsverfahren den großen Vorteil, dass dieses technisch einfacher umsetzbar ist und sich durch weitere Referenzpunkte die Genauigkeit der Positionsbestimmung erhöhen lässt (Multilateration). Um die bestmögliche Technik für die Projektgruppe auswählen zu können, müssen weitere Evaluationen durchgeführt werden. Die beiden Techniken „RFID“ und „WLAN“ stellen sich bis jetzt als sehr vielversprechend heraus und sollten genauer betrachtet werden. So setzt die Firma Zebra Technologie z.B. RFID sowie WLAN ein, um Lagerbestände oder Fahrzeuge (z.B. Gabelstapler) in einer Halle zu orten [Cor14]. Aber auch im Bereich Sport werden diese Techniken bereits aktiv eingesetzt. So trainiert der FC Bayern München mit einem RTLS von der *inmotiotec GmbH*, die bereits 20 Minuten nach dem Training eine weitreichende Analyse zu jedem Spieler bereit stellt [Gmb14]. Der nachfolgende Abschnitt soll noch mehr Bezug zum Sport herstellen und einige Werkzeuge näher erklären, mit denen es möglich ist, Sportanalysen durchzuführen.

2.3 Werkzeuge für Sportanalyse

Es gibt viele verschiedene Werkzeuge für Sportanalysen, die sich in ihrem Funktionsumfang und dem Einsatz von unterschiedlichen Technologien sehr stark unterscheiden. Ein Werkzeug für Sportanalysen soll im Sportbereich den Akteuren auf elektronischen Wege helfen, die zu verarbeitenden Informationen in informationelle Mehrwerte umzuwandeln. Im Verlauf dieses Teilkapitels sollen unterschiedliche Werkzeuge näher betrachtet werden. Zusätzlich werden Marktführer in diesem Bereich detailliert beschrieben, aber auch Nischenprodukte werden behandelt, um eine möglichst umfangreiche und repräsentative Sicht auf den Markt zu vermitteln. Insgesamt soll dieser Überblick dann zum Schluss dazu verwendet werden, um Kriterien für die zu erarbeitende Software aufzustellen. Durch die starke Heterogenität des Marktes wird im Folgenden versucht, die Übersicht zu verbessern, indem die Unterschiede der Werkzeuge durch passende Kriterien verglichen werden.

2.3.1 Kriterien für den Vergleich von Werkzeugen

Zunächst sollen die verschiedenen Werkzeuge verglichen werden, um einen Überblick über den Markt geben zu können. Damit die Werkzeuge vergleichbar sind werden in diesem Kapitel verschiedene Kriterien erarbeitet. Diese Kriterien sollen die Werkzeugeigenschaften möglichst vergleichbar machen. Dennoch sollen möglichst wenig Aspekte des Werkzeugs verloren gehen, also ein möglichst kompletter Überblick gegeben werden. Ziel dieser Kriterien ist der Aufbau einer Tabelle in der ein Überblick über den aktuellen Funktionsumfang und Technologieeinsatz der Werkzeuge auf dem Markt gegeben werden kann.

- Name und Hersteller des Werkzeugs

  Zur Identifikation des Werkzeugs.

- unterstützte Sportarten

  Spezialisierung auf diese Sportarten

- unterstützte Personenanzahl

Vom Werkzeug können x Personen verfolgt werden.

- eingesetzte Technologien

  Aufzählung der eingesetzten Technologien.

- Funktionen

  Funktionsumfang des Werkzeugs wird beschrieben.

- Systemkomponenten

  Benötigte Komponenten.

- Sonstige Eigenschaften

2.3.2 Klassifikation von Werkzeugen für Sportanalysen

Anschließend werden die Werkzeuge für Sportanalysen in verschiedene auf den Kriterien aufbauende Klassen unterteilt. Diese Unterteilung geschieht hinsichtlich ihrer Technologie, da hier eine erste Unterteilung sinnvoll ist um ganzheitlich den Markt der Werkzeuge zu beschreiben. Diese Beschreibung soll einen möglichst umfangreichen und repräsentativen Überblick über die breit diversifizierte Produktlandschaft geben. Es folgen die Klassen in ungeordneter Form.

- Videoanalyse Bei dieser Analyseklasse wird Videomaterial anhand von verschiedenen Kriterien analysiert und dem Trainer zur Verfügung gestellt. Hierdurch kann der Trainer rapide Spielzüge in einer Diskussion, Schritt für Schritt abspielen und kritische Moment genau beschreiben und Verbesserungsvorschläge geben. Genaueres wird in den einzelnen Details der verschiedenen Werkzeuge beschrieben.

- funkbasierte Analysen Funkbasierte Analysen, die Personen und Objekte in Echtzeit mit hoher Genauigkeit lokalisieren, verwenden Sensoren, z.B. am Spieler oder im Ball, um die Standorte und Standortveränderungen(Geschwindigkeiten/Beschleunigungen) der Objekte zu messen. Die funkbasierte Technologie bietet einen wesentlichen Vorteil gegenüber videobasierten Trackingsystemen, es können auch optisch verdeckte Objekte problemlos erkannt werden.

- ortsbasierte Analysen Ortsbasierte Analysen werden durch Sensoren zur Standortbestimmung unterstützt. Diese Analysen werden oft durch GPS und mit mobilen Endgeräten umgesetzt. Dabei werden zurückgelegte Strecken in Verbindung mit ermittelten Zeit zur Kalkulation verwendet. Die Sensoren des GPS sind allerdings deutlich ungenauer als die Sensoren bei funkbasierten Lösungen.

2.3.3 Werkzeuge für Sportanalysen

Im Folgenden werden die Analysewerkzeuge detailliert beschrieben. Diese Beschreibung wird unterteilt nach den in Abschnitt 2.3.2 aufgestellten Klassen zur Verbesserung der Übersichtlichkeit.

2.3.3.1 Beschreibung der Werkzeuge zur Videoanalyse

Die Werkzeuge zur Videoanalyse bieten viele verschiedene Vorteile, so müssen meist keine aufwendigen Vorbereitungen getroffen werden und kein teures Spezial-Equipment gekauft werden. Die Features unterscheiden sich stark. Ein Überblick wird in diesem Abschnitt gegeben.

KeeMotion

Das Videoanalysetool von *KeeMotion*[9] zeichnet Spiele auf und kann definierte Ereignisse während des Videos erkennen und Tags an diejenigen Stellen legen. Z.B. wenn während eines Basketballspiels eine Spieler wirft oder punktet. Hierdurch werden sowohl Trainer wie auch Produzenten und TV-Sender unterstützt. Sie haben ihre Informationen direkt griffbereit und Echtzeitdaten zum aktuellen Spiel.

Technologie:

KeeMotion verwendet eine Videoanalyse um die Daten zu generieren. Erfasste Events müssen von Keemotion angelegt werden, können also nicht vom Trainer oder weiteren Benutzern angelegt werden. Dies könnte zur Problemen führen, da nicht alle über ein gleiches Wissensniveau in allen Bereichen verfügen. Hier ist eine gute Kommunikation wichtig.

Funktionen:

Durch vordefinierte Events werden Tags an eine Timeline des Videos gehängt. Dies kann akkumuliert werden und so zum Erkenntnisgewinn führen. So können zum Beispiel Spieleranalysen während eines Spiels aufgezeichnet werden oder auch mit Anderen verglichen werden.

Systemkomponenten:

Reine Videoanalyse, die nur Kameras und eine Rechnernetz benötigt und Ergebnisse auf verschiedenen Endgeräten wie Tablets anzeigen kann.

Synergy Sports Technology

Synergy Sports Technology (http://corp.synergysportstech.com/) hat sich auf den Basketball spezialisiert und bietet in diesem Bereich ein Werkzeug das vorallem Trainer unterstützen soll. Spieler können verglichen werden und durch gezielte Videoanalysen trainiert werden. Synergy Sports Technology arbeitet mit allen NBA-Teams und mit mehr als 150 internationalen Teams zusammen. Die Funktionen sind ähnlich der von KeeMotion.

Dartfish

Dartfish[10] ist eine Software zur Analyse von Bewegungen und wird von den amerikanischen Volleyballmannschaften verwendet. Hierbei wird der Fokus auf einzelne Bewegungsabläufe gelegt, die dann optimiert werden sollen.

Technologie:

[9] http://www.keemotion.com/
[10] http://www.dartfish.tv/DartfishExpress.aspx

Dartfish setzt auf Videoanalyse, dabei können bis zu zwei Kameras eingesetzt werden. Auf den erstellten Video werden dann verschiedene Analysen gemacht.

Funktionen:

Dartfish bietet die Möglichkeit, Schlüsselpositionen genau zu betrachten und Bild für Bild abzuspielen. Diese Bild-für-Bild Wiedergabe kann abgemessen werden und untersucht, warum dieser Bewegungsablauf optimiert werden muss und wie dies am Besten geschieht. Ein Funktion von Dartfish sind hierbei die Zeichentools, die es dem Trainer ermöglichen illustrierte Änderungen zu vermitteln. Mit Dartfish können auch markierte Objekt automatisch verfolgt werden und Daten zu diesen erfasst werden.

Systemkomponenten:

Dartfish besteht aus bis zu zwei Kameras und einem Server zur Auswertung. Diese kann auf verschiedenen Endgeräten angeschaut und bearbeitet werden.

Kostenlos/OpenSource KINOVEA

Kinovea[11] ist eine kostenlose Opensource-Software zur Videoanalyse aus Frankreich. Kinovea bietet eine intuitive Benutzungsoberfläche (siehe Abbildung 2.9). Das Werkzeuge ist sehr mächtig, Details im Abschnitt Funktionen. Finanziert wird die Software von der Community, sowohl durch monetäre Beiträge, wie auch durch Programmierung weiterer Features.

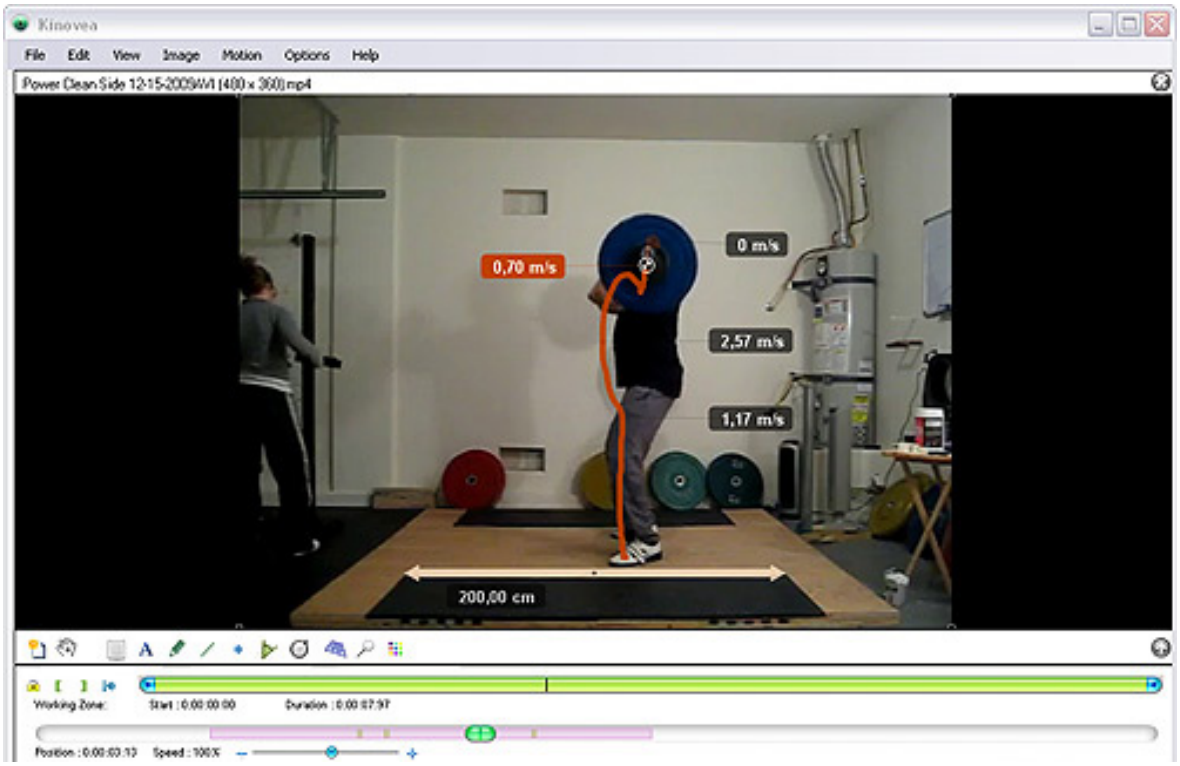


Abbildung 2.9: Intuitives User Interface von KINOVEA - [Kin13]

[11] http://www.kinovea.org/

Technologie:

KINOVEA kann verschiedene Videoformate verarbeiten, aber auch Bilddateien. Diese werden auf einem Rechner verarbeitet.

Funktionen:

KINOVEA unterstützt viele Videobearbeitunsfunktionen, wie z.B. Anpassung der Helligkeit, aber auch viele Funktionen eines Videoplayers, zum Beispiel Verlangsamen des Videos. Es kann sogar Bild für Bild angezeigt werden. Weiter können verschiedene Zeichnung gemacht werden, z.B: Winkelanzeigen oder Strecken. In Kombination mit der automatischen Verfolgung von Objekten können hier automatisiert Erkenntnisse gewonnen werden. KINOVEA bietet verschiedene Analysen um Geschwindigkeiten und Beschleunigung zu erkennen. Das Werkzeug bietet auch Möglichkeiten die Analysen zu teilen und verschiedene Exportmöglichkeiten.

Systemkomponenten:

KINOVEA unterstützt Hochgeschwindigkeitskameras, kann aber auch mit einer normalen Kamera verwendet werden. Die Verarbeitung findet auf einem Rechner statt.

2.3.3.2 Beschreibung der Werkzeuge zur funkbasierten Analyse

Die funkbasierte Technologie bietet einen wesentlichen Vorteil gegenüber videobasierten Trackingsystemen, sie kann optisch verdeckte Objekte problemlos orten.

RedFIR

RedFIR ist ein durch das Fraunhofer-Institut für integrierte Schaltungen (IIS) entwickeltes Konzept zur Sportanalyse, dass auf Basis von Funktechnologien die Positionsdaten von Spielern und Objekten erkennt und dieser Ereignis-basiert verarbeitet. Promotet wurde das Werkzeug auf der Cebit 2014 von Oliver Bierhoff. RedFIR basiert auf der SAP HANA-Technologie. RedFIR wurde bis jetzt vor allem im Fußballkontext eingesetzt. Dabei arbeitet das Frauenhofer IIS z.B. mit dem 1. FC Nürnberg einen Verein der 1. Bundesliga zusammen.

Das Frauenhofer IIS hat mit GOALref auch eine Technologie zur Erkennung ob ein Objekt bestimmte Bereiche passiert hat, z.B. die Torlinie beim Fußball.

Technologie:

RedFIR verwendet den 2,4GHz Band-Standard und ist somit ohne Lizenzgebühren einsetzbar. (Im Gegensatz zu, z.B. Radiofrequenzen oder Ähnlichem.)

Funktionen:

Miniatursender an Personen und Objekten senden Funksignale aus, welche von Empfangsantennen erfasst werden. Diese Antennen sind um das zu beobachtende Areal gleichmäßig verteilt. In den Antennen werden die Empfangszeiten protokolliert und so Rückschlüsse auf die Position der Personen oder Objekte gezogen. Das angeschlossene Rechnernetz kann die ermittelten Positionen hinsichtlich einer Ereigniserkennung auswerten.

Systemkomponenten:

Es werden Sender für Spieler und Ball, z.B. in Schuhen oder Spielerkleidung eingesetzt, diese Sender sind sehr robust und stoß-/wasserfest. Die Akkus der Sender werden drahtlos aufgeladen und haben eine Laufzeit von 3-4 Stunden.

Abbildung 2.10: Eine Empfängereinheit von RedFIR - [Ins14]

Empfängerantennen die rund um das zu überwachende Areal aufgestellt werden müssen. Empfänger sind zentralisiert Vernetzt. In der zentralen Einheit erfolgt die Synchronisierung der Elemente. Die Berechnung der Positionen wird auch von dieser zentralen Einheit erledigt. Dieser Empfänger ist in Abbildung 2.10 auf Seite 25 abgebildet.

Diese zentralisierte Einheit ist ein Rechnernetz das aus mehreren vernetzten Standardservern auf Linux-Basis basiert.Dieses Rechnernetz ermöglichst die Darstellung der Positionen auf einer 3D-Oberfläche, z.B. Spieler auf einen Fußballfeld.

Abbildung 2.11: Spieler mit Statistiken bei RedFIR - [BH14]

Zur Ereigniserkennung gibt es einen Eventobserver. Dieser überwacht bestimmte typische Ereignisse. Im Fußballkontext zum Beispiel den Ballbesitz, Passen, Flanken, Torschuss und weitere. Die

ermittelten Daten werden in einer Datenbank persistent gespeichert. Der Eventobserver dient der Erkenntnisgewinnung. Dieser Erkenntnisgewinn ist anhand eines Fußballspieler exemplarisch abgebildet in Abb. 2.11.

Prozone

Prozone bietet eine große Vielfalt an Softwareprodukten zur Sportanalyse an. Hierbei verwendet Prozone verschiedene Technologien und versucht hierdurch eine höhere Genauigkeit als die Konkurrenten zu erlangen. Prozone wurde 1995 gegründet und hat sich 2011 mit Amisco zusammengelegt. Prozone bietet ein breites Repertoire an Partnern, darum sind im Fußball Ajax Kapstadt, Arsenal London, Chelsea, aber auch deutsche Vereine wie Bayer Leverkusen und der FC Bayern München arbeiten mit Prozone zusammen. Zu den Partnern von Prozone zählt auch der Deutscher Fußball Bund (DFB) insgesamt. Es hat allerdings nicht nur Analysemethoden für Fußball, sondern ist auch im Rugby-Bereich angesiedelt. In diesem Bereich werden verschiedene australische und europäische Vereine als Partner genannt.

Technologie:

Prozone verwendet eine hohe Bandbreite an Technologien zur Generierung ihrer Daten. Es werden Herzfrequenzen, sowie Beschleunigung und Position durch verschiedene Sensoren gemessen.

Funktionen:

Es werden Sensoren für die Positionsbestimmung, Herzfrequenz und Videoanalysen verwendet. Die verwendeten Verfahren werden nicht näher beschrieben.

Systemkomponenten:

Die Analyse von Prozone kann ich vier Bereiche unterteilt werden. Die Performance Analyse, die physikalische Beobachtung, die Spielerrekrutierung und den Forschungs- und Beratungsbereich.

Die Performance Analyse bietet die Möglichkeiten der Echtzeitanalyse von Spielsituationen sowohl in den eigenen Reihen wie auch bei Gegner. Diese Analyse stellt die Stärken und Schwächen der Mannschaften dar. Des Weiteren können die Gegner analysiert werden und diese Analyse vor dem Match zur Vorbereitung verwendet werden. Die Performance Analyse zeigt außerdem die gemessene technische(z.B. fußballerische) und physische Performance der Mannschaften an. Dies ist der Versuch die Subjektivität aus den Mannschaftsanalysen zu minimieren und objektive Kennzahlen zu erhalten.

Die physikalische Beobachtung bringt Features zur Analyse der physischen Fitness der Spieler. Es werden sowohl die Herzfrequenz wie auch Beschleunigung und Wegstrecke der Spieler zur Berechnung eines Fitnesslevel herangezogen. Mit dieser Technologien können auch verletzte Spieler in das Spiel integriert werden und Belastungen objektiv gemessen werden. Sollten dann Überbelastungen oder rapide Abfälle der Fitness erkannt werden kann direkt eingegriffen werden, um weiteren Verletzungen vorzubeugen.

Prozone bietet ein Werkzeug zur Spielerrekrutierung in der Spieler objektiv beobachtet werden und weltweit im Vergleich stehen. Hierdurch können Spieler aus der ganzen Welt akquiriert werden. Außerdem werden Subjektivitäten aus unterschiedlichen Länder vorgebeugt. Es können auch ausgeliehene Spieler auf Fortschritte überprüft werden.

Letzte Komponente dieses Systems ist die Forschung und Beratung. Hier werden Analysen über bestehende Daten gemacht und hierdurch neue Erkenntnisse gewonnen. Es werden aber auch Analysemethoden erklärt, die vielleicht zu weiteren Erkenntnissen führen könnten und bereits eingesetzte Analysen werden verbessert.

2.3.3.3 Beschreibung der Werkzeuge zum Geolocation Tracking

Hier wird nur eine Ausprägung der Werkzeuge detailliert besprochen und abgegrenzt zu weiteren, da diese Klasse nicht zum Kerngeschäft gehört aber zur späteren Abgrenzung dient. In diesem Gebiet gibt es sehr viel Software, vor allem für den Einzelsportbereich (z.B. das Laufen). Runtastic wird hier als Systemvertreter beschrieben, andere Softwareanbieter unterscheiden sich nur marginal von dem Funktionsumfang.

runtastic

Runtastic[12] ist eine Software für mobile Endgeräte, die das GPS-Modul nutzt um Positionsangaben über den Nutzer machen zu können. Diese Positionsdaten werden gespeichert und zu einer Zeitlinie hinzugefügt. Auf Grund dieser Zuweisung werden Einschätzungen über Geschwindigkeiten und Beschleunigungen getroffen. Runtastic bietet weitere Services auf diesen Daten an, die unter Funktionen detaillierte erklärt werden.

Technologie:

Runtastic verwendet das GPS-Modul von mobilen Endgeräten. Außerdem ist Runtastic eng mit Facebook und Google+ verknüpft um dem Sport eine soziale Komponenten zugeben und eine Vergleichbarkeit über soziale Netzwerke zu ermöglichen. So können hier z.B. Trainingsstrecken, sowie Trainingshäufigkeit publiziert werden. Die Analyse der Daten findet in einem Webinterface statt, dies kann vom mobilen Endgerät wie auch von einem Rechner aus passieren.

Funktionen:

Runtastic bietet vor allem Mehrwerte durch die Analyse der Positionsdaten und ihre Zuordnung zu Zeitpunkten. Aufbauend auf dieser Zuordnung werden Laufstrecken visualisiert und weitere Erkenntnisse gewonnen. So können verschiedene Trainingstage eines Sportlers verglichen werden, aber auch Vergleiche zwischen Sportlern gemacht werden.

Systemkomponenten:

Runtastic versucht möglichst auf Systemkomponenten aufzubauen die beim Benutzer schon vorhanden sind. So zum Beispiel das GPS-Modul des mobilen Endgeräts oder auch die Präsentation im Webinterface.

2.3.4 Zusammenfassung

Das Eventobserver Pattern von RedFIR hat viele Ansätze, die zur Umsetzung einer Liveanalyse mit Odysseus P2P genutzt werden können. Es werden hier Events an einen zentralen Server geschickt und diese werden im Log erfasst. Dieser Log wird dann kontinuierlich zur Erkenntnisgewinnung verarbeitet. Events werden von den einzelnen Sendeeinheiten geschickt und im Log persistent ge-

[12] https://www.runtastic.com/de

speichert. Danach würden sich anderen Speichermethoden besser eignen, dennoch gehen hier keine Daten verloren.

Der Marktüberblick zeigt, dass die Werkzeuge zur Sportanalyse eine breites Spektrum an Sportarten und Funktionsumfängen abdecken. Diese Funktionen sollten als Grundlage zur Entwicklung einer geeigneten Software im Projekt betrachtet werden.

2.4 Datenströme

Herkömmliche Systeme speichern anfallende Daten persistent ab und bieten in der Regel jederzeit einen Zugriff auf diese Daten. Hierbei kommen vorwiegend Datenbanksysteme zum Einsatz, in denen die Daten strukturiert vorliegen und bei Bedarf abgerufen oder manipuliert werden können. Eine wichtige Eigenschaft dieser Daten ist darüber hinaus, dass jedes einzelne Datum wichtig und nicht vergänglich sind, also auch noch in mehreren Jahren relevant ist. Dies können zum Beispiel Unternehmensdaten für das Finanzamt sein, die über einen Zeitraum von zehn Jahren sicher aufbewahrt werden müssen.

Allerdings gibt es auch andere Anwendungsbereiche, in denen große Datenmengen beispielsweise durch Sensoren generiert werden. Die einzelnen Daten sind jedoch hierbei nicht immer wichtig und darüber hinaus eventuell auch vergänglich, sodass eine Speicherung teuer und nicht notwendig wäre. Oftmals reicht es auch aus, die Daten eines Zeitraumes zu komprimieren und den Mittelwert abzuspeichern oder nur die Daten eines festgelegten Zeitraumes zu betrachten. Vielfach ist es auch erforderlich, dass auf Änderungen der Daten schnell und nahezu in Echtzeit reagiert werden kann. Dabei kann es sich zum Beispiel um Sensoren an Brücken oder Induktionsschleifen handeln, die das jeweilige Verkaufsaufkommen beobachten und bei Staugefahr den Verkehr ausbremsen. Insgesamt gibt es eine Vielzahl von Anwendungsbereichen, in den die herkömmliche Datenspeicherung nicht sinnvoll und sehr teuer ist. Aus diesem Grund ist es von Vorteil, Datenströme in Echtzeit zu analysieren und Muster in diesen Daten zu erkennen (Pattern Matching) bzw. komprimierte Daten abzuspeichern. Ähnlich wie bei Datenbanken gibt es auch bei Datenströmen speziell angepasste Systeme, die eine solche Verarbeitung ermöglichen und darüber hinaus Optimierungen zur Laufzeit ermöglichen. DSMS kommen beispielsweise im elektronischen Handel, bei der Fertigungssteuerung, dem Energiemanagement oder in der Logistik zum Einsatz.

Ein anderer Anwendungsfall ist die Echtzeitanalyse im Sport, bei der die Bewegungen der Spieler auf dem Spielfeld in Echtzeit analysiert werden. Eine zentrale Herausforderung dabei sind die extrem großen Datenmengen, da die verwendeten Sensoren der Spieler bis zu 200 Positionen pro Sekunde senden und die des Balles sogar bis zu 2000 Positionen pro Sekunde (RedFIR) [Grü13a]. Darüber hinaus sollen dem Trainer in Echtzeit Auswertungen angezeigt werden, sodass eine Speicherung und Auswertung der Daten im Anschluss an das Spiel nicht sinnvoll ist.

2.4.0.1 Charakteristika von Datenströmen

Datenströme unterscheiden sich in ihren Eigenschaften von der herkömmlichen Datenverarbeitung in folgenden Punkten [Krä07]:

- **Von einer aktiven Quelle / Sensor** - Die Daten werden in der Regel nicht von der Datenquelle abgefragt, sondern aktiv durch die Quelle oder den Sensor gesendet. Dies ist einer der Hauptunter-

schiede im Vergleich zu bekannten Datenbanksystemen, da dort die Daten explizit abgerufen werden. Es gibt allerdings auch Anwendungsfälle bei denen die Daten kontinuierlich abgefragt und im Anschluss weiterverarbeitet werden. In beiden Fällen handelt es sich jedoch um eine Datengetriebene Verarbeitung. Bezogen auf das zuvor beschriebene Sportbeispiel bedeutet das, dass die Sensoren der Spieler oder des Balls die Position eigenständig mitteilen und nicht explizit angefragt werden müssen.

- **Zu beliebigen Zeitpunkten** - Die Daten eines Sensors müssen nicht durchgängig oder in regelmäßigen Abständen geliefert werden. Beispielsweise kann das GPS-Signal eines Fahrzeugs durch Tunnel oder andere örtliche Gegebenheiten verfälscht oder ganz unterbrochen werden. Die unterschiedliche Intensität, in der die Daten verarbeitet werden müssen, erfordert in verteilten Szenarien eine dynamische Lastverteilung auf andere Knoten zur Laufzeit.

- **In Echtzeit übersandt** - Die ankommenden Daten werden in Echtzeit übertragen. Auf diese Weise ist eine direkte Reaktion auf Datenveränderungen möglich, wie es zum Beispiel bei der Patientenüberwachung im Krankenhaus notwendig ist. Aber auch der Trainer einer Sportmannschaft kann so auf Veränderungen direkt reagieren und gegebenenfalls Spieler auswechseln.

- **Zeitlich geordnet** - Ankommende Daten sind in der Regel in einer zeitlichen Ordnung. Ein nachfolgendes Datenpaket hat also einen höheren Zeitstempel als vorherige Datenpakete. Viele Verarbeitungskonzepte basieren auf dieser zeitlichen Ordnung. Da es jedoch auch Quellen gibt, die die Daten nicht geordnet senden, gibt es verschiedene Möglichkeiten den Datenstrom nachträglich in die korrekte Reihenfolge zu bringen [Krä07], auf die hier nicht weiter eingegangen wird. Bei den Bewegungsdaten von Sensoren kann somit auch der zurückgelegte Weg des Spielers oder Balls bestimmt werden.

- **Potentiell unendlich** - Neben dem aktiven Senden von Daten durch die Quellen liegt ein weiterer Hauptunterschied darin, dass der genutzte Datenstrom potentiell unendlich ist. Das ist dadurch begründet, dass die Quellen kontinuierlich Daten liefern und zum aktuellen Zeitpunkt nicht ersichtlich ist, wann bzw. ob der Datenstrom überhaupt endet.

2.4.1 Herausforderung: Potentiell unendlich

Bei verschiedenen Operatoren, wie beispielsweise einem Join oder einer Aggregation, wird der gesamte Datenbestand benötigt, der jedoch aufgrund der potentiellen Unendlichkeit niemals vorhanden ist [Krä07]. Ein einfacher Nested-Loop-Join iteriert über die erste Datenquelle und vergleicht alle Elemente mit der anderen Datenquelle. Da die Datenströme jedoch potentiell unendlich sind, besteht hier die zentrale Problematik darin, dass es zu einer Blockierung kommen würde. Hier können als Lösung spezielle Verbundalgorithmen eingesetzt werden, die die eingehenden Daten zwischenspeichern und wieder freigeben, wenn diese nicht mehr benötigt werden [Gra12]. Allerdings gibt es auch hier wieder die Problematik, dass nicht eindeutig festgestellt werden kann, ob ein Element des Caches nicht mehr benötigt wird. Da der Speicherplatz begrenzt ist, würde in diesem Fall der Join-Speicher überlaufen. Aus diesem Grund wird bei der Datenstromverarbeitung in der Regel das Gesamtergebnis nur angenähert und somit mit approximativen Antworten gearbeitet. Die beiden grundlegenden Ansätze sind hierbei zum einen, dass eine Zusammenfassung aller verarbeiteten Daten betrachtet wird (Verdichtungsansatz). Bei diesem Ansatz werden die Daten eines Zeitraumes komprimiert, sodass nicht alle Elemente im Hauptspeicher vorhanden sein müssen. Dieser Ansatz wird in dieser Ausarbeitung

jedoch nicht weiter erläutert. Stattdessen wird der zweite Ansatz, bei dem nur ein Ausschnitt aus dem bestehenden Datenstrom betrachtet wird, genauer beschrieben (Fensteransatz).

2.4.1.1 Fensteransätze

Beim Fensteransatz wird nur ein kleiner Ausschnitt des Datenstroms betrachtet und ältere Elemente verworfen. Der Fensteransatz kann mit dem Fenster in einem Zug verglichen werden, da der Reisende auch dort zu einem Zeitpunkt nur einen kleinen Ausschnitt der Landschaft betrachten kann und andere Bereiche nur durch die Bewegung des Zuges sichtbar werden. Analog dazu wird bei der Datenstromverarbeitung ein Fenster auf den Datenstrom gelegt und zu einem Zeitpunkt nur dieser Ausschnitt betrachtet. Es gibt eine Vielzahl verschiedener Möglichkeiten, Fenster zu definieren, die jedoch von dem jeweiligen Anwendungsfall abhängen.

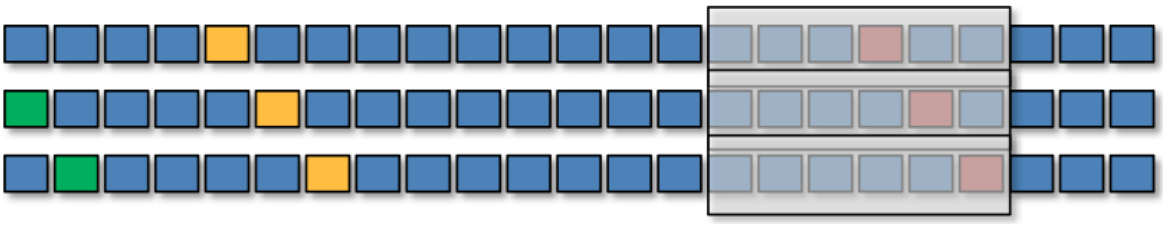


Abbildung 2.12: Sliding Window [Gra12]

Es gibt verschiedene Fensterarten, die festlegen wie sich die Endpunkte des Fensters verhalten. In dieser Ausarbeitung werden im Folgenden nur Sliding-Windows betrachtet, bei denen sich sowohl der Anfangs- als auch der Endpunkt des Fensters bewegen, da diese nicht zu einem Speicherüberlauf führen können und darüber hinaus auch neu eingehende Daten berücksichtigen. Ein Sliding-Window wird beispielhaft in Abbildung 2.12 dargestellt. Ein entscheidender Faktor bei der Definition von Fenstern ist die Größe oder auch Breite w. Die Breite des Fensters kann auf unterschiedliche Weise festgelegt werden [Krä07]:

- Es wird der **Zeitraum** festgelegt, in dem ein Datenelement betrachtet wird. Beispielsweise die durchschnittliche Laufgeschwindigkeit des Spielers A in den letzten 2 Minuten. Es werden in diesem Beispiel nur die Elemente berücksichtigt, die nicht älter als 2 Minuten sind.
- Neben der Zeit kann auch die **Anzahl an Elementen** angeben werden. Dabei wird festgelegt, wie viele Elemente zu einem Zeitpunkt in einem Fenster sind. Beispielsweise werden die letzten 100 Positionsdaten eines Spielers betrachtet.
- **Strominhalte** - Die Inhalte der Elemente beschreiben selber, welche Breite ein Fenster hat. Dabei gibt es einen Auslöser, der den Startpunkt des Fensters beschreibt und einen Auslöser für den Endpunkt. Beispielsweise die Positionsdaten eines Spielers, während dieser auf dem Spielfeld ist und nicht auf der Bank sitzt. Problematisch ist hierbei allerdings, wenn der Endpunkt nie oder sehr spät erreicht wird, da dann große Mengen im Speicher belegt werden und Berechnungen unter Umständen sehr aufwendig sind und somit lange dauern können.

In Verbindung mit der Breite des Fensters ist darüber hinaus auch wichtig, wann sich die Endpunkte des Fensters ändern, also nach wie vielen Takten/Elementen n. Bei einer Taktrate von $n = 1$

bewegt sich das Fenster bei jedem neu eingehenden Element. Somit kommen Elemente in mehreren Fenstern vor. Ein Beispiel hierfür wäre die Durchschnittsgeschwindigkeit der letzten 2 Minuten. Bei einem Jumping-Window mit $1 < n < w$ kommen die Elemente ebenfalls in mehreren Fenstern vor, allerdings verändert sich die Position des Fensters nicht bei jedem neu eingegangenen Element. Ein Beispiel hierfür wäre die Durchschnittsgeschwindigkeit der letzten 2 Minuten in Minuten-Intervallen. Sollen Elemente nur ein einziges Mal berücksichtigt werden, kann ein Tumbling-Window verwendet werden. Dabei ist n gleich der Breite des Fensters, sodass der Datenstrom in gleich große Abschnitte unterteilt wird. Dies kann beispielsweise für die Durchschnittsgeschwindigkeit in 2 Minuten-Intervallen genutzt werden [Gra12].

Im Folgenden werden nun zwei Ansätze vorgestellt, wie Fenster technisch realisiert werden können.

Intervall-Ansatz

Beim Intervall-Ansatz besitzt jedes Stromelement einen Start- und einen Endzeitstempel, anhand dessen festgestellt werden kann, ob dieses noch gültig ist. Beim Eintreffen im System wird dem Element ein Startzeitstempel hinzugefügt. Das kann entweder die Transaktionszeit sein oder ein existierender Stempel, der durch die Quelle hinzugefügt wurde. Bei letzterem müssen die Uhren der Quellen synchronisiert werden, damit es nicht zu Problemen kommt. Das Ende des Intervalls ist beim Eintreffen zunächst noch offen und wird durch den Fensteroperator individuell gesetzt. Entscheidend ist hier vor allem auch, um welche Art von Fenster es sich handelt. Die jeweiligen nachfolgenden Operatoren müssen dann entscheiden, ob das Element verarbeitet oder aus dem Strom entfernt wird.

Ein Ripple Join mit SweepArea speichert sich alle Elemente der beiden Eingangsdatenströme in SweepAreas, solange die Elemente gültig sind. Beim Eingang eines neuen Elements aus dem einen Strom wird die SweepArea des anderen Stroms zunächst aufgeräumt, indem alle Elemente entfernt werden, die nicht mehr zu einem Treffer führen können, also deren Gültigkeitsintervalle sich nicht mehr überschneiden. Danach werden potentielle Join Partner gesucht und vereint. Das Gültigkeitsintervall des kombinierten Elements ist die Schnittmenge der beiden Ursprungsintervalle. Beim Weiterleiten der Elemente wird darüber hinaus darauf geachtet, dass die zeitliche Reihenfolge der Elemente erhalten bleibt, da dies eine wichtige Eigenschaft des Datenstroms ist. Die Verarbeitung eines Joins ist also nur durch den Einsatz von Fenstern möglich, allerdings kann es passieren, dass potentielle Join-Partner nicht gefunden werden, da sich deren Fenster nicht überschneiden [Krä07].

Positiv-Negativ Ansatz

Beim Durchlaufen des Elementes durch den Fenster-Operator wird bei diesem Ansatz ein Plus-Marker und ein Startzeitstempel angehängt. Der Fensteroperator speichert sich das Element auf einer Merkliste. Bei jedem neu ankommenden Element wird dieses entsprechend weitergeleitet, zwischengespeichert und alle Elemente in der Merkliste überprüft, ob diese noch aktuell sind. Ist das nicht mehr der Fall, werden diese mit einem Minus-Marker und dem Endzeitstempel erneut weitergeleitet. Alle nachfolgenden Operatoren können auf diese Weise feststellen, wann ein Element nicht mehr gültig ist [Krä07].

Ein Aggregationsoperator beispielsweise nimmt das Element auf und berechnet den Durchschnittswert, wobei sich der Operator nur den Gesamtwert aller Elemente und die entsprechende Anzahl merkt. Handelt es sich um ein Element mit Plus-Marker, wird der Wert aufaddiert und die Anzahl

inkrementiert. Bei Elementen mit Minus-Marker wird analog dazu der Wert subtrahiert und die Anzahl dekrementiert. Im Vergleich zum Intervall-Ansatz ist dieser Ansatz relativ einfach umzusetzen, da dem Operator mitgeteilt wird, wenn ein Element nicht mehr gültig ist. Allerdings erfordert dieser Ansatz auch, dass jedes Element zwei mal weitergeleitet und verarbeitet werden muss [Gra12].

2.4.2 Pattern Matching

Grundlegendes Ziel der Datenstromverarbeitung ist das Erkennen von Mustern in den gelieferten Daten und ist somit in den Bereich des Complex Event Processing (CEP) einzuordnen. Die zuvor beschriebenen Operatoren, wie beispielsweise ein Join, beziehen sich oft auf einzelne Datenelemente, die analysiert und verarbeitet werden. Beim CEP können darüber hinaus komplexe Zusammenhänge zwischen den einzelnen Elementen erkannt werden [EN10]. Ein Beispiel kann ein Sequenzpattern sein, das den Verlauf des Gesundheitszustandes eines Patienten zeigt, der sich im Laufe der Zeit ändert. Neben den Sequenzpattern gibt es weitere Pattern [EN10]:

- **Logische Pattern** können festlegen, dass *alle*, *irgendein*, oder *keines* der Events auf eine Anforderung zutreffen.
- Unter **Schwellwert-Pattern** versteht man Pattern für Minimal- Maximal oder Durchschnittswerte. Steigt beispielsweise die Temperatur einer Turbine im Durchschnitt über einen festgelegten Schwellwert, ist das entsprechende Pattern erfüllt.
- **Pattern auf Untermengen** können ebenfalls definiert werden. Dabei kann das Pattern erfüllt werden, wenn ein im Verhältnis hoher oder niedriger Wert auftritt.
- Bei **dimensionalen Pattern** handelt es sich im Pattern, die sich auf eine zeitliche oder räumliche Dimension oder eine Kombination daraus beziehen. Das Sequenzpattern ist hierbei ein Pattern mit zeitlicher Dimension und es handelt sich somit um die zuvor beschriebenen Fenster. Bei einer räumlichen Dimension können beispielsweise minimale, maximale oder durchschnittliche Abstände erkannt werden.

Eine entsprechende Beispielanfrage für ein Sequenzpattern könnte: ”Ein aus dem Krankenhaus entlassener Patient wird innerhalb von 48 Stunden erneut wegen der gleichen Erkrankung eingewiesen” sein, vgl. Abbildung 2.13.

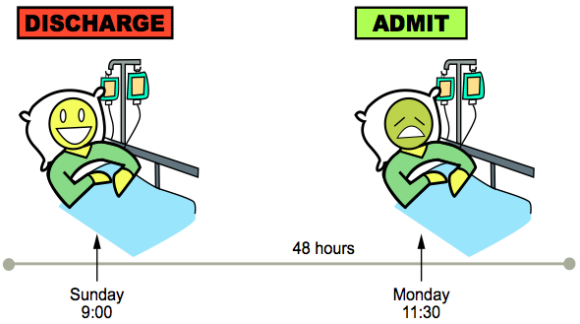


Abbildung 2.13: Veränderung über die Zeit (Sequenzpattern) [EN10]

Eine bekannte Anfragesprache für Sequenzpattern ist SASE+, bei der die Anfragen in nicht deterministische endliche Automaten (NFA) transformiert werden. Der konstruierte NFA wird durchlaufen, wenn passende Events eintreffen. Erreicht der NFA seinen Endzustand, ist das entsprechende Sequenzpattern erkannt worden. Neben einzelnen atomaren Events können auch Kleene-Ereignisse erkannt werden. Durch SASE+ Anfragen kann also die Veränderung über eine Zeitdauer ermittelt werden.

2.4.3 Verarbeitung von Datenströmen

Bisher werden bei der Verarbeitung von Datenströmen überwiegend hardwarebasierte und somit fest verdrahtete Lösungen angewandt. Diese sind durch die optimale Anpassung an den entsprechenden Datenstrom extrem effizient. Im Gegenzug sind diese jedoch sehr unflexibel, da hardwarebasierte Lösungen nur mit einem großen Aufwand an neue Gegebenheiten angepasst werden können. Eine solche Lösung bietet sich vor allem bei Datenströmen an, die aufgrund des Informationsinhaltes nur selten oder keinen Änderungen unterworfen sind [BGJ+09]. Dies können zum Beispiel medizinische Geräte sein, bei denen grundlegende Änderungen sehr selten vorkommen. Ähnlich zu den hardwarebasierten Lösungen ist auch die direkte Verarbeitung im Programmcode sehr effizient, jedoch aus dem oben genannten Grund auch relativ unflexibel und schlecht skalierbar. Kleine Änderungen am Datenstrom können auch hier große Veränderungen am Programmcode erfordern [Gra12].

Eine weitere Möglichkeit zur Lösung der angesprochenen Herausforderungen bei der Verarbeitung von Datenströmen ist der Einsatz eines DSMS.

2.4.3.1 Datenstrommanagementsysteme

Diese Systeme sind, ähnlich wie Datenbankmanagementsystem (DBMS), zur Verarbeitung und Analyse großer Datenmengen optimiert. In diesem Fall werden die Daten jedoch nicht zunächst gespeichert, sondern der ankommende Datenstrom wird *on-the-fly* analysiert und entsprechend manipuliert. Ein DSMS ist auf die Charakteristika von Datenströmen optimiert und bietet dem Anwender durch verschiedene Abfragemöglichkeiten Auswertungen auf diesen durchzuführen. Ein weiterer Vorteil ist auch, dass die meisten Systeme ein deterministisches Verhalten garantieren, d.h. bei gleichen Eingabedaten auch die gleiche Ausgabe liefern [Krä07]. Im Folgenden wird nun auf diese Art der Datenstromverarbeitung eingegangen und die Unterschiede zu DBMS herausgestellt.

Ein DSMS muss aus Gründen der Skalierbarkeit Optimierungen vornehmen können. Diese müssen zum Teil vor der Ausführung vorgenommen werden (Restrukturierung des Anfrageplans), aber auch während der Laufzeit, da die einzelnen Anfragen im Vergleich zu einem DBMS über einen längeren Zeitraum ausgeführt werden. Dabei müssen beispielsweise gemeinsame Teilbäume von Anfragen erkannt und ggf. zusammengelegt werden [Gra12].

An der Universität Oldenburg in der Abteilung Informationssysteme wird das DSMS Odysseus entwickelt, mit dem es möglich ist, entsprechende Systeme aufzusetzen. Damit das Framework an möglichst viele Anwendungsbereiche angepasst werden kann, ist es zu einem hohen Grad konfigurierbar und erweiterbar. Odysseus bietet verschiedene Anfragesprachen um Analysen auf den Datenströmen durchzuführen. Außerdem können dynamisch und mit geringem Aufwand neue Datenströme unterschiedlichster Art angebunden werden.

2.4.3.2 Unterschiede DBMS und DSMS

Der zentrale Unterschied zwischen Datenbanken und Datenströmen und der damit verbundenen Verarbeitung besteht darin, dass die Abfragen in DBMS immer Pull-basiert, also aktiv, und bei DSMS immer Push-basiert und somit passiv sind. Die Daten werden also bei DSMS nicht explizit abgefragt, sondern in der Regel aktiv von der Quelle, wie zum Beispiel einem Sensor, an das DSMS gesendet. Es gibt allerdings auch Anwendungsfälle, bei denen die Daten kontinuierlich von der Quelle abgefragt und dann weiterverarbeitet werden. In beiden Fällen ist die nachfolgende Verarbeitung jedoch Daten-getrieben. Somit werden die Anfragen bei DBMS auf eine passive, endliche Relation und bei DSMS auf eine aktive Quelle, die kontinuierlich, unter Umständen unendlich, und je nach Quelle auch unpräzise Daten sendet, gestellt [Krä07].

In Bezug auf die Anfragen handelt es sich bei DBMS immer um einmalige Anfragen, die je nach Datenmenge zeitlich variieren, aber in endlicher Zeit auf dem aktuellen Datenbestand ausgewertet werden können. Bei DSMS dagegen handelt es sich um langlaufende Anfragen, die auf die kontinuierlich eintreffenden neuen Elemente gestellt werden. Die Antworten dieser Anfragen sind bei DBMS, aufgrund der relationalen Algebra und der Endlichkeit der Daten, exakt. Im Gegensatz dazu sind die Antworten bei Datenströmen bezogen auf den gesamten Datenbestand oft nur approximativ, da zum einen viele der kontinuierlichen Anfragen, wie das Kartesische Produkt, nicht mit endlichem Speicher berechenbar sind. Zum anderen benötigen viele relationale Operatoren wie Aggregationen die gesamte Datenmenge um ausgewertet werden zu können. Für die angesprochenen Anwendungsbereiche sind approximative Ergebnisse bezogen auf alle Daten jedoch völlig ausreichend, da die Ergebnisse innerhalb der Fenster und somit der aktuellen Daten exakt sind [Krä07].

Wie auch bei bekannten DBMS gibt es auch im Bereich der DSMS Anfrageoptimierungen. Diese sind, wie auch bei DBMS, notwendig um die Last des Systems zu reduzieren und die Effizienz des Systems zu steigern. Bei DBMS werden die Anfragen vor der Ausführung optimiert, da die Anfragen einmalig und auf eine endliche Datenmenge durchgeführt werden. In DSMS ist es jedoch neben der statischen Optimierung darüber hinaus notwendig, die Anfrage während der Ausführung zu optimieren, da sich die Eigenschaften der Datenströme ändern können. Beispiele können die Datenverteilung, die Ankunftsraten neuer Elemente oder die Anzahl neuer Elemente im gleichen Zeitraum sein. Ohne die Optimierung zur Laufzeit, kann die Performance des Systems mit der Zeit stark sinken [Krä07].

In Abbildung 2.14 werden einige der beschriebenen Unterschiede der beiden Systeme noch einmal visualisiert.

2.4.4 Related Work - Publish/Subscribe

Eine andere Möglichkeit Events zu verarbeiten ist das Publish/Subscribe Konzept. Dabei gibt es Publisher, also Produzenten von Events oder Nachrichten und Subscriber, die sich auf eine Teilmenge der publizierten Nachrichten einschreiben. Die Nachrichten der Publisher werden durch eine zentrale Brokerkomponente an die entsprechenden Subscriber weitergeleitet. Wichtig ist bei diesem Konzept, dass Publisher und Subscirber in den drei Dimensionen Zeit, Ort und Synchronisation voneinander entkoppelt sind.

Die Subscriber sind allerdings in der Regel nicht an allen Nachrichten der Publisher interessiert. Aus diesem Grund besteht die Möglichkeit, dass die Subsciber Filter festlegen können, die durch die Brokerkomponente ausgewertet werden. Dabei werden themen- und inhaltsbasierte Filter unter-

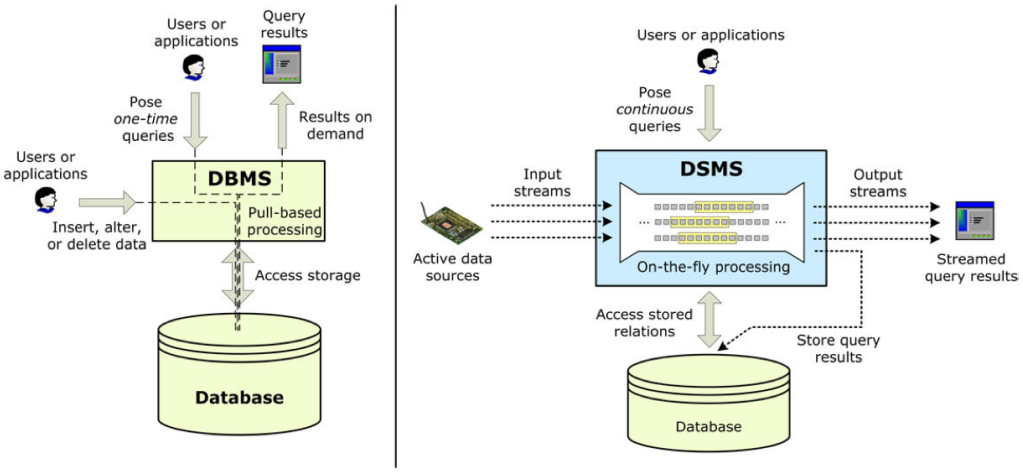


Abbildung 2.14: Unterschiede zwischen DBMS und DSMS [Krä07]

schieden. Bei der themenbasierten Filterung geben die Publisher an, welchen Themen die gesendeten Nachrichten zugeordnet werden können (Advertisement). Die Subscriber können dann das Interesse an den Themen bekunden (Subscription). Bei der inhaltsbasierten Filterung können die Subscriber spezielle Filterprädikate festlegen, die von der Brokerkomponente auf jede eingehende Nachricht ausgewertet und entsprechend an die passenden Subscriber weitergeleitet werden [EFGK03].

Die Publish/Subscribe Funktionalität ist im Datenstrommanagement-Framework Odysseus ebenfalls integriert und kann durch entsprechende Operatoren verwendet werden.

2.4.5 Zusammenfassung

Durch den immer stärker werdenden Einsatz von Sensoren beispielsweise in Smart Homes werden immer größere Datenmengen generiert, die schnell verarbeitet, aber nur selten persistent gespeichert werden müssen. Diese Datenströme werden von den Sensoren aktiv, in Echtzeit und zeitlich geordnet gesendet. Eine der größten Herausforderungen bei Datenströmen ist, dass diese potentiell unendlich sein können. Viele Operatoren benötigen die gesamte Datenmenge um ein Ergebnis liefern zu können, wie beispielsweise ein normaler Join-Operator oder eine Aggregation. Aus diesem Grund gibt es Fensteransätze, bei denen nur ein Ausschnitt des Datenstroms betrachtet wird. Entscheidend ist beim Fensteransatz, wie groß das Fenster ist (Breite) und wann sich die Endpunkte des Fensters ändern.

Ein zentraler Aspekt bei der Verarbeitung von Datenströmen ist das Erkennen von Mustern (Pattern Matching). Hierfür gibt es verschiedene Pattern, wie beispielsweise Schwellwert-Pattern oder Sequenzpattern, die die Veränderung über einen Zeitraum erkennen.

Die Verarbeitung von Datenströmen bringt neue Herausforderungen, ermöglicht es aber auch, ohne eine persistente Speicherung der Daten, diese zu manipulieren und auf Veränderungen direkt zu reagieren. Es gibt verschiedene Arten, wie die Verarbeitung durchgeführt werden kann. Dazu gehören die Verarbeitung in Hardware oder durch speziellen Programmcode. Beide Ansätze sind jedoch sehr statisch, wenig anpassbar und schlecht skalierbar. Aus diesem Grund werden hier vorwiegend DSMS betrachtet, da diese beliebig an die verschiedenen Anforderungen angepasst werden können.

Ein alternativer Ansatz ist das Publish/Subscribe Konzept, bei denen Publisher Nachrichten publizieren und Subscriber das Interesse an diesen bekunden können. Die Nachrichten werden durch das Brokernetzwerk den passenden Subscribern zugeordnet.

2.5 Odysseus

Es gibt bereits eine Vielzahl von DSMS, allerdings sind diese in ihren Eigenschaften kaum bzw. schlecht anpassbar [BGJ+09]. Da aber genau dies für einige Anwendungssysteme erforderlich ist, wurde das Framework „Odysseus“ (siehe auch Abschnitt 2.4.3.1) von der Abteilung Informationssysteme der Universität Oldenburg entwickelt. Dieses Teilkapitel beschäftigt sich im Folgenden mit der technischen Ebene des Frameworks, um dem Leser ein Verständnis über den Aufbau zu vermitteln.

2.5.1 Odysseus - Ein technischer Überblick

Um einen technischen Überblick über das Framework geben zu können, werden die wichtigsten Aspekte in unterschiedliche Bereiche gegliedert. Zunächst soll ein grober Überblick über die Architektur mit ihren Komponenten gegeben werden. Darauf aufbauend sollen Anfragesprachen betrachtet werden, die in Odysseus bereits implementiert und verwendet werden können, um dann in einem nächsten Schritt die Struktur zur Registrierung von Datenströmen verstehen zu können. Abschließend soll beschrieben werden, was die Bestandteile der Verarbeitung sind und wie diese anwendungsspezifisch erweitert werden können.

2.5.1.1 Architektur

Mit Odysseus wurde ein komponentenbasiertes Framework für maßgeschneiderte DSMS geschaffen, das dem Anwender ermöglicht, das System ohne großen Aufwand auf spezifische Anwendungsfälle anzupassen. Der Grund hierfür sind fixe und variable Funktionen innerhalb der Komponenten, die als Fix- bzw. Variationspunkte bezeichnet werden. Während Fixpunkte Odysseus-interne Verwaltungsstrukturen beschreiben und einzelne Verarbeitungsschritte ausführen, kann mithilfe von Variationspunkten die Verarbeitung von Datenströmen an anwendungsspezifische Anforderungen angepasst werden [BGJ+09].

Die Abbildung 2.15 zeigt die einzelnen Komponenten des Frameworks, die im Folgenden der Reihenfolge nach näher betrachtet werden sollen.

Übersetzungskomponente (Translate)

Mithilfe der Übersetzungskomponente bietet Odysseus dem Anwender die Möglichkeit, unterschiedliche Anfragesprachen zu nutzen, um Anfragen auf Datenströmen auszuführen. Grundlage ist hierfür eine abstrakte Parserschnittstelle, die es ermöglicht, einzelne Parser von unterschiedlichen Anfragesprachen an eine logische Algebra-Basis zu binden. Die logische Algebra-Basis ist eine entscheidende Schnittstelle, die mit den abstrakten Operatoren entsprechende Verwaltungsstrukturen bereitstellt, um aus den einzelnen Anfragen unterschiedlicher Sprachen Anfragepläne zu erstellen.

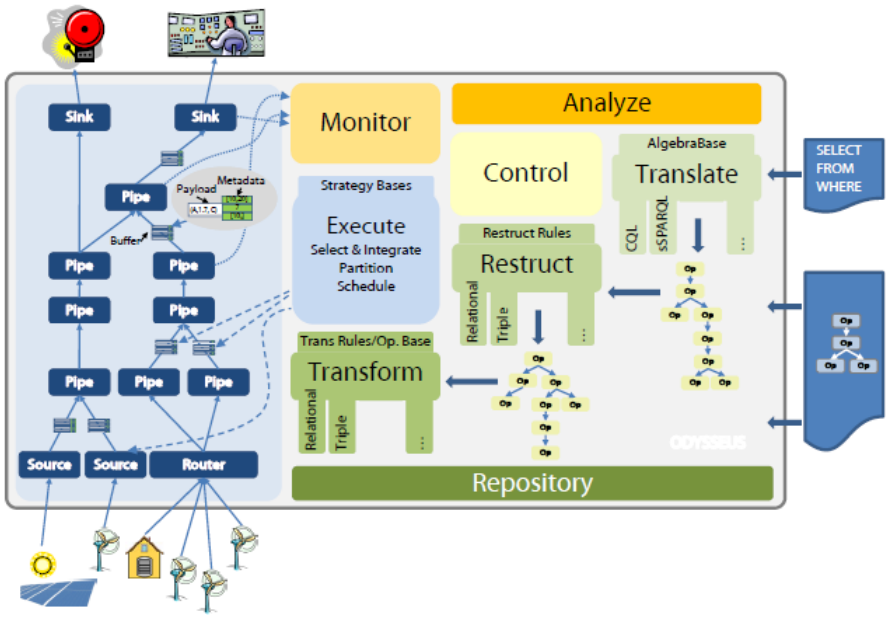


Abbildung 2.15: Architektur des Odysseus Frameworks [BGJ+09]

Weiterhin gehören auch bereits vordefinierte allgemeine Operatoren dazu, die u.a. aus dem Bereich der relationalen Algebra stammen. Da es sich bei diesen Operatoren um logische Operatoren handelt, können sie für unterschiedliche Anwendungsfälle wiederverwendet werden, da sie im Gegensatz zu physikalischen Operatoren noch keine Algorithmen besitzen.

Des Weiteren ist es möglich, auch anwendungsspezifische Operatoren zu nutzen, die entweder direkt vom abstrakten Operator oder aber von den allgemeinen Operatoren erben [BGJ+09].

Restrukturierungskomponente (Restruct)

Innerhalb der Übersetzungskomponente werden aus den anwendungsspezifischen Anfragen Anfragepläne erstellt, die nicht unbedingt am effizientesten gestaltet sind. Grund hierfür ist der abstrakte Syntaxbaum, dem jeder Anfrageplan zugrunde liegt. Um den entstandenen Anfrageplan zu restrukturieren und zu optimieren, ist in Odysseus die Restrukturierungskomponente eingefügt worden. Diese Komponente beinhaltet eine eigens für Odysseus entwickelte Regelengine, der wiederum als Eingabeparameter ein Anfrageplan und diverse Restrukturierungsregeln mitgeteilt werden. Anschließend wird der Plan auf deren Basis hin optimiert. Somit ist das Ergebnis dieser Komponente ein optimierter, logischer Anfrageplan [BGJ+09].

Transformationskomponente (Transform)

Die Transformationskomponente dient der Umwandlung eines logischen Anfrageplans in einen äquivalenten physischen Anfrageplan. Dieser Schritt ist notwendig, da die Operatoren der logischen Ebene nur beschreiben, was mit den Daten passieren sollen. Bei der Umsetzung des physischen Plans werden physische Gegenstücke zu den logischen Operatoren gesucht und eingesetzt. Da die Möglichkeit besteht, dass für einen Operator unterschiedliche Implementierungen vorhanden sind, wer-

```
1 output = SELECT({predicate='price > 100'}, input)
```

Listing 2.1: Anfragesprache PQL

den auch mehrere physische Anfragepläne erzeugt. In einem nächsten Schritt wird dann anhand einer Bewertung der jeweils bestmögliche Plan ausgewählt. Das Ergebnis der Transformationskomponente ist somit ein ausführbarer physischer Anfrageplan [BGJ+09].

Ausführungskomponente (Execute)

Das Ausführen von physischen Anfrageplänen wird innerhalb der Ausführungskomponente gesteuert. Zentraler Fixpunkt ist dabei der Scheduler, der die Ausführungsumgebung bereitstellt. Neben diesen gibt es als Variationspunkte die Scheduling-Strategie (ScS) und die Pufferplatzierungsstrategie (PPS). Diese Punkte sollen im Folgenden genauer erläutert werden.

Innerhalb des Schedulers wird bestimmt, in welcher Reihenfolge die einzelnen Anfragen auf Datenströme ausgeführt werden. Da es sich um einen Fixpunkt innerhalb des Odysseus-Frameworks handelt, hat der Anwender hier nicht die Freiheiten, die in den anderen Komponenten durch die Variationspunkte gegeben sind. Allerdings kann durch die eingangs erwähnten PPS und ScS trotzdem Einfluss auf den Ablauf genommen werden. Wird ein Puffer zwischen zwei Operatoren des Anfrageplans platziert, so kann der nachfolgende Operator erst durch den Puffer angestoßen werden. Dadurch hat der Anwender die Möglichkeit, die Ausführung der Operatoren zu steuern. Die ScS sind neben den PPS eine weitere Möglichkeit, um das Scheduling zu beeinflussen. Hierbei wird dem Scheduler mitgeteilt, in welcher Reihenfolge einzelne Teilpläne verarbeitet werden sollen. Das Thema „Verarbeitung von Anfrageplänen“ wird in Kapitel 2.5.1.4 noch genauer erläutert.

Abschließend lässt sich somit sagen, dass trotz der Tatsache, dass es sich bei dem Scheduler um einen Fixpunkt handelt, dem Nutzer durch PPS und ScS Möglichkeiten gegeben werden, den Ablauf von Anfragen auf Datenströmen zu steuern [BGJ+09].

2.5.1.2 Anfragesprachen (Query Languages)

Odysseus bietet die Option, eine der bereits implementierten Anfragesprachen zu nutzen. Eine der Sprachen wäre z.B. Continuous Query Language (CQL). CQL basiert auf SQL und wurde speziell für das Arbeiten auf Datenströmen entwickelt. Da sich die Sprache allerdings im Aufbau kaum von SQL unterscheidet, wird sie im Folgenden vernachlässigt. Der Fokus dieser Arbeit soll auf Procedural Query Language (PQL) liegen.

Procedural Query Language

Bei PQL handelt es sich um eine Operator-basierte Anfragesprache, die innerhalb des Frameworks angewendet werden kann. Dabei werden in dieser Sprache die logischen Operatoren genutzt, um sie leichter wiederzuverwenden. Die Implementierung dieser Operatoren (physische Operatoren) wird innerhalb der Transformationskomponente (siehe Kapitel 2.5.1.1) vorgenommen. Anhand eines Beispiels im Listing 2.1 soll die Semantik der Sprache erklärt werden: Es handelt sich um einen einfachen Basisoperator (SELECT), der Elemente des eintreffenden Datenstroms auswertet und entsprechend selektiert. Die einzelnen Bestandteile des Statements sind dabei:

```
input = ACCESS({source='Source', wrapper='GenericPush',
transport='TCPClient', protocol='CSV', dataHandler='Tuple',
options=[['host', 'example.com'],['port', '8080'],
['read', '10240'],['write', '10240']],
schema=[['id', 'Double'],['data', 'String']]})
```

Listing 2.2: Verwendung des ACCESS-Operators

- **output =** Dieser Bestandteil beschreibt eine zusätzliche Möglichkeit der PQL, um längere, kompliziertere Anfragen in Variablen zwischenzuspeichern und die Ergebnisse dann mit Hilfe der Variablen in anderen Anfragen zu nutzen. Dabei kann der Anwender zwischen „Intermediate Name“ (“=”, Variable ist bis zum Ende der Anfrage gültig), „View“ (“:=”, Variable wird global gespeichert) und „Source“ (“::=”, Variable wird ähnlich wie bei „View“ gespeichert, allerdings wird hier der physische Operator gespeichert und nicht wie bei der „View“ der logischeOperator) [Kuk14c].

- **SELECT** Durch den SELECT-Operator wird die jeweilige Operation definiert, die auf den in der Klammer befindlichen Operator „input“ ausgeführt werden soll [Kuk14c].

- **predicate='price > 100'** Innerhalb der Klammer eines Operators folgen in der geschweiften Klammer mögliche Parameter, die der Operator benötigt. Sollten mehrere Parameter benötigt werden, werden diese durch Kommata voneinander getrennt. In diesem Fall wird hier ein Prädikat gesetzt, dass den ankommenden Datenstrom so verarbeitet, dass nur Daten mit dem Attribut „price“ größer als 100 selektiert werden. Denkbar wären weitere Parameter, die komplexere Strukturen enthalten, wie z.B. Arrays und Key-Value-Paare [Kuk14c].

- **input** Dieser Operator beschreibt einen „Input“-Operator. Diese dienen in den Statements als Quellen, auf die die Anfrage angewendet wird. Möglich wäre hier auch, dass als Quelle eine andere Anfrage genutzt wird. Um die Anfrage nicht unnötig kompliziert zu machen, wird das zuvor beschriebene Konzept des Zwischenspeicherns genutzt [Kuk14c].

2.5.1.3 Access Framework

Im vorangegangen Kapitel wurde beschrieben, wie die Anfragesprache PQL aufgebaut ist und Anfragen an Datenströme ermöglicht. Nun soll anhand der Sprache ebenfalls erläutert werden, wie Datenströme in Odysseus integriert werden, um sie entsprechend nutzen zu können. Innerhalb der PQL gibt es die zwei Operatoren „ACCESS“ und „SENDER“, die zur Registrierung externer Datenströme zur Verfügung stehen [Kuk14a].

Das Integrieren von externen Datenströmen als Quelle wird dabei durch den ACCESS - Operator ermöglicht, dessen Struktur nachstehend durch ein Beispiel im Listing 2.2 veranschaulicht werden soll:

Hierbei wird weder View (:=) noch Source (::=) benötigt, da jeder ACCESS - Operator automatisch als Source erstellt wird [Kuk14b]. Die Parameter legen jeweils folgendes Verhalten fest:

- **source** - 'Source' ist ein eindeutiger Name für diesen Input-Operator [Kuk14b].

```
output = SENDER({sink='Sink', wrapper='GenericPush',
transport='TCPClient', protocol='CSV',dataHandler='Tuple',
options=[['host', 'example.com'],['port', '8081'],
['read', '10240'],['write', '10240']]}, input)
```

Listing 2.3: Verwendung des SENDER-Operators

- **wrapper** - „GenericPush“ wird bei aktiven Quellen eingesetzt. Zusätzlich bietet Odysseus den „GenericPull“-Operator an. Das Pull-Prinzip findet z.B. Anwendung bei Dateien. Es besteht aber auch die Möglichkeit, weitere Wrapper zu implementieren [Kuk14a].
- **transport** - Der Parameter „Transport“ beschreibt die Kommunikation zwischen Odysseus und externen Datenquellen. In diesem Beispiel wird der TCP - Transporthandler eingesetzt. [Kuk14e].
- **protocol** - Durch den „Protocol“-Parameter wird bestimmt, wie Daten von externen Datenströmen intern in Odysseus bearbeitet werden sollen bzw. wie ausgehende Daten von Odysseus übertragen werden sollen. Neben „CSV“ gibt es bspw. HTML, JSON, TEXT, XML [Kuk14d].
- **dataHandler** - Der „Data Handler“ ist für die richtige Repräsentation der Elemente des Datenstroms in interne Objekte in Abhängigkeit zum zuvor beschriebenen „Protocol Handler“ verantwortlich. In diesem Fall werden die Elemente in Form von Tupeln dargestellt [Kuk14a].
- **options** - Durch diesen Parameter können zusätzliche Eigenschaften zum „Protocol Handler“ und „Transport Handler“ hinzugefügt werden. Hierunter fallen z.B. Eigenschaften wie Port und Host [Kuk14a].
- **schema** - Mit dem „Schema“ wird beschrieben, wie die Datenreihe des ACCESS - Operators letztendlich präsentiert werden sollen. Hierbei umfasst das Schema zwei Attribute (id, data) mit deren jeweiligen Typen [Kuk14a].

Neben dem ACCESS - Operator gibt es den SENDER - Operator, der dafür zuständig ist, Ergebnisse an externe Endknoten zu senden. Bei diesen Endknoten kann es sich z.B. um mobile Endgeräte, Datenbanken, DSMS o.ä. handeln. Die Verwendung des SENDER-Operators wird dabei in Listing 2.3 beschrieben. Es wird deutlich, dass sich die beiden Operatoren im Grunde sehr ähnlich sind. Unterschiede bestehen darin, dass die Operatoren “Source” und “Schema” wegfallen. Dafür kommt ein neuer Parameter sowie ein Operator hinzu. Der “Sink” - Parameter beschreibt ähnlich dem “Source” - Parameter beim ACCESS - Operator eine Senke. In diesem Fall ist es allerdings nicht die Ausgangsquelle, von der die Datenströme kommen, sondern die Zielquelle, zu der die Ergebnisse weitergeleitet werden sollen. Wichtig ist auch hier, dass der Parameter eindeutig ist [Kuk14a].

Zweite Änderung ist der “Input” - Operator. Der Aufbau eines PQL - Statements wurde bereits in Abschnitt 2.5.1.2 behandelt, sodass dieser Operator selbsterklärend ist.

2.5.1.4 Verarbeitung von Anfrageplänen

Nachdem in den vorherigen Kapiteln Architektur und Anfragesprachen genauer beschrieben wurden, soll in diesem Kapitel anhand einer Grafik erläutert werden, wie die Verarbeitung von Anfrageplänen durchgeführt wird. Dabei wird neben einer allgemeinen Sicht auf die Verarbeitung auch beschrieben, wie anwendungsspezifische Operatoren im Framework definiert und implementiert werden können.

Allgemeine Verarbeitung

Im Abschnitt 2.5.1.1 wurde beschrieben, wie die Architektur des Odysseus Framework aussieht und wie die einzelnen Komponenten an der Umsetzung von Anfragepläne beteiligt sind.

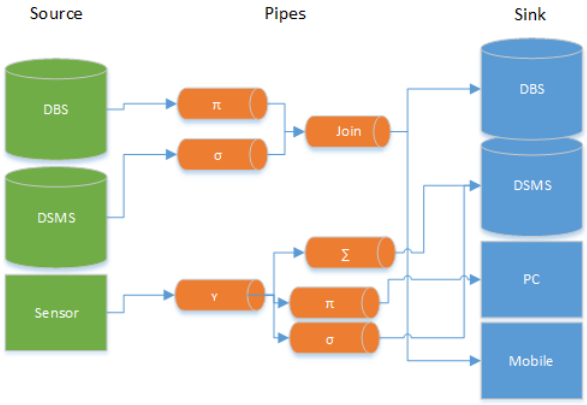


Abbildung 2.16: Datenstromverarbeitung (In Anlehnung an [KS04])

Um einen Anfrageplan auf einen Datenstrom nun anwenden zu können, muss dieser wie bereits im Abschnitt 2.5.1.3 erwähnt, z.B. mittels des PQL ACCESS - Operators beim Odysseus Framework registriert werden. Bei diesem Vorgang wird somit eine neue Datenquelle (**Source**) integriert. Diese können unterschiedlicher Herkunft sein, wie in Abbildung 2.16 zu sehen (u.a. DBS, DSMS, Sensor).

Nach der Registrierung der Datenquellen können nun auch Anfragen auf die eintretenden Datenströme angewendet werden. Dies wird nun mittels der eingangs erstellten Anfragepläne durchgeführt. Dabei werden die einzelnen Operatoren dieses Plans durch die *Pipes* in der Abbildung 2.16 dargestellt. Pipes umfassen die physischen Operatoren, die auf Grundlage der physischen Algebra-Basis auf Datenströme angewendet werden können.

Um die Ergebnisse des Anfrageplans weiterleiten zu können, müssen auch sog. Senken (**Sinks**) im Odysseus Framework registriert werden. Hierfür eignet sich bspw. der PQL SENDER-Operator (Abschnitt 2.5.1.3).

Definition eigener Operatoren

In diesem Kapitel wird anhand eines Beispiels gezeigt, wie ein eigener Operator definiert werden kann. Es soll ein “Route”-Operator mit mehreren Prädikaten erstellt werden, sodass auf einen eingehenden Datenstrom der Reihe nach alle Prädikate angewendet werden. Je nach Aussage des Prädikats (true, false, null) werden die ankommenden Daten zu unterschiedlichen Ausgängen weitergeleitet.

Logischer Operator: Der logische Operator muss das Interface “ILogicalOperator” implementierten. Hier bietet Odysseus auf Grundlage des “AbstractLogicalOperator” (implementiert das Interface) zwei Varianten an. Der “Route”-Operator soll nur einen Eingangskanal besitzen, sodass von der Klasse “UnaryLogicalOp” geerbt wird. Ein anschauliches Beispiel für die Erweiterung der Klasse “BinaryLogicalOp” wäre in diesem Fall z.B. ein “Join”-Operator (Verbinden von Elementen aus zwei Eingängen). Weiterhin spielen Annotationen eine wichtige Rolle. Bei der Definition der Klasse eines logischen Operators wird die Annotation “@LogicalOperator” genutzt. Diese benötigt die Parameter

zur Namensgebung, minimale Anzahl an Ports, die benötigt werden, sowie die Anzahl an maximalen Ports, die der Operator handhaben kann. Neben "@LogicalOperator" gibt es weitere Annotationen. Hierunter fällt z.B. "@Parameter", die wiederum verschiedene Parameter des Operators definiert. Bei der Definition dieser Annotation muss der Typ angegeben werden, der durch den Parameter unterstützt werden soll. Weitere Angaben (max.7) sind nicht zwingend erforderlich. Der Route-Operator soll nun den Namen "Route" erhalten und die minimale Anzahl als auch maximale Anzahl an Ports soll bei "1" liegen. Des Weiteren muss die Klasse das Interface "IProvidesPredicates" implementieren, da der Operator Prädikate enthalten soll. Da sich das Eingangsschema nicht vom Ausgangsschema unterscheidet (Elemente werden nur weitergegeben), muss die Methode "getOutputSchemaIntern(int port)" nicht implementiert werden, da das Ausgangsschema standardmäßig durch die Erweiterung der Klasse "AbstractLogicalOperator" dem Eingangsschema entspricht (der Port ist dabei der Ausgangsport des Operators). Nach Umsetzung dieser Zeilen ist der logische Route-Operator implementiert, wobei im Wesentlichen klar wird, dass es sich bei dieser Implementierung mehr um eine Konfiguration handelt. Die eigentliche Implementierung findet nun im physischen Operator statt [Gra14].

Physischer Operator: Jeder selbstgeschriebene physische Operator muss von der abstrakten Klasse "AbstractPipe<R,W>" erben. Dabei beschreibt die Variable "R" den Objekttyp, der vom Operator gelesen wird (read) und "W" den Objekttyp, den der Operator schreibt (write). Mit der Vererbung geht auch gleichzeitig die Aufgabe einher, die folgenden Methoden zu überschreiben:

- **getOutputMode()** - Legt fest, ob Kopien von einem Element erstellt werden sollen oder z.B. das zu lesende Element direkt bearbeitet werden soll. Im Fall des "Route"-Operators soll das zu lesende Element direkt weitergegeben werden, sodass "OutputMode.INPUT" die richtige Wahl ist [Gra14].
- **process_next(R input, int port)** - Verarbeitet das nächste Element "input" vom entsprechenden Port und schickt es per Aufruf der Methode **transfer(R output, int port)** an einen weiteren Operator. In diesem Zusammenhang besteht weiterhin die Möglichkeit, die Methoden **process_open()** und **process_close()** zu überschreiben. Diese werden aufgerufen, sobald der Operator initialisiert wird bzw. die Anfrage durchgeführt wurde. Beim "Route"-Operator wird das Element mit jedem der Prädikate evaluiert und je nach Ausgang der Evaluation an den entsprechenden Port weitergeleitet, sodass die Implementierung von process_next ausreicht [Gra14].
- **processPunctuation(PointInTime timestamp, int port)** - Mit dieser Methode wird festgelegt, wie eintreffende Punctuations verarbeiten werden sollen. Punctuations sind Nachrichten, die mit dem Datenstrom gesendet werden und nicht zwangsläufig dem Schema des Datenstroms entsprechen. Dabei kann z.B. immer ein aktueller Zeitstempel übertragen werden, was der "Route"- Operator machen soll [Gra14].
- **process_isSemanticallyEqual(IPhysicalOperator ipo)** - Diese Methode wird implementiert, um die Wiederverwendbarkeit von Operatoren zu ermöglichen. Dabei werden unterschiedliche Operatoren auf Äquivalenz überprüft, sodass nicht mehrere Operatoren dieselbe Aufgabe durchführen. Dies wird über die Regelengine des Frameworks geregelt [Gra14].

Letzter Schritt ist die Übersetzung des logischen Operators in den physischen Operator. Dies geschieht innerhalb der Transformationskomponente (2.5.1.1). Hierfür muss eine Klasse erstellt werden, die die beiden Implementierungen miteinander verbindet. Dafür wird von der Klasse "AbstractTransformationRule" geerbt. Zum Schluss wird eine Regel-Klasse für diese Verbindung erstellt und im Framework registriert.

2.5.2 Zusammenfassung

Odysseus ist ein Framework, mit dem es möglich ist, maßgeschneiderte DSMS umzusetzen. Die große Besonderheit an diesem Framework ist die Erweiterbarkeit bzw. Individualisierung. Dies kann gelingen, da in Odysseus mittels des Konzepts von Fixpunkten und Variationspunkten dem Anwender die Möglichkeit geboten wird, eigene Anfragesprachen zu implementieren und trotzdem von Anfrageplanerstellung und -optimierung zu profitieren und Anfragen direkt auf Datenströme anzuwenden. Die einzelnen Komponenten umfassen jeweils immer Fix- und Variationspunkte, sodass auf unterschiedlichen Ebenen eine Erweiterung und Anpassung möglich ist.

Odysseus selbst bietet durch die Implementierung der beiden Anfragesprachen PQL und CQL bereits standardmäßige Umsetzungen an, mit denen Anfragen auf Datenströme umgesetzt werden. Des Weiteren sind in den Implementierungen auch Operatoren eingearbeitet, die es dem Nutzer erlauben, externe Datenquellen bzw. Ziele zu definieren.

2.6 Admission Control

Im Mittelpunkt eines DBMS steht das Ausführen von *Anfragen*, mit denen der Nutzer Daten anfordern und manipulieren kann. Darüber hinaus arbeiten im Hintergrund noch weitere wichtige Komponenten, die für eine optimale Nutzung eines DBMS nötig sind. Eine Komponente ist die *Admission Control (AC)*. Diese Komponente hat die Aufgabe zu prüfen, ob eine Anfrage unter den gegebenen Ressourcen (CPU, Arbeitsspeicher etc.) ausgeführt werden kann, ohne dass es zu einer Überlastung kommt [Ber10].

Ein *DSMS* zeichnet sich dadurch aus, dass es Datenströme mittels kontinuierlicher Anfragen verarbeiten kann (wie bereits in den vorherigen beiden Teilkapiteln beschrieben). Ein *Datenstrom* ist eine potentiell unendliche Sequenz von Datenelementen, für deren Erhebung sogenannte aktive Datenquellen verantwortlich sind (siehe 2.4.0.1). Im Gegensatz zu DBMS können DSMS jedes Datum nur einmal verarbeiten, da die Speicherung aller Daten aufgrund der theoretisch unendlich großen Menge unrealistisch ist [Krä07]. Dies hat zur Folge, dass an die AC eines DSMS andere Anforderungen gestellt werden müssen, als an die AC eines DBMS.

In Abschnitt 2.6.1 wird auf die AC eines DBMS eingegangen. Dabei werden insbesondere die *Kostenmodelle* gezeigt, die Voraussetzung für die AC sind. In Abschnitt 2.6.2 werden Ansätze für die AC eines DSMS vorgestellt. Hierbei werden auch die Unterschiede zu der AC eines DBMS dargelegt. Zum Abschluss folgt eine kurze Zusammenfassung.

2.6.1 Admission Control in Datenbankmanagementsystemen

Die AC eines DBMS nutzt i. d. R. Kostenmodelle, die auch in der Verarbeitung einer Anfrage zum Bestimmen des optimalen Anfrageplans verwendet werden. Daher wird zunächst die Verarbeitung einer Anfrage genauer erläutert, bevor auf die Kostenmodelle eingegangen wird. Abschließend werden die Reaktionsmöglichkeiten der AC eines DBMS genauer betrachtet.

2.6.1.1 Verarbeitung einer Anfrage

Die Abbildung 2.17 zeigt die einzelnen Phasen, die bei der Planung und Ausführung einer SQL-Anfrage in einem gängigen relationalen Datenbanksystem (z.B. MySQL oder Oracle) durchlaufen werden.

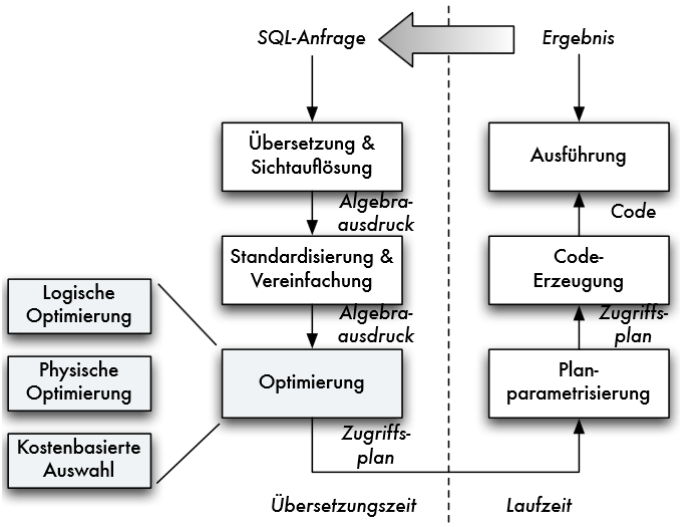


Abbildung 2.17: Phasen der Anfrageverarbeitung [Kud07]

Übersetzung und Sichtauflösung: In der ersten Phase wird die SQL-Anfrage hinsichtlich ihrer Syntax auf Korrektheit überprüft und in einen Ausdruck der relationalen Algebra überführt. Das Ergebnis der Phase ist ein Anfrageplan mit logischen Operatoren. *Logische Operatoren* zeichnen sich dadurch aus, dass sie den Operator beschreiben (z.B. Mindestanzahl und Höchstanzahl der eingehenden Relationen) [Kud07, Mit95].

Standardisierung und Vereinfachung: Diese Phase überführt den logischen Anfrageplan in ein standardisiertes Format, damit die Anfrage später effizienter ausgeführt werden kann. Dazu werden z.B. Verschachtelungen durch effizienter ausführbare Verbundoperationen ersetzt [Kud07].

Optimierung: In dieser Phase findet zunächst die logische Optimierung statt. Dabei wird noch nicht auf das interne Schema oder die Größe der Relationen zugegriffen. Mittels Transformationsregeln können z.B. mehrere aufeinander folgende Selektionen oder Projektionen zusammengefasst werden, was letztlich zu einer Effizienzsteigerung bei der Ausführung der Anfrage führen soll. Anschließend erfolgt die physische Optimierung, bei der die logischen Operatoren durch konkrete Algorithmen ersetzt werden. Dabei können mehrere zu einander äquivalente physische Anfragepläne entstehen. Um den besten Plan auszuwählen, müssen die verschiedenen Pläne mit Kosten bewertet werden. Der Plan mit den geringsten Kosten ist i. d. R. der beste Plan. Das Ergebnis dieser Phase ist ein Anfrageplan mit physischen Operatoren. Im Gegensatz zu logischen Operatoren, beinhalten die *physischen Operatoren* die konkreten Implementierungsdetails eines Operators [Kud07, Mit95].

Planparametrisierung, Code-Erzeugung und Ausführung: In einigen optimierten Plänen müssen Platzhalter noch durch konkrete Werte ersetzt werden. Hierfür ist die Planparametrisierung zuständig. Anschließend wird der Anfrageplan in einen ausführbaren Code transformiert und kann ausgeführt werden [Kud07].

2.6.1.2 Kostenmodelle

Bei der physischen Optimierung können zu einem logischen Anfrageplan mehrere zueinander äquivalente physische Anfragepläne entstehen. Um den Plan auszuwählen, mit dem die Anfrage am effizientesten ausgeführt werden kann, müssen die einzelnen Pläne verglichen werden. Dazu werden die Kosten zu jedem Plan ermittelt und anschließend die Pläne anhand der Kosten verglichen. Die Kosten beschreiben dabei die Höhe der Nutzung von Ressourcen (z.B. Anzahl der Zugriffe auf den Arbeitsspeicher) [Kud07].

Um die Kosten exakt zu bestimmen, wäre es am einfachsten, die physischen Pläne auszuführen. Dieser Ansatz scheitert jedoch daran, dass die Anzahl an alternativen Plänen sehr hoch sein kann und somit der Aufwand für die Kostenermittlung zu hoch ist. Es gibt z.B. allein für die verschiedenen Verbundreihenfolgen von 10 Relationen über 17 Mrd. Varianten [Kud07]. Daher werden in den gängigen DBMS die Kosten anhand von Kostenmodellen geschätzt. Kostenmodelle nutzen dabei i. d. R. Statistiken als Grundlage, die ein DBMS als Metadaten zur Verfügung stellt [Mit95].

Statistiken

Statistiken können nach [Mit95] folgendermaßen klassifiziert werden:

Statistische Kenngrößen pro Segment: Zum Beispiel die Anzahl der Datenseiten pro Segment oder die Anzahl der leeren Seiten pro Segment.

Statistische Kenngrößen pro Relation: Zum Beispiel die Anzahl der Tupel in einer Relation oder die Anzahl der Seiten mit Tupeln aus einer Relation.

Statistische Kenngrößen pro Zugriffspfadstruktur: Zum Beispiel die Anzahl der Blatt-Seiten im Zugriffspfad oder die Höhe des Zugriffspfades.

Statistische Kenngrößen pro Attribut: Zum Beispiel das Histogramm der Wertverteilung oder die Anzahl der Attributwerte.

Eine besondere Rolle spielt bei der Kostenbestimmung die Häufigkeitsverteilung der Werte eines Attributs. Diese sind notwendig für die Selektivitätsabschätzung eines Operators. Relationale Operatoren transformieren eine Eingabe in eine Ausgabe. Dabei bestimmt die Selektivität eines Operators, wie viele Tupel das Prädikat eines Operators erfüllen und von der Eingabe in die Ausgabe gelangen. Bei einer Selektivität von 0.5 würde z.B. die Hälfte aller Tupel das Prädikat des Operators (z.B. Selektion oder Join) erfüllen. Die Selektivität kann ausschließlich anhand von Häufigkeitsverteilungen der Werte eines Attributes geschätzt werden, sodass die Kostenmodelle nie exakte Ergebnisse liefern können. Ziel ist allerdings, die Selektivität so genau wie möglich zu schätzen und somit das Kostenmodell zu optimieren. Es gibt verschiedene Methoden, um die Häufigkeitsverteilungen zu schätzen. Dabei wird grundsätzlich zwischen parametrischen und nicht-parametrischen Methoden unterschieden [Bec10].

Parametrische Methoden

Parametrische Methoden können die Häufigkeitsverteilung der Werte eines Attributes anhand einer Modellfunktion beschreiben. Als Modellfunktionen können z.B. die Normalverteilung, die Gleichverteilung oder die Zipf-Verteilung genutzt werden. Die Parameter lassen sich dabei z.B. durch Sampling bestimmen. Das heißt, anhand von Stichproben kann auf die Gesamtheit geschlossen werden. Dies ist bei den parametrischen Methoden ausreichend, da die Modellfunktionen sowieso nur eine Annäherung an die Verteilung der Werte liefern können [Bec10].

Nicht-Parametrische Methoden

Die nicht-parametrischen Methoden unterscheiden sich von den parametrischen Methoden in der Form, dass keine Annahme über die Verteilung der Werte eines Attributes getroffen wird. Die am häufigsten verwendete nicht-parametrische Methode ist das Histogramm. Ein Histogramm ist eine Datenstruktur, mit der Häufigkeitsverteilungen beschrieben werden können. Die Datenstruktur teilt den Definitionsbereich eines Merkmals bzw. Attributs in Klassen bzw. Intervalle und ordnet diesen anschließend die Häufigkeiten zu. Ein Beispiel für ein Histogramm ist in Abbildung 2.18 dargestellt. Dieses Beispiel zeigt ein breitengleiches Histogramm, da bei dieser Variante die Intervalle die selbe Länge haben. Darüber hinaus gibt es viele weitere Varianten. Beispielsweise zeichnet sich das tiefengleiche Histogramm dadurch aus, dass die Intervallgrenzen in der Form definiert werden, dass alle Intervalle annähernd über gleich viele Häufigkeiten verfügen [Bec10].

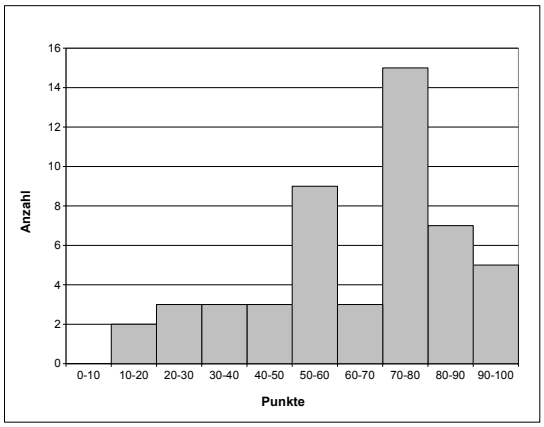


Abbildung 2.18: Punktespiegel einer Klausur als Histogramm

Logische- und physische Kostenmodellierung

Mit den Statistiken als Grundlage können die Kosten eines Anfrageplanes abgeschätzt werden. Dabei werden in iterativer Vorgehensweise die Kosten zu jedem Operator bestimmt und anschließend die Kosten der Operatoren summiert. Dabei muss zwischen der logischen- und physikalischen Kostenmodellierung unterschieden werden [Bec10].

Logische Kostenmodellierung

In der logischen Kostenmodellierung werden die Kosten eines Operators anhand der Anzahl an Tupel in der Ausgabe eines Operators ausgewiesen. Die Anzahl an Tupel in der Ausgabe eines Operators wird je nach Operatorart unterschiedlich berechnet. Im Folgenden wird die Kostenberechnung anhand des Selektions- und Verbundoperators gezeigt.

Der Selektion-Operator beinhaltet ein Prädikat, welches die Tupel der Eingabe erfüllen müssen, um in die Ausgabe zu gelangen. Durch die folgende Formel kann die Ausgabe geschätzt werden [Bec10]:

$N_{Ausgabe} = S * N_{Eingabe}$

Hierbei ist $N_{Ausgabe}$ die Kardinalität der Ausgaberelation, $N_{Eingabe}$ die Kardinalität der Eingaberelation und `S` die zuvor geschätzte Selektivität des Operators.

Der Verbund-Operator verbindet mehrere Relationen miteinander. Je nach Verbundart muss die Ausgabe unterschiedlich berechnet werden. Durch die folgende Formel kann z.B. die Ausgabe eines Gleichverbundes geschätzt werden [Bec10]:

$N_{R_1 \bowtie R_2} = S_j * N_{R_1} * N_{R_2}$

Dabei steht $N_{R_1 \bowtie\ R_2}$ für die Kardinalität der Ausgaberelation, N_{R_1} und N_{R_2} für die Kardinalitäten der Eingaberelationen und S_J für die Verbundsselektivität.

Physikalische Kostenmodellierung

Bislang wurden die Kosten nur anhand der Anzahl an Tupel in der Ausgabe eines Operators gemessen. Für die AC sind allerdings die physikalischen Kosten von größerer Bedeutung, da diese unmittelbar Auskunft über die Nutzung der Hardware geben. Diese können nach [Mit95] folgendermaßen klassifiziert werden:

E/A-Kosten: Tupel müssen zur Verarbeitung aus dem Externspeicher in den Datenbankpuffer, einen Teil des Arbeitsspeichers, geladen werden. Die Anzahl der Zugriffe auf den Externspeicher ist ein wichtiger Kostenfaktor.

CPU-Kosten: Für die Verarbeitung von Tupeln wird die CPU in Anspruch genommen. Dabei können die Kosten anhand der Zugriffe auf das Zugriffssystem gemessen werden. Das Zugriffssystem ist für die Verwaltung von physischen Sätzen und Zugriffspfaden verantwortlich.

Kommunikationskosten: Innerhalb einer verteilten Datenbank ist z.B. die Netzwerkkommunikation zwischen den Teilen einer Datenbank ein weiterer Kostenfaktor.

Die physikalischen Kosten eines Operators, welcher eine Basisrelation ausliest, lässt sich z.B. folgendermaßen berechnen [Mit95]:

$cost(R) = extspcard(R) + W * osscard(R)$

Hierbei steht `R` für eine Basisrelation, die Funktion `extspcard` für die Anzahl der Externspeicherzugriffe und die Funktion `osscard` für die Anzahl der Zugriffssystemaufrufe. Der Parameter `W` bestimmt das Verhältnis zwischen E/A-Kosten und CPU-Kosten. Je nach System muss `W` individuell bestimmt werden. Beispielsweise sollte `W` bei einem System mit einem schnellen Prozessor klein gewählt werden, sodass die CPU-Kosten geringer gewertet werden als die E/A-Kosten [Mit95].

2.6.1.3 Admission Control

DieAC nutzt die Methoden der Kostenmodellierung, um die Last einer Anfrage zu schätzen. Um eine Überlastung des Systems zu verhindern, muss gelten [Mic11]:

$L + L_a <= L_{max} * H, H \in (0, 1]$

L_a steht für die zusätzliche Last bzw. Kosten, die durch eine neue Anfrage potentiell entstehen kann. Unter `L` wird die momentane Systemlast verstanden und L_{max} ist die maximale Systemlast. Die Systemlast kann hierbei als die Summe aller Kosten beschrieben werden, die durch Anfragen entstehen. `H` bezeichnet den Headroom, mit dem die maximale Systemlast angepasst werden kann, falls Leistungsreserven vorgesehen sind. Die Lastüberprüfung findet unmittelbar vor dem Ausführen der Anfrage statt. Dabei kann die AC unterschiedlich reagieren [Mic11, HSH07]:

Direkt ausführen: Die hinzukommende Last ist nicht zu groß und die Anfrage kann direkt ausgeführt werden.

Eingeschränkt ausführen: Die hinzukommende Last ist zu groß. Daher wird die Ausführung der Anfrage eingeschränkt. Beispielsweise könnte die Anzahl an Tupel, die als Ergebnis der Anfrage entstehen, beschränkt werden.

Zurückstellen: Die hinzukommende Last ist zunächst zu groß. Es ist jedoch davon auszugehen, dass die Anfrage in absehbarer Zeit ausgeführt werden kann.

Vorschläge: Die hinzukommende Last ist zunächst zu groß. Es könnten jedoch Vorschläge gemacht werden, welche Kombinationen aus Anfragen ohne Überlastung ausgeführt werden können. Dabei können z.B. Anfragen mit einer höheren Priorität bevorzugt werden.

Ablehnen: Die hinzukommende Last ist zu groß und würde das System erheblich beeinträchtigen. Daher wird die Anfrage abgelehnt.

2.6.2 Admission Control in Datenstrommanagementsystemen

Die AC eines DBMS lässt sich nicht auf ein DSMS übertragen. Daher wird zunächst auf die Unterschiede bezüglich der AC beider Systeme eingegangen. Anschließend werden drei Kostenmodelle vorgestellt, die von der AC eines DSMS verwendet werden können.

2.6.2.1 Unterschiede zwischen Admission Control eines DBMS und eines DSMS

Die AC eines DBMS kann für eine Anfrage abschätzen, wie groß die Ergebnismenge sein wird, da sie auf persistent gespeicherte Daten zugreifen kann. Ein DSMS hingegen muss kontinuierlich neue Datenstromelemente verarbeiten, sodass die Ergebnismenge vor dem Starten der Anfrage nicht abgeschätzt werden kann [Mic11].

Während ein DBMS eine endliche Ergebnismenge liefert, ist die Ergebnismenge bei einer Anfrage eines DSMS theoretisch unendlich lang. Dies hat zur Folge, dass es in einem DBMS ausreicht, die Kosten vor dem Ausführen einer Anfrage abzuschätzen. In einem DSMS hingegen müssen die Kosten kontinuierlich abgeschätzt werden, da sich die Systemlast während der Ausführung einer Anfrage ändern kann. Dies ist zum Einen darauf zurückzuführen, dass die Systemlast durch andere fremde

Systeme beeinflusst werden kann, zum Anderen kann die Datenrate des Datenstroms auch variieren und die Werteverteilung der Attribute kann sich ändern (Datenevolution). Unter der Datenrate wird die Anzahl an Datenstromelementen pro Zeiteinheit verstanden. Während ein System z.B. ohne Probleme ein Tupel pro Sekunde verarbeiten kann, wird das System bei 100 Tupel pro Sekunde überlastet. Um diese Änderungen der Datenrate zu berücksichtigen, muss eine kontinuierliche Kostenabschätzung erfolgen [Mic11].

Ein weiterer Unterschied ist, dass in einem DBMS die Anfragen i. d. R. unabhängig voneinander ausgeführt werden. In einem DSMS hingegen findet häufig Query Sharing statt. Hierbei teilen sich verschiedene Anfragen physische Operatoren. Dieses muss in der Kostenabschätzung berücksichtigt werden, sodass die Kosten für einen Operator nicht mehrfach verrechnet werden [Mic11].

2.6.2.2 Kostenmodelle

Es gibt verschiedene Kostenmodelle für DSMS, die von der AC benutzt werden können. Im Folgenden werden in Anlehnung an [Mic11] drei Kostenmodelle vorgestellt, die in Odysseus [AGG+12], einem generischen Framework für DSMS, verwendet werden.

Kostenmodell auf Basis der Anfragenanzahl

Ein mögliches Kostenmodell ist, dass nur so viele Anfragen zugelassen werden, bis eine vorher festgelegte maximale Anzahl an Anfragen erreicht wird. Dieses Kostenmodell würde nur wenig Speicherplatz und Rechenzeit verbrauchen. Allerdings ist das Kostenmodell sehr ungenau, da die Anfragen i. d. R. eine unterschiedliche Menge an Ressourcen benötigen und dies bei dem Kostenmodell nicht berücksichtigt wird. So kann es z.B. vorkommen, dass die maximale Anzahl an Anfragen erreicht wurde, obwohl die Systemlast noch sehr gering ist. Auch berücksichtigt diese Methode die Datenrate und Query Sharing nicht.

Kostenmodell auf Basis der Operatoranzahl

Ein anderes Kostenmodell ist, dass im System eine maximale Anzahl an physischen Operatoren festgelegt wird, die parallel ausgeführt werden dürfen. Somit können die Kosten einer Anfrage durch die Anzahl an Operatoren der Anfrage festgelegt werden. Dieses Kostenmodell würde ebenfalls nur wenig Speicherplatz und Rechenzeit benötigen. Zudem werden die Kosten genauer und unabhängig von der Anzahl der Anfragen berechnet. Es kann zudem Query Sharing berücksichtigt werden. Der wesentliche Nachteil dieser Methode ist, dass alle Operatoren gleich behandelt werden, obwohl diese sich beim Speicherplatz und in ihrer Rechenzeit erheblich unterscheiden können. Außerdem wird die Datenrate bei dieser Methode nicht berücksichtigt.

Kostenmodell auf Basis der Operatoreigenschaften

Die beiden vorgestellten Kostenmodelle sind zwar leicht umzusetzen, allerdings werden die Kosten nur sehr grob geschätzt und stellen keine zufriedenstellende Lösung dar. Es lässt sich erkennen, dass ein besseres Kostenmodell die unterschiedlichen Operatoren genauer untersuchen und auch die Datenrate berücksichtigen muss. An dieser Stelle kann aufgrund der Komplexität nur ein kleiner Einblick in ein solches Modell gegeben werden.

Kostenfaktoren

Ein DSMS verarbeitet die Datenströme i. d. R. sofort, ohne dass diese im Externspeicher gespeichert werden. Die Datenströme befinden sich daher nur im Arbeitsspeicher, sodass ein wesentlicher Kostenfaktor der Arbeitsspeicherverbrauch ist. Ein DSMS hat keinen Einfluss darauf, wie viele Daten eine Datenquelle pro Zeiteinheit (Datenrate) liefert. Werden mehr Daten pro Zeiteinheit geliefert, als verarbeitet werden können, wird die CPU überlastet. Somit muss das Kostenmodell die Verarbeitungsgeschwindigkeit der Datenstromelemente als weiteren Kostenfaktor berücksichtigen. Weitere Kostenfaktoren können z.B. die Anzahl an Zugriffen auf den Arbeitsspeicher oder die Kommunikation in einem verteilten DSMS sein.

Konstruktion von Histogrammen

Häufigkeitsverteilungen in Form von Histogrammen spielen auch in DSMS eine wichtige Rolle, da diese für die Selektivitätsabschätzung eines Operators benötigt werden. Während in DBMS die Histogramme auf Grundlage der gespeicherten Daten konstruiert werden können, ist dies in DSMS nicht möglich. Es gibt jedoch andere Methoden, um auch in DSMS Histogramme zu erstellen. Zu den Methoden gehören z.B. die Partition Incremental Discretization oder die Sampling-Methode. Die Histogramme müssen zudem ständig aktualisiert werden, sodass die Datenevolution auch berücksichtigt wird.

Kosten eines Operators

Der Arbeitsspeicherverbrauch eines Operators wird maßgeblich dadurch beeinflusst, ob dieser Statusbehaftet ist oder nicht. Statusbehaftete Operatoren zeichnen sich dadurch aus, dass diese Datenstromelemente zwischenspeichern müssen, um zu einem Ergebnis zu kommen. Ein Beispiel ist der Aggregationsoperator, welcher den Durchschnitt aus einer Menge von Datenstromelement bilden muss. Nicht-statusbehaftete Operatoren müssen hingegen keine Datenstromelemente zwischenspeichern und können bei Berechnung des Arbeitsspeicherverbrauchs vernachlässigt werden. Die Zeit zur Verarbeitung eines Datenstromelements innerhalb eines Operators kann in Prozessorzyklen gemessen werden. Die Anzahl der Prozessorzyklen ist dabei stark abhängig von der Art eines Operators und muss individuell festgelegt werden.

Beobachtung des Ausführungsplanes

Eine Anfrage in DSMS wird kontinuierlich ausgeführt. Dies hat den Vorteil, dass einige Abschätzungen im Laufe der Zeit durch exakte Messungen ersetzt werden können. So ist es z.B. möglich, dass ein Operator während der Ausführung festhält, wie hoch die Selektivität ist. Auch kann jeder Operator selber ermitteln, wie lange er für die Verarbeitung eines Datenstromelements benötigt. Mit diesen Messungen kann letztlich die Kostenabschätzung während der Ausführung einer Anfrage optimiert werden.

2.6.3 Zusammenfassung

Die AC kann entscheidend dazu beitragen, dass ein System nicht überlastet wird. Allerdings kann die AC auch dazu führen, dass nicht das gesamte Ressourcenpotential ausgenutzt wird. Die AC arbeitet

nämlich hauptsächlich mit Schätzungen, die nie exakt sein können. Bei falschen Schätzungen ist es somit möglich, dass Anfragen abgelehnt werden, obwohl diese noch hätten ausgeführt werden können. Auch ist es möglich, dass die AC Anfragen akzeptiert, obwohl das System schon ausgelastet ist. Zudem muss festgehalten werden, dass die AC eines DSMS nicht so genau arbeiten kann, wie die eines DBMS. Dies ist darauf zurückzuführen, dass die Selektivität für einen Operator innerhalb eines DBMS sehr genau geschätzt werden kann, da mit bereits bekannten Daten gearbeitet wird. Die Genauigkeit der Selektivitätsabschätzung für einen Operator innerhalb eines DSMS kann sehr stark schwanken, da die Datenströme letztlich unvorhersehbar sein können und von der einen auf die andere Sekunde völlig unterschiedliche Datenstromelemente liefern können.

2.7 Replikation und Fragmentierung

Das folgende Teilkapitel beschreibt die Replikation und Fragmentierung. Das Ziel, sowohl der Replikation als auch der Fragmentierung ist es, die Ressourcen in einem verteilten DBMS optimal zu nutzen und somit einen Performanzgewinn zu erzielen [Kud92] [ÖV99]. Die Replikation verfolgt dieses Ziel indem sie Kopien der Daten auf den einzelnen Instanzen hinterlegt, so dass nicht bei jeder Transaktion auf die, im Zweifel auf einer anderen Instanz liegenden, Originaldaten zurückgegriffen werden muss. Des Weiteren erhöht das Replizieren von Daten in einem verteilten DBMS die Ausfallsicherheit, da die Daten nicht verloren sind wenn es noch mindestens eine Kopie der Daten gibt [ÖV99]. Bei der Fragmentierung wird das Ziel des Performanzgewinns dadurch verfolgt, dass die Daten in verschiedene Partitionen auf unterschiedliche Instanzen unterteilt werden. Diese Partitionierung hat zur Folge, dass jede Instanz weniger Daten für eine Anfrage auswerten muss [Kud92] [ÖV99]. Diese Arbeit beschäftigt sich mit der Fragestellung, ob und inwieweit sich diese beiden Konzepte der Replikation und Fragmentierung von DBMS auf DSMS übertragen lassen. DSMS zeichnen sich in diesem Zusammenhang dadurch aus, dass die Daten flüchtig und die Anfragen persistent sind [Krä07]. Zunächst werden dazu die beiden Konzepte vorgestellt und die Möglichkeiten, diese in ein DSMS zu übernehmen diskutiert. Anschließend wird die Umsetzung der Replikation und einiger Fragmentierungsverfahren in dem DSMS Odysseus konzeptionell beschrieben.

2.7.1 Replikation

Wie bereits erwähnt verfolgt die Replikation das Ziel, die Ausfallsicherheit zu erhöhen und Antwortzeiten zu verringern [ÖV99]. In diesem Kapitel wird der Mechanismus der Replikation zunächst in DBMS und anschließend in DSMS erläutert.

2.7.1.1 Replikation in einem DBMS

Bei der Replikation von Daten in einem verteilten DBMS werden die Tupel einer Relation kopiert und die Kopien auf mehrere Instanzen verteilt. Durch eine solche Duplizierung kann zum einen die Ausfallsicherheit erhöht werden, da die Daten bei Verlust einer Instanz immer noch auf anderen Instanzen verfügbar sind. Zum anderen kann der Zugriff auf die Daten durch eine Replikation verbessert werden. So wird zum Beispiel die Antwortzeit reduziert, wenn eine lokale Kopie der Daten, also ein Replikat, vorliegt. [ÖV99] [Iak12]. Eine solche Replikation erfordert allerdings immer auch eine Synchronisation, die alle Replikate zu festgelegten Zeitpunkten auf den gleichen Stand bringt. Man

unterscheidet anhand der Synchronisation zwei grundlegende Arten der Replikation: die synchrone und die asynchrone Replikation [Iak12].

Bei der synchronen Replikation gilt eine Änderungsoperation auf den Daten nur als abgeschlossen, wenn die Änderung für alle Replikate übernommen wurde [Iak12]. Um diese Synchronisation umzusetzen wird ein sogenanntes „Commit-Protokoll“ eingesetzt, dass in [SS07] näher erläutert wird. Im Gegensatz zur synchronen gilt bei der asynchronen Replikation eine Änderungsoperation unmittelbar als abgeschlossen. Dies hat zur Konsequenz, dass die einzelnen Kopien nur zum Zeitpunkt der Replikation identisch sind [Iak12].

2.7.1.2 Replikation in einem DSMS

In einem DSMS ist es nicht sinnvoll, Anfragen nur auf dem aktuell im System befindlichen Datenbestand auszuführen, da dieser flüchtig ist [Krä07]. Aus diesem Grund bezieht sich eine Replikation im Kontext von DSMS auf die Duplizierung der persistenten Anfragen. So kann ein Datenstrom parallelisiert und von mehreren Instanzen gleichzeitig mit der gleichen Anfrage verarbeitet werden [Bra13a]. Durch diesen Umstand liegt das Hauptaugenmerk bei der Replikation in DSMS auf der Ausfallsicherheit und nicht auf der Verkürzung der Antwortzeiten. Die Unterscheidung von synchroner und asynchroner Replikation ist in DSMS insofern nicht sinnvoll, da zum einen die Daten selber nicht dupliziert und zum anderen die Anfragen zur Laufzeit nicht durch den Benutzer verändert werden. Zwar kann sich durch interne Mechanismen, wie z.B. eine Planmigration, eine Anfrage zur Laufzeit ändern, allerdings ohne dabei von der ursprünglichen Anfrage abweichende Ergebnisse zu liefern [Krä07]. Dennoch spielt die Synchronisation bei der Replikation in DSMS insofern eine Rolle, als dass es sinnvoll erscheint, die einzelnen Datenströme, die die replizierten Anfragen als Ergebnisströme liefern anschließend zu synchronisieren. Diese Synchronisation hat zum einen den Hintergrund, dass die Ergebnisse somit an einem Punkt abrufbar sind, unabhängig von der Replikation. Zum anderen kann durch eine sinnvoll gestaltete Synchronisation gegebenenfalls ein Performanzgewinn erzielt werden, wenn die einzelnen Instanzen unterschiedlich lange für die Anfrageverarbeitung benötigen und die Synchronisation nicht auf das Vorliegen der Daten von allen Replikaten wartet. Dies wäre vor dem Hintergrund der Ausfallsicherheit auch nicht zielführend. An dieser Stelle sei noch anzumerken, dass bei einer Synchronisation von Datenströmen auf die zeitliche Ordnung dieser zu achten ist [Bra13a].

2.7.2 Fragmentierung

Das Ziel einer Fragmentierung ist es einen Performanzgewinn dadurch zu erzielen, dass die Daten auf verschiedene Partitionen aufgeteilt werden [Kud92]. Bevor auf den Mechanismus der Fragmentierung in DBMS und DSMS eingegangen wird, wird zunächst die zugrundeliegende Parallelisierung erläutert.

2.7.2.1 Parallelisierung

Bei einer Parallelisierung von (Teil-) Anfragen (im Folgenden nur noch Anfragen genannt) in einem DBMS gibt es verschiedene Bedingungen oder Ziele. Eines davon ist semantische Äquivalenz. So muss bei der gleichen Anfrage das Ergebnis einer verteilten, parallelisierten Verarbeitung mit dem einer sequentiellen übereinstimmen [GJPPM+12]. Die Parallelisierung soll zu einem Performanzge-

winn und einem besseren Ausnutzen der Ressourcen führen und nicht zu einem anderen Ergebnis. Der angesprochene Performanzgewinn ist ebenfalls ein Ziel. Betrachtet man mehrere voneinander unabhängige Anfragen, so kann dieses Ziel durch eine parallele Verarbeitung dieser Anfragen erreicht werden. Ist dem hingegen eine einzige Anfrage Mittelpunkt der Betrachtung, so kann dieses Ziel nicht nur dadurch erreicht werden, die Anfrage auf mehreren Instanzen gleichzeitig zu verarbeiten. Die Verarbeitungslast muss pro Instanz geringer sein als bei einer sequentiellen Verarbeitung. Das kann dadurch erreicht werden, dass eine Instanz nicht alle Tupel einer Relation zur Verarbeitung heranzieht. Die Tupel werden auf die parallel arbeitenden Instanzen verteilt, so dass zwar jedes Tupel verarbeitet wird (semantische Äquivalenz), nach Möglichkeit allerdings auch nur von einer Instanz [GJPPM+12]. Dies reduziert die Verarbeitungslast pro Instanz und kann demnach zu einem Performanzgewinn führen. Wie aus dem Unterschied zwischen der Betrachtung mehrerer oder lediglich einer Anfrage hervorgeht, gibt es verschiedene Arten von Parallelisierung in einem DBMS [TLRG08]: die Parallelität zwischen Anfragen, die Parallelität innerhalb einer Anfrage, die Parallelität zwischen Operatoren, die Parallelität zwischen unabhängigen Operatoren und die gemischte Parallelität. Eine Erläuterung dieser Parallelisierungsarten ist [Bra13a] zu entnehmen.

2.7.2.2 Fragmentierung in einem DBMS

Bei einer Fragmentierung von Daten in einem DBMS werden die Tupel einer Relation auf mehrere Partitionen verteilt. Handelt es sich dabei ebenfalls um eine Aufteilung auf verschiedene Instanzen, so ermöglicht diese Aufteilung der Tupel die Speicherung großer Datenmengen. Die Menge ist abhängig von den Kapazitäten der einzelnen DBMS-Instanzen und insbesondere von der Anzahl der Instanzen [ÖV99]. Die Fragmentierung einer Relation kann grundlegend auf drei Arten und Weisen geschehen: durch eine horizontale, vertikale oder gemischte Fragmentierung [Kud92]. Bevor die einzelnen Fragmentierungsarten vorgestellt werden, sollen kurz die Regeln (entnommen aus [Kud92]) angebracht werden, die eine Fragmentierung einhalten muss:

- Vollständigkeit
  Alle Daten der globalen Relation müssen vollständig auf die Fragmente abgebildet werden.
- Rekonstruierbarkeit
  Es muss immer möglich sein, jede globale Relation aus ihren Fragmenten zu konstruieren.
- Ausschließlichkeit
  Alle Fragmente sind disjunkt, besitzen also keine überlappenden Datenteile.

Horizontale Fragmentierung

Bei einer horizontale Fragmentierung wird ein Tupel mit all seinen Attributwerten auf ein Fragment verschoben. Die Vollständigkeit und Ausschließlichkeit sind bei einer horizontalen Fragmentierung dadurch gegeben, dass jedes Tupel auf genau ein Fragment verschoben wird. Durch eine Vereinigung aller Fragmente erhält man wieder die globale Relation, womit die Rekonstruierbarkeit erfüllt ist [Kud92] [TLRG08].

Vertikale Fragmentierung

Bei einer vertikale Fragmentierung werden dahingegen von jedem Tupel nur bestimmte Attributwerte auf ein gemeinsames Fragment verschoben. Es sind für alle Tupel die gleichen Attribute in den

gleichen Fragmenten [Kud92]. Um das Kriterium der Vollständigkeit bei einer vertikalen Fragmentierung zu erfüllen, muss jedes Attribut eines jeden Tupels in mindestens einem Fragment vorhanden sein. Die Rekonstruierbarkeit, also das Erstellen der globalen Relation aus seinen Fragmenten, kann durch eine Verbindung der Fragmente erreicht werden. Dadurch ist es allerdings notwendig, dass der Schlüssel der globalen Relation in allen Fragmenten vorhanden ist. Dadurch kann das Kriterium der Ausschließlichkeit bei einer vertikalen Fragmentierung nur bedingt eingehalten werden. Die Attribute, die redundant auf mehrere Fragmente verteilt werden sollten sich allerdings auf die Schlüssel beschränken. Für alle übrigen Attribute gilt dann wiederum die Regel der Ausschließlichkeit [Kud92] [TLRG08].

Gemischte Fragmentierung

Eine gemischte Fragmentierung entspricht dem Vorgehen, dass eine globale Relation entweder mit horizontaler oder vertikaler Fragmentierung aufgeteilt wird und die entstehenden Fragmente dann wiederum mit der entsprechend anderen Fragmentierungsart aufgeteilt werden. Diese Verschachtelung kann beliebig tief gehen, allerdings übertrifft sie in der Praxis nicht eine Tiefe von zwei [ÖV99].

Damit die Fragmentierung der Daten zu einer semantischen Gruppierung führt, sind neben dem verwendeten Verteilungsverfahren noch diverse Informationen von Nöten. Zu diesen Informationen gehören die Größe der globalen Relation sowie Zugriffshäufigkeiten für die einzelnen Daten. Unter der Zugriffshäufigkeit wird verstanden, wie oft und in welcher Kombination einzelne Attribute in Operationsprädikaten oder allgemein in Anfragen vorkommen [ÖV99]. So ist es das Ziel einer semantischen Gruppierung, die Daten in einer Gruppe (einem Fragment) zu haben, die häufig zusammen verwendet werden. Im Folgenden werden zwei verschiedene Arten der horizontalen Fragmentierung vorgestellt: die primäre und die abgeleitete horizontale Fragmentierung.

Primäre horizontale Fragmentierung

Eine primäre horizontale Fragmentierung bezieht sich auf eine sogenannte Owner-Relation, die keine Fremdschlüssel von anderen Relationen beinhaltet. Dies bedeutet, dass bei der Fragmentierung keine eventuell bereits gegebene Fragmentierung durch Zugriffe über Fremdschlüssel berücksichtigt werden müssen. Da eine primäre horizontale Fragmentierung einer Selektion entspricht, kann durch ein Selektionsprädikat angegeben werden, welche Daten zusammen bleiben. Man spricht dann von einem Fragmentierungsprädikat. Dieses Prädikat kann in ein Minterm-Prädikat überführt werden. Für eine optimale horizontale Fragmentierung ist es nun erforderlich, die passenden Minterm-Prädikate zu bestimmen [ÖV99]. Die Bestimmung der Minterm-Prädikate ist [Bra13a] zu entnehmen.

Abgeleitete horizontale Fragmentierung

Im Gegensatz zu der primären bezieht sich die abgeleitete horizontale Fragmentierung auf eine sogenannte Member-Relation, die einen Fremdschlüssel einer bereits fragmentierten Owner-Relation beinhaltet. Um eine solche Member-Relation unter Berücksichtigung der bereits vorhandenen Fragmentierung der Owner-Relation zu fragmentieren, werden so genannte *Semi-Joins* verwendet [ÖV99]. Eine Erläuterung dieses Vorgehens ist ebenfalls in [Bra13a] zu finden.

Auf die Frage hin, wie die einzelnen Tupel auf die einzelnen Instanzen verteilt werden, gibt es je nach Art der Fragmentierung wiederum verschiedene Verfahren. Für Algorithmen zur vertikalen Fragmentierung sei hier auf [ÖV99] verwiesen, während die wichtigsten Strategien bei der primären

horizontalen Fragmentierung, entnommen aus [TLRG08], im Folgenden kurz vorgestellt werden sollen.

Round-Robin

Das Round-Robin-Verfahren verteilt die Tupel der Reihe nach auf jeweils eine andere Instanz und beginnt nach der letzten Instanz wieder von vorne. Die Reihenfolge der Instanzen bleibt dabei bestehen. So wird während der Verteilung gewährleistet, dass sich die Größen der Fragmente maximal um ein Tupel unterscheiden. Allerdings ist die Verteilung der Tupel nur von ihrer Position innerhalb der Relation abhängig und entspricht somit keiner semantischen Gruppierung. Das Round-Robin-Verfahren eignet sich demnach sehr gut, wenn eine Gleichverteilung wichtiger als eine semantische Gruppierung ist.

Hash-Verfahren

Bei dem Hash-Verfahren werden die Tupel mit Hilfe einer festgelegten Hash-Funktion verteilt. Häufig ist diese Hash-Funktion eine Modulo-Rechnung, der ein (zusammengesetzter) Schlüssel und die Anzahl an zur Verfügung stehenden Fragmente zu Grunde liegt. Die Qualität der Hash-Funktion ist zum einen ausschlaggebend für die Unterschiede in den Fragmentgrößen und kann zum anderen dazu genutzt werden, die Tupel in semantische Gruppen zu unterteilen, da alle Tupel in ein Fragment verteilt werden, wenn sie den gleichen (zusammengesetzten) Schlüssel haben. Das Hash-Verfahren ist, anders als das Round-Robin-Verfahren, nicht gut geeignet, wenn es auf eine Gleichverteilung ankommt. Sein Vorteil liegt in der semantischen Gruppierung der Tupel durch die Fragmentierung. Insbesondere eignet es sich, wenn die Anfragen auf exakte Attributwerte und nicht auf Wertebereiche zielen.

Range-Verfahren

Das Range-Verfahren teilt die Tupel einer Relation in logische Bereiche auf. Die Unterschiede der Fragmentgrößen ist hier davon abhängig, wie gleichverteilt die Tupel bezüglich des entsprechenden Attributes sind. Dafür entspricht die Verteilung im Gegensatz zum Round-Robin-Verfahren einer semantischen Gruppierung. Das Range-Verfahren ist für eine Gleichverteilung das am schlechtesten geeignete Verfahren unter den hier vorgestellten. Sein Vorteil liegt, wie beim Hash-Verfahren, in der semantischen Gruppierung der Tupel durch die Fragmentierung. Insbesondere eignet es sich, wenn die Anfragen nicht auf exakte Attributwerte sondern auf Wertebereiche zielen.

2.7.2.3 Fragmentierung in einem DSMS

Die Fragmentierung von Daten in einem DSMS muss sich, anders als in einem DBMS, nicht mit Relationen, sondern mit Datenströmen befassen. Diese weisen allerdings gewisse Analogien zu Relationen auf. So entsprechen die Tupel ein und desselben Datenstromes einem bestimmten Schema, ähnlich wie alle Tupel einer Relation die gleichen Attribute haben [GJPPM+12]. Die Grundüberlegung für eine Fragmentierung im Kontext eines DSMS ist also die gleiche wie bei einem DBMS. Einzelne (in Datenströmen gebündelte) Daten mit bestimmten Attributen sollen auf verschiedene Fragmente verteilt werden. Im Folgenden werden die drei, in Abschnitt 2.7.2.2 vorgestellten, Fragmentierungsarten erneut betrachtet. Dieses Mal allerdings mit Hinblick auf eine Verwendung in einem DSMS.

Horizontale Fragmentierung

Da, wie bereits angesprochen, alle Tupel in einem Datenstrom einem gemeinsamen Schema entsprechen, können die Tupel analog zu einem DBMS auf verschiedene Instanzen zur Verarbeitung verteilt werden [GJPPM+12]. Wird jedes Tupel an genau eine Instanz zur Verarbeitung weitergeleitet, werden die Regeln der Vollständigkeit und Ausschließlichkeit eingehalten. Auch Rekonstruierbarkeit ist gegeben und zwar durch eine Vereinigung aller Teildatenströme nach deren Verarbeitung. Somit kann die horizontale Fragmentierung auch für DSMS angewendet werden. Allerdings ist dabei, wie schon bei der Replikation in Abschnitt 2.7.1.2, zu beachten, dass Datenströme zeitlich geordnet sind. Dies gilt nicht nur für eingehende, sondern auch für ausgehende Datenströme. Da die verschiedenen Instanzen durchaus unterschiedlich lange für die Verarbeitung von Tupeln benötigen, ist eine zeitliche Ordnung bei der Vereinigung nicht ohne Weiteres gegeben. Sie kann aber durch einen Zwischenspeicher realisiert werden, in dem die Tupel verweilen bis kein Tupel mehr aus einer verarbeitenden Instanz eintreffen kann, die in dem eingehenden Datenstrom vor dem entsprechenden Tupel eingeordnet war [Bra13a].

Vertikale Fragmentierung

Für DSMS kann die vertikale Fragmentierung ebenfalls Verwendung finden. Hierbei werden die Attribute aller Tupel eines Datenstromes auf verschiedene Instanzen zur Verarbeitung aufgeteilt. Da für eine abschließende Verbindung der entstandenen Teildatenströme Schlüssel benötigt werden, müssen Datenstromschemata diese für eine vertikale Fragmentierung enthalten. Verfügen die Datenströme über Schlüssel, können ebenfalls die Regeln der Vollständigkeit, der Rekonstruierbarkeit und der Ausschließlichkeit (mit Ausnahme der Schlüssel) eingehalten werden. Ist kein Schlüssel vorhanden, so kann der Datenstrom nicht vertikal fragmentiert werden. Um die zeitliche Ordnung des ausgehenden Datenstromes zu gewährleisten, kann ein Zwischenspeicher wie bei der horizontalen Fragmentierung verwendet werden [Bra13a].

Gemischte Fragmentierung

Da sowohl die horizontale als auch die vertikale Fragmentierung bei Datenströmen anwendbar sind, gilt dies auch für die gemischte Fragmentierung.

In Anschnitt 2.7.2.2 wurden außerdem verschiedene Verteilungsarten für eine primäre horizontale Fragmentierung vorgestellt: Round-Robin, Hash und Range. Diese sind ohne Probleme auf DSMS übertragbar, da sie sich auf die Attribute von Tupeln bzw. die Position von Tupeln in einer Relation beziehen [GJPPM+12] [Kos00]. Die räumliche Position eines Tupels entspricht dabei in einem DSMS seiner zeitlichen Position in einem Datenstrom.

2.7.3 Konzeptionelle Umsetzung in Odysseus

Das Datenstrommanagement-Framework Odysseus dient zur Erstellung maßgeschneiderter DSMS [BGJ+09]. Sowohl die Replikation als auch Verfahren der horizontalen Fragmentierung sind in Odysseus umgesetzt. Allerdings hat sich seit der ersten Umsetzung in Odysseus, welche in [Bra13a] be-

```
1  #PARSER PQL
2  #METADATA TimeInterval
3  #DODISTRIBUTE true
4
5  #PEER_PARTITION QUERYCLOUD
6  #PEER_MODIFICATION REPLICATION 2
7  #PEER_ALLOCATE ROUNDROBIN
8
9  #ADDQUERY
10 result = SELECT({PREDICATE = 'price > 50'}, nexmark:bid)
```

Listing 2.4: Die Verwendung der Replikation in Odysseus-Script

schrieben ist, einiges verändert, so dass dieses Kapitel einen kurzen Überblick geben soll. Odysseus verfügt in seiner Peer-To-Peer-Variante über einen modularen Anfrageverteiler, den `QueryDistributor` [Mic14]. Dieser Anfrageverteiler besteht aus mehreren, teils optionalen Modulen: Vorprozessoren, Partitionierern, Modifikatoren, Allokatoren und Postprozessoren. Diese einzelnen Module werden mittels der „Declarative Service Specification“ von Open Services Gateway initiative (OSGi) [Wüt08] eingebunden. So ist es zum einen als Entwickler möglich einzelne neue Module zu entwickeln ohne den Anfrageverteiler anpassen zu müssen und zum anderen kann der Nutzer die Kombination der einzelnen Module für seine konkrete Anfrage selbst bestimmen [Mic14].
Die erwähnten Modifikatoren verändern dabei den logischen Plan einer Anfrage, ohne allerdings den Ergebnisstrom zu verändern. Umsetzungen solcher Modifikatoren sind eben die Replikation, sowie die drei, in Abschnitt 2.7.2.2 und 2.7.2.3 beschriebenen, Verfahren der horizontalen Fragmentierung: Round-Robin, Hash und Range. Alle diese Modifikatoren arbeiten auf Grundlage einer Menge von Teilanfragen, die durch einen Partitionierer aus der ursprünglichen Anfrage gewonnen werden [Mic14]. Im Folgenden werden diese Modifikatoren näher betrachtet.

2.7.3.1 Replikation in Odysseus

Die Replikation in Odysseus dupliziert die einzelnen Teilanfragen und fügt für jeden ausgehenden Datenstrom einen Operator zur Synchronisation, den `ReplicationMerge`, ein. Die Funktionsweise dieses Operators ist in [Bra13a] beschrieben. Dieses Vorgehen hat zur Folge, dass, wenn eine Anfrage in mehrere aufeinander folgende Teilanfragen unterteilt wird, zwischen diesen Teilanfragen jeweils eine Synchronisation der Replikate stattfindet. Somit braucht für jede Teilanfrage lediglich ein Replikat verfügbar bleiben um weiterhin Ergebnisse der gesamten Anfrage zu erhalten.

Ein Beispiel für die Anwendung der Replikation ist in Listing 2.4 Zeile 6 zu sehen, wobei der einzige Parameter den Grad der Replikation, also die Anzahl der Replikate (inkl. des Originals), angibt. Die verwendete Skriptsprache „Odysseus-Script“ wird einführend in [Bra13a] beschrieben.

2.7.3.2 Fragmentierung in Odysseus

Die in Odysseus umgesetzten Verfahren der Fragmentierung duplizieren, wie die Replikation, ebenfalls Teilanfragen, allerdings nur jene, die einen zu fragmentierenden Datenstrom verarbeiten. Eine weitere Ausnahme bei der Duplizierung bildet die Quelle, die den zu fragmentierenden Datenstrom

```
1  #PEER_MODIFICATION FRAGMENTATION_HORIZONTAL_ROUNDROBIN
       ↪ nexmark:bid 2
2  #PEER_MODIFICATION FRAGMENTATION_HORIZONTAL_HASH nexmark:
       ↪ bid 2 price
3  #PEER_MODIFICATION FRAGMENTATION_HORIZONTAL_RANGE nexmark:
       ↪ bid.price 0 100 200
```

Listing 2.5: Die Verwendung der Fragmentierungsverfahren in Odysseus-Script

liefert. Dieser Quelle wird ein Operator zur Verteilung des Datenstromes auf die einzelnen Fragmente nachgeschaltet. Dieser Operator unterscheidet sich je nach gewähltem Fragmentierungsverfahren: `RoundRobin`-, `Hash`- oder `RangeFragment`. Die Funktionsweise dieser Operatoren ist ebenfalls in [Bra13a] beschrieben. Die Datenströme, die die einzelnen Fragmente verlassen, werden anschließend mit einem `Union` (einer Vereinigung) wieder zusammengeführt. Analog zur Replikation folgt aus diesem Vorgehen, dass zwischen aufeinander folgenden Teilanfragen die Fragmente jeweils zusammengeführt werden.

Listing 2.5 zeigt für jede der drei Fragmentierungsarten eine beispielhafte Verwendung, wobei die jeweilige Zeile die Zeile 6 in Auflistung 2.4 ersetzen kann. Das Round-Robin-Verfahren benötigt den Namen der Quelle, die den zu fragmentierenden Datenstrom liefert, und die Anzahl der zu erstellenden Fragmente. Zusätzlich benötigt das Hash-Verfahren noch diejenigen Attribute aus dem Datenstrom, die den Hashschlüssel bilden. Das Range-Verfahren benötigt eine detailliertere Angabe der Quelle, nämlich das Attribut, für das Bereiche (Ranges) gebildet werden sollen. Diese Bereiche werden durch ihre unteren Grenzen angegeben.

2.7.4 Zusammenfassung

Zusammenfassend wurden in diesem Abschnitt zunächst die wesentlichen Merkmale der Replikation und der Fragmentierung für DBMS herausgearbeitet und überprüft, inwiefern sich diese Konzepte in DSMS übertragen lassen. Es hat sich dabei herausgestellt, dass dies sehr wohl möglich ist. Allerdings geht es, wie bereits angesprochen, in einem DSMS um die Replikation bzw. Fragmentierung von Anfragen und nicht von Daten. Des Weiteren wurde das Konzept, mit dem die Replikation und Fragmentierung in dem Datenstrommanagementframework Odysseus umgesetzt wurde, beschrieben. [Bra13a] ist eine ausführlichere Beschreibung dieser Umsetzung zu entnehmen. In diesem Zusammenhang soll auch erwähnt werden, dass es weitere DSMS wie zum Beispiel „StreamCloud“, beschrieben in [GJPPM+12], gibt, die auf eine Parallelisierung und Fragmentierung zur Lastverteilung setzen. In Hinblick auf die Verwendung der Peer-To-Peer-Variante von Odysseus zur Lastverteilung ist mit dem modularen Anfrageverteiler sowie den bereits vorhanden Modifikatoren ein Grundstein gelegt. Und, wie bereits erwähnt, erlaubt der modulare Aufbau eine einfache Integration neuer Konzepte zur Partitionierung, Modifikation und Allokation. Vor dem Hintergrund der Replikation und Fragmentierung sind hierbei die Modifikatoren von besonderer Bedeutung. So erscheint es sinnvoll, weitere Verfahren der Fragmentierung als solche Modifikatoren zu entwerfen. Hier seien vor allem die abgeleitete horizontale und die vertikale Fragmentierung zu nennen.

Für das Szenario der Livesportanalyse mit Odysseus wird vor allen Dingen die Fragmentierung eine wichtige Rolle spielen, da eine serielle Verarbeitung mehrerer hundert Signale pro Sekunde und

pro Sender kaum möglich erscheint. Auch eine Replikation erscheint für dieses Szenario sinnvoll. So sind Datenstromobjekte, die einmal das System verlassen, ob regulär oder durch einen Ausfall der Verarbeitung, im Gegensatz zu Videoaufzeichnungen endgültig für eine Analyse verloren. Zwar kann man die Position eines Senders noch berechnen wenn nicht alle Signale erfasst werden, für das eigentliche Data Mining auf diesen Positionsdaten sollte allerdings eine gewisse Ausfallsicherheit vorhanden sein.

2.8 Load-Balancing

Möchte man Sportereignisse in Echtzeit analysieren oder kontinuierlich Daten von Finanzmärkten oder andere Sensordaten auswerten, greift man dazu i.d.R. auf ein DSMS, wie zum Beispiel Odysseus [AGG+12] zurück. Im Gegensatz zu DBMS, bei denen Anfragen auf einer endlichen Menge von Daten ausgeführt werden, analysieren DSMS Mengen potentiell unendlicher Datenströme in Echtzeit. Dabei fallen schnell enorme Datenmengen an, deren Analyse von einem einzelnen Rechner nicht mehr in akzeptabler Zeit bewältigt werden kann. Außerdem ist ein solches DSMS nicht ausfallsicher genug, um in kritischen Situationen eingesetzt zu werden: Fällt der Rechner aus, ist keine weitere Analyse möglich. Ein verteiltes DSMS kann dieses Problem lösen, indem die jeweiligen Berechnungen in einem Netzwerk auf verschiedene Knoten aufgeteilt werden können. Durch die Verteilung der Last können größere Mengen an Daten bewältigt werden und der Ausfall einzelner Knoten wird durch das Gesamtsystem aufgefangen. Dabei ist allerdings entscheidend, wie die Last auf die einzelnen Knoten verteilt wird [XZH05].

Bei einer ungünstigen Lastverteilung kann das volle Potential des DSMS Netzwerks nicht genutzt werden. Einerseits kann bereits ein einzelner überlasteter Knoten das ganze System ausbremsen, andererseits sind nicht ausgelastete Knoten auch nicht optimal, da zur Verfügung stehende Rechenleistung nicht genutzt wird und unnötige Kosten entstehen. Während Lastmanagement in anderen Anwendungsbereichen (z.B. bei Webservern) bereits weit verbreitet und entsprechend erforscht ist, stellen die besonderen Eigenschaften von Datenströmen ein Lastmanagement vor besondere Herausforderungen [XZH05].

In diesem Unterkapitel soll ein Überblick über Lastmanagement in verteilten DSMS gegeben werden. Zunächst wird dazu in Abschnitt 2.8.1 allgemein auf Lastmanagement eingegangen. Danach werden in Abschnitt 2.8.2 die speziellen Anforderungen in DSMS erläutert, die im Lastmanagement beachtet werden müssen. Anschließend wird in Abschnitt 2.8.3 mit der Potentialgetriebenen Lastverteilung [WSGÖ08] ein Ansatz zur Lastverteilung in DSMS genauer vorgestellt. Schließlich folgt in Abschnitt 2.8.4 eine kurze Zusammenfassung des Teilkapitels.

2.8.1 Eigenschaften von Lastmanagement

Dieser Abschnitt gibt zunächst ein Überblick über Lastmanagement im Allgemeinen. Ziel eines Lastmanagements ist es i.d.R. dafür zu sorgen, die an ein System gestellten Anfragen mit Hilfe verfügbarer Informationen so zu verteilen, dass die vorhandenen Ressourcen optimal genutzt werden. Gleichzeitig soll der Algorithmus selbst das System möglichst wenig zusätzlich belasten [Hać89].

Solch ein System könnte beispielsweise ein Verbund (Cluster) von Webservern sein, die redundant ausgelegt sind, d.h. alle über die gleichen Daten verfügen. Abbildung 2.19 zeigt ein Beispiel für ein

einfaches Lastmanagement in einem solchen System. Hier sorgt das Lastmanagement dafür, dass jeder dieser Server reihum abwechselnd angesprochen wird (Round-Robin). Da die Anfragen nur auf drei Server verteilt werden, wird die vierte Anfrage wieder dem ersten Server zugeordnet. Durch ein solches Vorgehen wird vermieden, dass einer der Server alle Anfragen bekommt, während die anderen Server nicht angesprochen werden. In diesem Beispiel wird allerdings vernachlässigt, dass die Knoten die Anfragen nicht unterschiedlich schnell beantworten können. So könnte Anfrage 1 beispielsweise sehr aufwändig sein, so dass die Zuteilung von Anfrage 4 an den gleichen Server möglicherweise nicht die optimale Entscheidung darstellt.

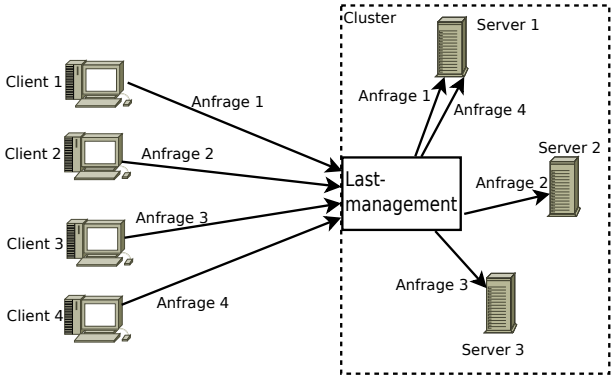


Abbildung 2.19: Beispiel für einfaches Lastmanagement

Um Last optimal auf verschiedene Knoten eines Clusters zu verteilen, müssen verschiedene Faktoren berücksichtigt werden, die sich je nach Anwendungszweck unterscheiden. Man unterscheidet dabei zunächst, ob es sich um einen Cluster mit loser oder enger Kopplung handelt. Enge Kopplung beschreibt einen Cluster, bei dem die einzelnen Knoten einander bekannt sind und Änderungen an einem Knoten i.d.R. das gesamte System betreffen. Ein Beispiel für eine solche enge Kopplung ist ein Mehrprozessorsystem, bei dem die einzelnen Knoten (Prozessoren) über einen gemeinsamen Bus miteinander kommunizieren. Die Erweiterung des Systems ist nicht möglich, ohne den gesamten Bus anzupassen. Bei der losen Kopplung sind die einzelnen Knoten weitgehend unabhängig voneinander. Dies ermöglicht i.d.R. eine bessere Skalierbarkeit und Austauschbarkeit einzelner Knoten. Diese Vorteile gehen allerdings meist zu Lasten der Performance und erhöhen die Komplexität, da die Kommunikation der einzelnen Knoten über möglichst allgemein gehaltene Protokolle stattfinden muss. Bei einem Cluster mit loser Kopplung kann es sich zum Beispiel um ein Netzwerk handeln, bei dem die einzelnen Rechner die Knoten sind, die über die Netzwerkleitungen miteinander kommunizieren [Hać89].

Die einzelnen Knoten innerhalb eines Clusters können homogen oder heterogen sein. Von homogenen Knoten spricht man, wenn alle Knoten gleichwertig sind, also über gleiche Kapazitäten (z.B. den gleichen Arbeitsspeicher und gleiche Prozessoren) verfügen und die gleichen Operationen durchführen können. Im Gegensatz dazu handelt es sich bei Knoten in einem heterogenen Netzwerk um unterschiedliche Knoten, die unterschiedlich stark belastet werden können oder nur bestimmte Aufgaben wahrnehmen können. Dies ist bei einer Lastverteilung entsprechend zu beachten, um zum Beispiel zu verhindern, dass ein Knoten eine Aufgabe zugewiesen bekommt, die für ihn nicht lösbar ist [Hać89].

Die Lastverteilung selbst kann statisch oder dynamisch erfolgen. Bei der statischen Lastverteilung wird die anfallende Last anhand vorher fest definierter Regeln (z.B. Round-Robin) auf die verschie-

denen Knoten verteilt. Eine Anpassung der Regeln während der Laufzeit findet nicht statt. Dies kann dazu führen, dass ein bereits überlasteter Knoten weitere Arbeit zugewiesen bekommen. Bei der dynamischen Lastverteilung wird dagegen kontinuierlich die Auslastung (und ggf. andere Informationen) der einzelnen Knoten überwacht und anhand dieser Information in Echtzeit über die Verteilung entschieden. Statische Ansätze werden vor allem dann genutzt, wenn sich die auftretende Last recht genau vorhersagen lässt. Ist dies nicht der Fall muss man i.d.R. auf aufwändigere, dynamische Verfahren zurückgreifen [Hać89].

Der Verteilungsalgorithmus selbst wiederum kann dabei zentral oder dezentral sein. Bei einem zentralen Verteilungsalgorithmus wird dieser Algoritmus auf einem einzelnen Knoten durchgeführt. Dieser leitet die Last dann an den vom Algorithmus ausgewählten Knoten weiter. Dezentrale Algorithmen werden verteilt auf den Knoten ausgeführt und sorgen durch Kommunikation der Knoten untereinander dafür, dass die Last aufgeteilt wird. Diese Algorithmen unterscheidet man in senderinitiierte und empfängeriniziierte Algorithmen. Bei den senderiniziierten Algorithmen sucht der überlastete Knoten einen weniger belasteten Knoten dem ein Teil der Arbeit übertragen werden kann. Diese Ansätze eignen sich vor allem für Cluster mit leichter bis mittlerer Last. Im Gegensatz dazu suchen bei den empfängerinitiierten Ansätzen die nicht ausgelasteten Knoten selbstständig nach überlasteten Knoten, denen sie Arbeit abnehmen können. Diese Ansätze sind eher für Systeme mit hoher Gesamtlast geeignet [Hać89]

Insgesamt zeigen diese unterschiedlichen Eigenschaften, dass es keinen universellen Ansatz für Lastmanagement gibt. Durch die vielen unterschiedlichen Umstände, die es zu berücksichtigen gilt, muss Lastmanagement immer der jeweiligen Situation angepasst werden. Der folgende Abschnitt 2.8.2 zeigt daher, welche besonderen Herausforderungen Datenströme an ein Lastmanagement stellen und welche Faktoren in diesem Kontext zu beachten sind.

2.8.2 Lastmanagement in verteilten DSMS

Nachdem in Abschnitt 2.8.1 auf die Grundlagen von Lastmanagement eingegangen wurde, werden in diesem Abschnitt die Besonderheiten für das Lastmanagement bei verteilten DSMS erläutert.

Da die Menge an eingehenden Daten bei der Analyse von Datenströmen sehr stark schwanken kann, muss das Lastmanagement dynamisch auf die jeweils aktuelle Situation reagieren. Daher ist ein statisches Lastmanagement für DSMS nicht sinnvoll. Bei nicht verteilten DSMS wird häufig auf Load Shedding zurückgegriffen. Dabei werden bei Überlast nicht mehr alle eingehenden Daten ausgewertet, sondern bestimmte, „unwichtig“ erscheinende Tupel vernachlässigt. Dies führt dazu, dass die Antwortzeiten in etwa konstant bleiben, dafür aber die Genauigkeit der Antwort leicht abnimmt. Diese Abstriche müssen bei verteilten DSMS vermutlich nicht gemacht werden, da mit zusätzlichen Knoten mehr Rechenleistung zur Verfügung gestellt wird [KKP11].

Bei verteilten DSMS handelt es sich i.d.R. um ein Netzwerk mit loser Kopplung: Die einzelnen Knoten sind eigenständige Rechner, die über ein Netzwerk miteinander kommunizieren können. Ob diese Knoten homogen oder heterogen sind hängt allerdings vom jeweiligen DSMS ab. Da Odysseus auf verschiedensten Rechnern lauffähig ist, diese aber unter Umständen nicht die volle Funktionalität anbieten wird im Folgenden von einem heterogenen, verteilten DSMS ausgegangen. Außerdem sind die einzelnen Knoten autonom und können sich beliebig vom Netzwerk an- oder abmelden.

In klassischen DBMS, Webservern und den meisten anderen Anwendungsbereichen von Lastmanagement wird eine Anfrage einmal gestellt, die benötigten Ressourcen abgefragt und zurückgeliefert. Das Lastmanagement nimmt dabei grundsätzlich eine Anfrage entgegen, untersucht sie nach den nachgefragten Ressourcen und leitet die Anfrage an die Knoten mit der geringsten Last weiter. In DSMS wird eine Anfrage stattdessen dauerhaft im System platziert und kontinuierlich auf die eingehenden Datenströme ausgeführt. Dadurch ändert sich die Aufgabe des Lastmanagements für DSMS dahingehend, dass die Anfrage selbst verteilt werden muss, während die Daten jeweils an die entsprechenden Teile der Anfrage weitergeleitet werden [KKP11].

Anfragen innerhalb eines DSMS sind i.d.R. als zyklenfreie Anfragebäume aufgebaut, die aus einzelnen Operatoren bestehen. Abbildung 2.20 zeigt ein Beispiel eines solchen Baums: `SRC_1` und `SRC_2` stellen die Datenquellen dar, auf die jeweils mit einem `ACCESS` Operator zugegriffen wird. Dann legen `WINDOW` Operatoren die Fenster für die Datenströme fest. Nach einem `JOIN` der Daten wird eine `SELECT` Operation ausgeführt, bevor die Daten am `SINK` ausgegeben werden.

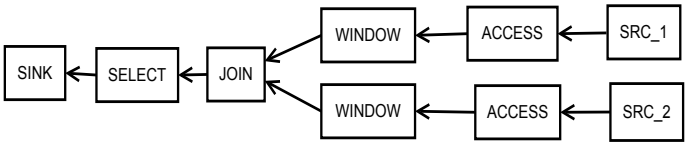


Abbildung 2.20: Beispielanfrage als Anfragebaum

Möchte man die Rechenlast einer solchen Anfrage auf verschiedene Rechner eines Netzwerks verteilen, bietet es sich an, die verschiedenen Operatoren auf die einzelnen Knoten aufzuteilen. Dabei können einem Knoten je nach Bedarf ein oder mehrere Operatoren zugewiesen werden. Eine solche Verteilung ist i.d.R. nicht eindeutig, daher müssen Kriterien gefunden werden, die einen Vergleich der unterschiedlichen Varianten ermöglichen. Da das Ziel einer solchen Zuordnung eine möglichst optimale Aufteilung der Rechenlast ist, kann die Qualität einer Verteilung zum Beispiel durch die Latenz der Gesamtanfrage gemessen werden [WSGÖ08].

Um eine möglichst niedrige Latenz zu erreichen, muss ein Lastmanagementalgorithmus möglichst viele Einflussfaktoren der Gesamtlatenz berücksichtigen. Dabei sind für verteilte DSMS vor allem die folgenden Gesichtspunkte ausschlaggebend:

Durchschnittliche Auslastung: Die im Kontext von DSMS aufzuteilende „Last“ ist vor allem die Rechenleistung, die zur Auswertung eines Operators notwendig ist. Ist ein Knoten überlastet, müssen eingehende Tupel in einer operatorspezifischen Warteschlange gesammelt werden und die weitere Verarbeitung verzögert sich. Um eine möglichst schnelle Verarbeitung der eingehenden Tupel zu erreichen, ist es sinnvoll, die Warteschlagen der Operatoren möglichst kurz zu halten. Deshalb wird in regelmäßigen Abständen die Auslastung der einzelnen Operatoren gemessen. Die Summe dieser Einzelwerte ergibt die Gesamtauslastung des Knotens [WSGÖ08].

Lastvarianz: Die auszuwertende Datenmenge ist nicht konstant sondern hängt von den Datenquellen ab und kann stark variieren: Ein Bewegungssensor in Ruheposition sendet zum Beispiel nur wenige Daten. Dagegen kann der gleiche Sensor in Bewegung mitunter enorme Datenmengen erzeugen, die es auszuwerten gilt. Daher muss die Verteilung laufend angepasst werden. Um die Lastvarianz einzelner Knoten zu minimieren, wird versucht die Operatoren so auf die einzelnen

Knoten zu verteilen, dass Operatoren mit ähnlichen Lastverläufen auf unterschiedlichen und Operatoren mit unterschiedlichen Lastverläufen auf den gleichen Knoten platziert werden. Dadurch gleichen sich die Lastverläufe aus und die durchschnittliche Varianz wird reduziert. Dazu kann der Korrelationskoeffizient zwischen den jeweiligen Lastverläufen berechnet werden. Operatoren mit niedrigen Koeffizient sollen entsprechend nah beieinander und solche mit hohem Korrelationskoeffizienten möglichst weit voneinander entfernt platziert werden [XZH05].

Nutzung der Netzwerkverbindung: Liegen in einer Anfrage aufeinander folgende Operatoren auf verschiedenen Knoten, müssen die Zwischendaten über das Netzwerk transportiert werden. Liegen beide Operatoren auf dem selben Knoten kommt es zu keiner Verzögerung durch das Netzwerk. Durch Minimierung der durchschnittlichen Nutzung der Netzwerkverbindung kann versucht werden, aufeinander folgende Operatoren nah beieinander zu platzieren [WSGÖ08].

Umverteilungskosten: Die Bewegung eines Operators zu einem anderen Netzwerkknoten erfordert, dass der derzeitige Zustand des Operators über das Netzwerk zum Zielknoten geschickt werden. Erst wenn die Latenzgewinne durch die Verringerung der Lastvarianz und durchschnittlichen Auslastung die Kosten eines solchen Transports übersteigen lohnt sich die Neuplatzierung des Operators [IS11].

Insgesamt gilt es also beim Lastmanagement von Datenströmen einige Besonderheiten zu beachten: Zum einen kann das Datenaufkommen der jeweiligen Quellen stark schwanken, gleichzeitig soll aber die Latenz möglichst konstant bleiben um eine Analyse in Echzeit zu ermöglichen. Zudem unterscheidet sich die Analyse von Datenströmen erheblich von anderen Anwendungsfeldern des Lastmanagement, da hier die Anfragen dauerhaft im System installiert werden müssen und nicht nur einmalig ausgeführt werden. Die entscheidenden Faktoren für die Latenz einer Anfrage sind die durchschnittliche Auslastung und die Lastvarianz der Knoten, die Nutzung der Netzwerkverbindungen und gegebenenfalls die Umverteilungskosten der Operatoren.

Im nächsten Absatz wird mit der potentialgetriebenen Lastverteilung [WSGÖ08] ein Ansatz für das Lastmananagement verteilter DSMS vorgestellt.

2.8.3 Potentialgetriebene Lastverteilung

Ein Lösungsansatz für die in Abschnitt 2.8.2 beschriebenen Herausfoderungen beim Lastmanagement in DSMS ist die potentialgetriebene Lastverteilung. Die Idee zu diesem Ansatz stammt aus der Physik: Analog zu Magneten die sich abstoßen oder anziehen können oder Federn, die gespannt sind, wirken „Kräfte“ auf die Operatoren. Immer wenn das Potential dieser Kräfte zu groß wird, wechselt der Operator in einen anderen Knoten. Die Kräfte werden so gewählt, dass sie den Kriterien für eine ideale Lastverteilung entsprechen, so dass durch eine Verringerung des Kraftpotentials der Operatoren eine Verbesserung der Lastverteilung erreicht wird. Insgesamt strebt das Gesamtsystem ständig einen Gleichgewichtszustand an, in dem alle Kräfte möglichst klein werden. Dieser Gleichgewichtszustand entspricht der optimalen Operatorenverteilung. [WSGÖ08]

Abbildung 2.21 zeigt schematisch, wie die Umverteilung eines Operators in diesem Ansatz funktioniert. Die drei Abschnitte zeigen jeweils die gleichen beiden Knoten eines Netzwerks, auf denen die Operatoren `JOIN`, `SELECT` und `ACCESS` verteilt sind. Im ersten Abschnitt sind `JOIN` und `SELECT` auf Knoten eins und `ACCESS` auf Knoten zwei. Zur Vereinfachung soll nur der `SELECT` Operator

näher betrachtet werden. Die Auslastung von Knoten eins ist normal, die auf den SELECT Operator wirkendende Kraft, mit Energiepotential(EP) gekennzeichnet, ist zwar vorhanden, reicht aber nicht aus um den Operator in (den weniger ausgelasteten) Knoten zwei zu ziehen. Im zweiten Abschnitt ist Knoten eins überlastet (z.B. durch eine erhöhte Last beim JOIN Operator). Diese erhöhte Last führt zu einem höheren Energiepotential. Dies reicht hier tatsächlich aus, um den SELECT Operator zu bewegen. Der dritte Abschnitt zeigt wie der Operator von Knoten eins zu Knoten zwei gewandert ist. Dadurch wird die Last auf Knoten eins wieder verringert und eine Lastumverteilung hat stattgefunden.

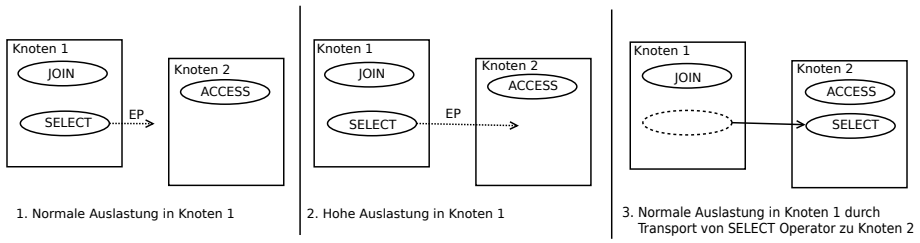


Abbildung 2.21: Potentialgetriebene Lastverteilung (Beispiel)

Da in Odysseus die einzelnen Knoten autonom sind, muss beachtet werden, dass die Operatoren den Knoten nicht einfach zugewiesen werden können. Stattdessen könnte man Knoten, z.B. über ein geeignetes Protokoll, fragen, ob sie den entsprechenden Operator übernehmen. Die angefragten Knoten könnten dann zustimmen oder ablehnen und ggf. müsste eine weniger gute Verteilung gewählt werden. Da dies allerdings für das Berechnen einer möglichst optimalen Verteilung zunächst keine Rolle spielt, wird darauf nicht weiter eingegangen.

Im Folgenden wird nun zunächst beschrieben, wie das Energiepotential eines Operators berechnet wird. Danach wird erläutert, wie die einzelnen Operatoren mit Hilfe dieses Energiepotentials auf die verschiedenen Knoten verteilt werden.

2.8.3.1 Energiepotential eines Operators

Die drei Kriterien, die in diesem Algorithmus berücksichtigt werden sind durchschnittliche Auslastung eines Knotens, dessen Lastvarianz und die Inanspruchnahme der Netzwerkverbindung. Der Ansatz selbst ist aber um weitere Faktoren ergänzbar, es müssen nur die entsprechenden Kräfte definiert werden. Das Energiepotential eines Operators besteht entsprechend aus drei Teilpotentialen: Das *Lastpotential* bildet die durchschnittliche Auslastung eines Knotens ab, das *Korrelationspotential* hilft dabei, die Lastvarianz eines Knotens durch geschickte Verteilung der Operatoren zu minimieren und das *Netzwerkpotential* berücksichtigt die Nutzung der jeweiligen Netzwerkverbindungen [WSGÖ08].

Das Lastpotential eines Operators o auf einem Knoten h in einer Plazierung P, ist in [WSGÖ08] definiert als

$$\text{Lastpotential}(o,h,P) = \begin{cases} \frac{ld_h}{q_h} & \text{wenn } \frac{ld_h}{q_h} < 1 \\ 1 + p \cdot (\frac{ld_h}{q_h} - 1) & \text{sonst.} \end{cases}$$

Dabei bezeichnet ld_h die durchschnittliche Last des Knotens, q_h ist die Solllast des Knotens. Die Fallunterscheidung sorgt dafür, dass das Potential bei einem nicht voll ausgelasteten Knoten zunächst linear bis zu einem Wert von 1 ansteigt. Wird ein vorher festgelegtes Sollpotential überschritten (der Knoten ist überlastet) wird zur jeweilige Überlast ein zusätzlicher Straffaktor p multipliziert. Durch einen hohen Straffaktor wird erreicht, dass das Potential bei einem Überschreiten der Solllast überproportional stark ansteigt.

Das Korrelationspotential eines Operators ist nach [WSGÖ08] definiert als der Korrelationskoeffizient zwischen der Last des Operators und der Summe der Einzellasten aller anderen Operatoren auf dem gleichen Knoten:

$$\text{Korrelationspotential}(o, h, P) = \rho(o, h)$$

Das Netzwerkpotential eines Operators ist 0, falls alle nachfolgenden Operatoren auf demselben Knoten liegen. Ansonsten ist das Potential definiert als

$$\text{Netzwerkpotential}(o, h, P) = \sum_{l \in L} c(l) \quad \text{mit} \quad c(l) = \begin{cases} d_l & \text{wenn } u_l \leq 1 \\ d_l \cdot u_l & \text{sonst.} \end{cases}$$

Dabei ist L die Menge der ausgehenden Netzwerkverbindungen, d_l ist die jeweilige Verzögerung einer Leitung und u_l der Nutzungsgrad der zur Vefügung stehenden Bandbreite einer Leitung. Solange ausreichend Bandbreite zur Verfügung steht ($u_l \leq 1$) bestimmt allein die Leitungsverzögerung das Potential, bei einer überlasteten Leitung wird diese Verzögerung aber nochmal mit dem jeweiligen Nutzungsgrad der Leitung multipliziert [WSGÖ08].

Das Gesamtpotential eines Operators ergibt sich aus einer gewichteten Summe der festgelegten Einzelpotentiale:

$$\begin{aligned} \text{Gesamtpotential}(o, h, P) = w_l \cdot \text{Lastpotential}(o, h, P) \\ + w_k \cdot \text{Korrelationspotential}(o, h, P) \\ + w_n \cdot \text{Netzwerkpotential}(o, h, P). \end{aligned}$$

Um zu gewährleisten, dass alle Faktoren in die Platzierung einfließen, müssen die einzelnen Gewichtungsfaktoren so gewählt werden, dass alle Potentiale in etwa gleichen Einfluss auf das Gesamtpotential haben [WSGÖ08].

Auf diese Weise lässt sich für jeden Operator ein solches Energiepotential berechnen. Der nächste Teilabschnitt beschreibt, wie dieses Energiepotential zur Verteilung der Operatoren genutzt werden kann.

2.8.3.2 Verteilungsalgorithmus

Ziel des Verteilungsalgorithmus soll es sein, eine Platzierung der einzelnen Operatoren zu finden, die ein möglichst geringes Gesamtpotential aufweist. Dazu wird so vorgegangen, dass zunächst eine erste, initiale Verteilung gefunden wird, die dann laufend angepasst wird. Noch vor der initialen Verteilung werden die Operatoren einmal zufällig verteilt, damit die für die tatsächliche Verteilung benötigten Daten gesammelt werden können [WSGÖ08].

Da das Finden einer optimalen initialen Verteilung der Operatoren sehr komplex ist, nutzt man dazu heuristische Verfahren. Dazu schlägt [WSGÖ08] zwei Ansätze vor: einen Greedy Algorithmus und dynamische Programmierung. Beide Ansätze gehen von einer geordneten Liste der Operatoren aus. Es bietet sich an, die Operatoren absteigend nach durchschnittlicher Last zu ordnen, da sich Operatoren mit weniger Last leichter verteilen lassen als Operatoren mit viel Last. Beide Algorithmen gehen außerdem davon aus, dass alle Knoten am Anfang leer sind.

Der Greedy Algorithmus findet nacheinander für jeden Operator den Knoten mit dem geringsten Energiepotential und platziert den Operator auf diesem Knoten. Dies wird für jeden Operator genau ein mal durchgeführt, so dass am Schluss alle Operatoren zugeordnet sind [WSGÖ08].

Die dynamische Programmierung notiert die minimalen Gesamtpotentiale einer Platzierung in einer Tabelle. Nacheinander werden dabei sämtliche Platzierungsmöglichkeiten für einen Operator ausprobiert, bevor dann mit der minimalen Gesamtsumme und dem nächsten Operator fortgefahren wird. Die Spalte, welche in der letzten Reihe die minimale Gesamtsumme aufweist wird ausgewählt [WSGÖ08].

Die kontinuierliche Anpassung der initialen Verteilung kann grundsätzlich mit den gleichen Algorithmen erfolgen. Um dabei allerdings die jeweils vorherige Verteilung mit einzubeziehen und zu gewährleisten, dass die Kosten für eine Neuverteilung nicht den Nutzen überschreiten, sind $\lfloor \theta \cdot Operatorzahl \rfloor$ Operatoren fest. Bei diesen festen Operatoren handelt es sich um die Operatoren mit der niedrigsten Ordnung, also diejenigen Operatoren, die selbst am wenigsten Einfluss auf die Gesamtlast haben. Die Zahl θ ist ein vom Nutzer vorgegebener Wert, der zur Feineinstellung genutzt werden kann [WSGÖ08].

Insgesamt wird auf diese Weise ein dynamisches Lastmanangement realisiert, dass den in Abschnitt 2.8.2 geschilderten Herausforderungen gewachsen ist. Gleichzeitig bleibt die Möglichkeit, durch die Definition zusätzlicher Teilpotentiale noch andere Faktoren zu berücksichtigen.

2.8.4 Zusammenfassung

Eine komplexe Analyse von Datenströmen in Echtzeit erfordert i.d.R. sehr viel Rechenleistung. Da ein einzelner Rechner dadurch schnell überlastet ist, nutzt man verteilte DSMS, die die Arbeit auf mehrere Knoten aufteilen. Um eine möglichst optimale Verteilung der Last zu gewährleisten ist ein Lastmanagement dabei unumgänglich.

Daher wurde in Abschnitt 2.8.1 zunächst das klassische Lastmanamgent erläutert. Dabei werden Anfragen innerhalb eines Rechnerverbunds möglichst so an die einzelnen Knoten verteilt, dass alle Knoten möglichst gleichmäßig ausgelastet werden. Für ein optimales Lastmanagement müssen dabei je nach Anwendungsfall eine Menge Faktoren berücksichtigt werden, zum Beispiel die Art der Kopplung oder ob ein statisches oder dynamisches Lastmanagement gewählt wird.

In Abschnitt 2.8.2 wurde auf die Besonderheiten beim Lastmanagement für verteilte DSMS eingegangen. Im Gegensatz zum klassischen Lastmanagement werden die Anfragen in DSMS dauerhaft im System platziert und ausgewertet. Insbesondere die Tatsache, dass Datenströme zum Teil enorme Schwankungen in der Datenmenge aufweisen erweist sich dabei als problematisch und muss vom Lastmanagement berücksichtigt werden.

Schließlich wurde mit der Potential-getriebenen Lastverteilung in Abschnitt 2.8.3 ein möglicher Lösungsansatz für diese Problematik vorgestellt. Dieser Ansatz basiert auf der Idee, dass auf die Operatoren physische Kräfte wirken, die versuchen die Operatoren auf andere Knoten zu ziehen. Diese Kräfte sind dabei so gewählt, dass sie die drei Faktoren durchschnittliche Auslastung, Lastvarianz und die Nutzung der Netzwerkverbindung berücksichtigen. Werden diese Kräfte zu groß, bewegt sich der Operator auf den Knoten mit dem geringeren Kraftpotential. Dadurch gelingt es, das System laufend an veränderte Lastverhältnisse anzupassen und dabei die meisten der relevanten Faktoren für DSMS zu berücksichtigen. Gleichzeitig bleibt der Ansatz über die Wahl geeigneter Kräfte anpassbar.

Zusammenfassend kann gesagt werden, dass die Potential-getriebene Lastverteilung eine geeignete Möglichkeit darstellt, ein Lastmanagement in einem verteilten DSMS wie Odysseus zu realisieren.

2.9 Recovery

Fehler können durch Ausfall von Hardware oder fehlerhafter Software entstehen. Für diese Fehlerfälle gibt es bezüglich DBMS verschiedene Schutzmechanismen. Diese sorgen dafür, dass die Datenbanken in einem konsistenten Zustand gehalten werden. DSMS verarbeiten Datenströme, die kontinuierlich von der Außenwelt an das System geliefert werden, unabhängig vom aktuellen Zustand des Systems. Damit keine oder möglichst wenig Daten durch das Auftreten von Fehlern verloren gehen, sind Techniken der Fehlerbehandlung notwendig, um ein effektives Recovery zu erreichen.

In diesem Teilkapitel wird in Abschnitt 2.9.1 zunächst eine Differenzierung der Fehlerarten vorgenommen und weiterhin das Rollback Recovery für Transaktionen sowie Absicherungsmöglichkeiten bei Fehlern an Massenspeichern beschrieben. In Abschnitt 2.9.2 wird eine Abgrenzung der Anforderungen von DSMS gegenüber DBMS vorgenommen. Es werden mögliche Fehler in DSMS identifiziert und das grundsätzliche Verhalten im Fehlerfall beschrieben und eine Klassifikation für Recovery-Techniken gegeben. In Abschnitt 2.9.3 werden verschiedene Techniken zum Recovery in DSMS sowie ihre Unterschiede und Einsatzmöglichkeiten vorgestellt. Zum Abschluss folgt eine kurze Zusammenfassung.

2.9.1 Datenbank Recovery in klassischen DBMS

Fehler an Datenbanken können durch Hardware- oder Softwareprobleme auftreten und Inkonsistenzen in der Datenbank hervorrufen. Werden während solch eines Fehlerfalls Transaktionen ausgeführt, müssen diese bei der Fehlerkorrektur behandelt werden [Zeh98]. Ein Ansatz ist, ein Rollback aller ungültigen Transaktionen durchzuführen. Eine übliche Variante wird in Teilabschnitt 2.9.1.1 detaillierter beschrieben.
Zur Absicherung gegenüber potentiellen Platten- oder Maschinenfehlern (Medienfehler) werden in modernen Datenbank-Systemen Sicherheitsmaßnahmen wie Duplikation beziehungsweise Replikation eingesetzt, um vor dem Ausfall eines einzelnen Massenspeichers geschützt zu sein [KBL06]. Sollte das Hauptgerät ausfallen, kann das Backup-System übernehmen. Oft wird das Backup-System auch an einem physisch weit entfernten Ort betrieben.

2.9.1.1 Log-basierte Rollback Recovery

Für alle Transaktionen, die während des Absturzes aktiv und noch nicht abgeschlossen waren, findet ein Rollback statt. Um festzustellen, welche Veränderungen diese Transaktionen bisher an der Datenbasis getätigt haben, wird jede Aktion in Logs geschrieben. Rollbacks werden auch eingesetzt wenn die Transaktion anderweitig, beispielsweise durch den Nutzer oder wegen eines Deadlocks, abgebrochen wird.

Als Sicherheitsmaßnahme werden in DBMS Logs benutzt, welche auf beständige Speichermedien geschrieben werden und einen Systemabsturz überstehen [KBL06]. Oft wird auch doppelt auf zwei verschiedene Geräte geschrieben, um gegen einen einzelnen Ausfall sicher zu sein. In sogenannten Write-Ahead Logs werden Veränderungen, die durch eine Transaktion vorgenommen werden, zuerst in das Log geschrieben, bevor sie vom Hauptspeicher zur Datenbank geschickt werden. Das soll verhindern, dass die Datenbasis verändert wird, ein Absturz auftritt, aber noch kein Log Eintrag geschrieben wurde und somit ein Rollback nicht möglich wäre. Wobei es hier strikte Umsetzungen gibt, die dieses Prinzip konsequent befolgen, und wiederum Systeme, in denen Einträge zunächst in einen im Hauptspeicher befindlichen Log Buffer gehalten werden. Von dort werden die Änderungen zu einem späteren Zeitpunkt im Log festgeschrieben. Dies geschieht spätestens bevor die Transaktion abgeschlossen ist. Der letzte Ansatz reduziert Overhead, erschwert jedoch den Recovery-Prozess [KBL06].

Damit während der Recovery-Prozedur das Log nicht vollständig abgearbeitet werden muss, werden periodisch Speicherpunkte erstellt, welche ebenfalls in das Log geschrieben werden. Daraus ist ersichtlich, welche Transaktionen zum Zeitpunkt des Speicherpunktes aktiv waren. Von diesem Speicherpunkt aus können mit Hilfe der im Log aufgezeichneten Veränderungen, nicht abgeschlossene Transaktionen rückgängig gemacht werden.

Abbildung 2.22 zeigt ein Beispiel für dieses Verfahren. Wir gehen von striktem Write-Ahead-Logging aus. Die Recovery-Prozedur fängt an der letzten Stelle im Log an. Dort ist ein Update Eintrag für T1 und danach einer für T6. Daher weiß die Prozedur, dass diese beiden Transaktionen zum Zeitpunkt des Absturzes aktiv waren. Da diese jedoch nicht abgeschlossen wurden, macht sie die dadurch entstandenen Veränderungen rückgängig. T4 ist zwar im Speicherpunkt vorhanden, wurde jedoch vor dem Absturz abgeschlossen. Der Speicherpunkt zeigt, dass neben T1 und T6 auch noch T3 aktiv war. Auch für diese Transaktion liegt kein Commit-Eintrag vor. Die Recovery-Prozedur räumt für diese drei Transaktionen alle entstandenen Veränderungen auf. T2 und T5 werden nicht mehr überprüft, da diese Transaktionen bereits vor dem Speicherpunkt abgeschlossen beziehungsweise abgebrochen wurden.

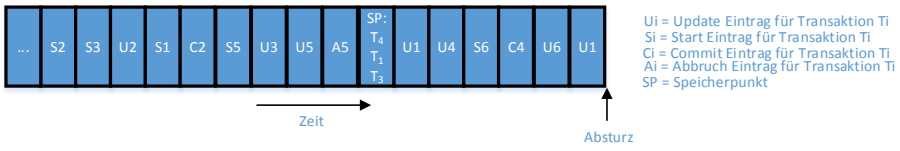


Abbildung 2.22: Log Rollback Recovery, in Anlehnung an [KBL06], Seite 824

2.9.1.2 Recovery im Falle eines Medienfehlers

Varianten der Wiederherstellung nach einem Medienfehler [Stö01]:

- **Komplettes/partielles Recovery:** Die Wiederherstellung einer kompletten Datenbank in einen transaktionskonsisten Zustand wird als komplettes Recovery bezeichnet. Werden nur Teile der Datenbank und nicht die komplette Datenbank wiederhergestellt, so wird dies als partielles Recovery bezeichnet.

- **Point-In-Time-Recovery:** Point-In-Time-Recovery bezeichnet die Wiederherstellung einer Datenbank in einen transaktionskonsistenten Zustand zu einem bestimmten Zeitpunkt in der Vergangenheit.

- **Online-Recovery:** Ist es möglich während der Wiederherstellung Schreibzugriffe auf bestimmte, konsistente Bereiche der Datenbank auszuführen, so spricht man von Online-Recovery.

- **Paralleles Recovery:** Werden mehrere Sicherungsmedien zur Wiederherstellung eingesetzt, findet ein Paralleles Recovery statt.

2.9.2 Recovery in DSMS

Die Fähigkeit, einen konsistenten Zustand zu erreichen, ist grundlegend für Datenbank-Systeme. Versucht man nun dieses Konzept auch auf DSMS anzuwenden, gibt es jedoch bedeutende Unterschiede, die es zu beachten gilt [ABB+04]. DBMS bauen auf Transaktionen auf, die die ACID Eigenschaften besitzen. Anfragen werden meist nur einmal ausgeführt. Diese Eigenschaften lassen sich aber nicht direkt auf kontinuierliche Anfragen übertragen, da hier ein permanenter Datenstrom abläuft, der auch dann Daten an das System sendet, wenn dieses aufgrund eines Ausfalls oder Ähnliches nicht aktiv ist. In DSMS sind die kontinuierlichen Anfragen Bestandteil des persistenten Zustands des Systems und dürfen als solche nicht verloren gehen [ABB+04]. In den folgenden Kapiteln wird von verteilten DSMS ausgegangen. Diese können beispielsweise wie in Abbildung 2.23 aufgebaut sein.

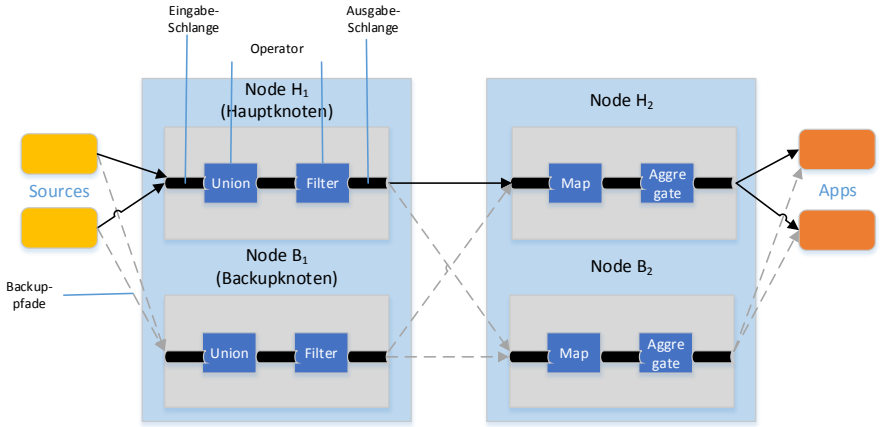


Abbildung 2.23: Verteiltes DSMS

2.9.2.1 Kategorisierung von Fehlern

Innerhalb eines verteilten DSMS können Fehler in unterschiedlicher Form durch verschiedene Quellen wie beispielsweise Software-,Hardware- oder Netzwerkfehler auftreten. Diese Fehler treten auch in verteilten DBMS auf, jedoch unterscheiden sie sich in der Art der Fehlerbehandlung. Balazinska, Hwang und Shah beschreiben diese Fehler nach folgenden Fehlermodellen [BHS08]:

Prozessknoten-Fehler:

- Fail-stop Fehler: Der Prozessknoten stoppt die Ausführung und verliert seine gespeicherten Zustände, inklusive der Anfragen. Andere Knoten können den Fehler feststellen.
- Byzantine Fehler: Der Prozessknoten besitzt ein fehlerhaftes Verhalten, welches sich zum Beispiel in falschen Ausgaben äußern kann. Für diese Fehler gibt es noch kaum Lösungen für DSMS, obwohl Ansätze existieren [BHS08].

Kommunikations-Fehler: Innerhalb eines Kommunikationsnetzwerks, wie es bei einem verteilten DSMS vorliegt, kann es zu Verzögerungen im Austausch oder sogar den Verlust von Nachrichten kommen. Um letzteres zu verhindern, wird oft auf zuverlässige Protokolle wie TCP zurückgegriffen, welche jedoch das Problem der Verzögerungen nicht lösen. Netzwerk-Fehler können auftreten, wenn einzelnen Knoten keine Verbindung zum Netzwerk herstellen können oder eine Partition des Netzwerks auftritt, bei der verschiedene Teilnetzwerke voneinander getrennt werden. Die Beschreibung der Fehlerholung nach einem Netzwerk-Fehler wird im Rahmen dieser Arbeit nicht beschrieben, wird aber unter anderem in [BHS08] behandelt.

2.9.2.2 Fehlererkennung und Phasen des Recoverys

In Systemen mit asynchroner Kommunikation ist es dafür notwendig Kontrollmechanismen einzurichten, um unterscheiden zu können, ob ein Knoten langsam, abgestürzt oder vom Netzwerk getrennt ist. Dieses Problem wird in vielen System darüber gelöst, dass die Knoten sich untereinander sogenannte *keep-alive* Nachrichten schicken, um festzustellen, ob der jeweils andere Knoten noch im Netzwerk aktiv ist [BHS08].

Wenn der Ausfall eines Knotens festgestellt wird, wird die **Take-Over-Phase**, also die Übernahme der Anfragen eines Knoten durch einen anderen Knoten, eingeleitet [SHB04]. Dieser setzt die Verarbeitung der Anfragen fort. Wie diese Phase im Detail aussieht hängt von der eingesetzten Recovery-Technik ab, von denen einige in Abschnitt 2.9.3 beschrieben werden.

Erholt sich der ausgefallene Knoten oder wird er ersetzt, beginnt die **Catch-Up-Phase**. Dabei wird der Grad der Fehlertoleranz, welcher vor dem Ausfall gegeben war, wiederhergestellt [SHB04]. Dies kann je nach Recovery-Technik mit unterschiedlich hohem Aufwand verbunden sein.

2.9.2.3 Grad des Recoverys

Im Folgenden gehen wir davon aus, dass in einem Anfrage-Netzwerk jeweils Paare von Prozessknoten, ein Hauptknoten H und ein Backup-Knoten B, gegeben sind. Der Hauptknoten H bearbeitet ein Anfragefragment Q und ein Set von Eingabe-Tupeln (I_1, I_2, ... I_n) , welche bei regulärer Bearbeitung die Ausgabe A erzeugen. H fällt während der Ausführung aus, B stellt dies nach einiger Zeit über eine keep-alive Nachricht fest und übernimmt die Bearbeitung. Die erzeugte Ausgabe von H ist A_H

und die von B erzeugte Ausgabe nach der Übernahme ist A_B. Es lassen sich nun drei verschiedene Güteklassen des Recoverys beschreiben [HBR+05]:

Precise Recovery: Die stärkste Recovery-Güteklasse wird Precise Recovery genannt, da es denselben Output generiert, als wenn es keinen Fehler gegeben hätte. B bearbeitet nur die Eingabe-Tupel, die von H noch nicht ausgegebenen wurden. Hier gilt $A_H + A_B = A$.

Rollback Recovery: Eine etwas schwächere Güteklasse des Recoverys ist das Rollback Recovery. Es stellt immer noch sicher, dass kein Informationsverlust stattfindet. Jedoch kann es dazu kommen, dass B Eingabe-Tupel verarbeitet, die schon von H bearbeitet wurden und so Duplikate in der Ausgabe erzeugt oder B eine andere Abarbeitungsfolge durchführt und so zu einer von H verschiedenen Ausgabe gelangt. Hier gilt $A_H + A_B = A'$.

Gap Recovery: Die einfachste Form des Recoverys ist die Gap Recovery. B führt die Verarbeitung der Eingabe-Tupel an der Stelle fort, wo die Übernahme stattgefunden hat. Dabei werden alle Eingabe-Tupel, welche während des Fehlers gesendet wurden, nicht berücksichtigt. Auch hier gilt $A_H + A_B = A'$.

2.9.3 Recovery-Techniken für verteilte DSMS

Nachfolgend werden verschiedene Techniken zur Recovery bei einem Ausfall eines Prozess-Knotens bei der Bearbeitung von Datenströmen in verteilten DSMS genannt. Diese Techniken decken verschiedene Grade der Recovery ab. Bei der Beschreibung der Techniken wird wiederum von der Verfügbarkeit von Knotenpaaren wie in Kapitel 3.3 ausgegangen.

2.9.3.1 Amnesia

Amnesia ist die einfachste Technik, um eine hohe Verfügbarkeit bei der Bearbeitung zu erreichen. Sie ist in Abbildung 2.23 dargestellt. Im Falle eines Fehlers eines Prozessknotens übernimmt der Backup-Knoten die Verarbeitung der Eingabe-Tupel. Dieser startet die Bearbeitung aus einem leeren Zustand heraus, kann also direkt beginnen. Die Anzahl der verlorenen Tupel wird dabei von der Zeit bis zur Fehlererkennung und der Übertragungsrate der Tupel bestimmt.
Grad der Recovery: Amnesia führt immer zu einer Gap Recovery.
Recovery Zeit: Die Recovery-Zeit sei definiert als der Zeitraum zwischen des Feststellens des Fehlers durch den Backup-Knoten bis zur Verarbeitung der Eingabe-Tupel. Diese Zeit beträgt hier null, wenn die Zeit zum Umleiten der Tupel vernachlässigt wird. Da der Backup-Knoten vorher zustandslos ist, kann er direkt mit der Verarbeitung der Eingabe-Tupel beginnen.
Overhead: Kein Overhead vorhanden.
Anwendung: Dieses Verfahren wird unter anderem in *Mortar* eingesetzt [LY08].

2.9.3.2 Passive Standby

Beim Passive Standby speichert der Hauptknoten seinen Zustand in periodischen Abständen und sendet die Veränderungen seit dem letzten Speicherpunkt an den Backup-Knoten. Dies ist in Abbildung 2.24 illustriert. Der Backup-Knoten kann bei einem Fehler die Bearbeitung von dem letzten Speicherpunkt aus fortsetzen. Dabei gilt es zu vermeiden, dass die Bearbeitung durch den Hauptknoten während der Speicherung unterbrochen wird [HBR+05]. Die Ausgabe-Warteschlange des Vorgänger-

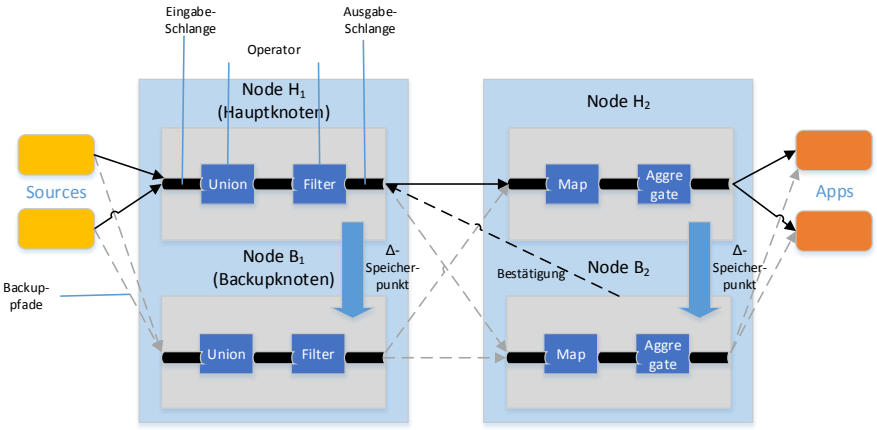


Abbildung 2.24: Passive Standby

Knoten hält solange Input-Tupel vor, bis der Backup-Knoten eine Bestätigung sendet, dass diese in dem letzten Speicherpunkt enthalten sind.
Grad des Recoverys: Passive Standby unterstützt das Rollback Recovery und kann zum Precise Recovery ausgebaut werden. Eingabe-Tupel werden so lange vorgehalten, bis bestätigt ist, dass sowohl der Hauptknoten als auch der Backup-Knoten diese erhalten haben. Dies ist der Fall, wenn der Hauptknoten die zu bearbeitenden Eingabe-Tupel in den neuen Speicherpunkt schreibt und der Backup-Knoten diesen Speicherpunkt erhält. Auf diese Weise können bei einem Fehler des Hauptknotens die noch nicht ausgetauschten Eingabe-Tupel vom Backup-Knoten aus dem übergeordneten Knoten gelesen und die Bearbeitung ab dem letzten Speicherpunkt fortgesetzt werden.
Recovery Zeit: Passive Standby hat eine kurze Recovery-Zeit, da der Backup-Knoten einen sehr aktuellen, kompletten Speicherpunkt des Zustandes des Hauptknotens besitzt. Sie ergibt sich aus der Zeit, die benötigt wird, um die nicht durch den letzten Speicherpunkt abgedeckten Eingabe-Tupel erneut zu verarbeiten, bevor neue Tupel verarbeitet werden können.
Overhead: Der Overhead für die Verarbeitung besteht aus dem Einpflegen der Änderungen der Speicherpunkt Nachrichten. Der Overhead an Bandbreite ergibt sich aus dem Austausch der Speicherpunkte und ist somit abhängig von Größe und Intervall der Speicherpunkte.

2.9.3.3 Active-Standby

Active-Standby, dargestellt in Abbildung 2.25, unterscheidet sich vom Passive Standby dadurch, dass sowohl der Hauptknoten als auch der Backup-Knoten die Eingabe-Tupel erhalten und diese parallel verarbeiten. Der Backup-Knoten hält dabei jedoch seine Ausgabe-Tupel zurück. Da es in nicht deterministischen Systemen zu verschiedenen Ausgaben kommen kann, synchronisieren der Hauptknoten und der Backup-Knoten ihre Ausgabe-Tupel in regelmäßigen Abständen. Die Ausgabe-Warteschlange des Vorgänger-Knotens hält solange Input-Tupel vor, bis beide Knoten eine Bestätigung senden, dass diese verarbeitet sind. Nimmt der vorherige Hauptknoten seine Arbeit wieder auf, wird er zum Backup-Knoten und muss zunächst einen Rückstand an Eingabe-Tupeln aufholen, bis er wieder synchron zum neuen Hauptknoten läuft.
Grad des Recoverys: Active-Standby unterstützt ebenfalls das Rollback Recovery, da der Backup-Knoten parallel die Verarbeitung der Eingabe-Tupel ausführt und bei einem Fehler des Hauptknoten

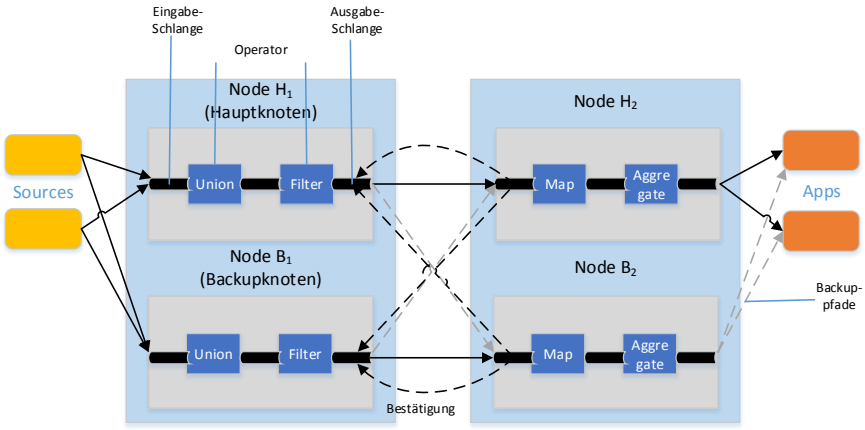


Abbildung 2.25: Active Standby

übernehmen kann. Da es in nicht deterministischen Systemen Unterschiede in den Ausgabe-Tupeln geben kann, müssen für ein Precise Recovery Mechanismen eingerichtet werden, die die Ausgabe abstimmen.

Recovery Zeit: Die Recovery-Zeit beim Active-Standby besteht aus der Zeit, die benötigt wird, um noch nicht synchronisierte, duplizierte Ausgabe-Tupel, die sich in der Ausgabe des Backup-Knotens befinden, erneut zu senden.

Overhead: Der Overhead für die Verarbeitung und die benutzte Bandbreite liegt bei 100%, da der Backup-Knoten ebenfalls die Verarbeitung ausführt.

Anwendung: Active-Standby wird unter anderem in *Borealis* eingesetzt. Borealis ist ein System für die Verarbeitung von verteilten Datenströmen [AAB+05].

2.9.3.4 Upstream-Backup

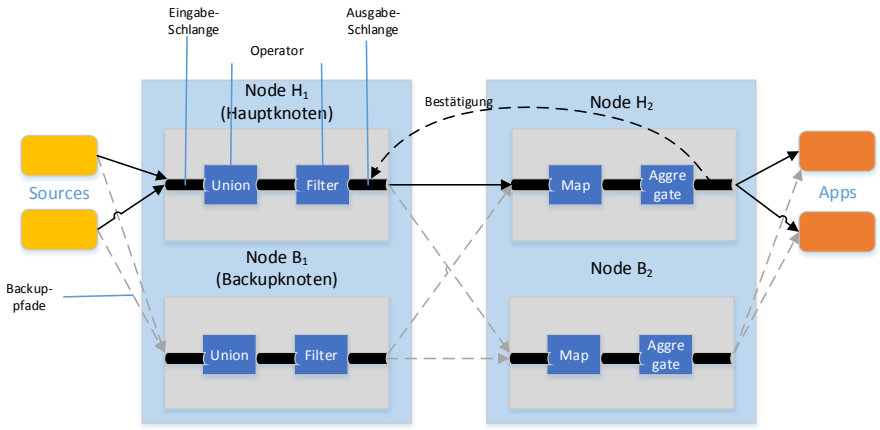


Abbildung 2.26: Upstream-Backup

Beim Upstream-Backup werden Eingabe-Tupel solange in der Ausgabe-Schlange der vorgelagerten Stufe gespeichert, bis der Hauptknoten die erfolgreiche Verarbeitung in der letzten Stufe gemeldet

hat. Diese Technik ist in Abbildung 2.26 dargestellt. Fällt der Hauptknoten nun aus, kann ein Backup-Knoten nun die Verarbeitung wiederholen, indem es die Eingabe-Tupel von der vorgelagerten Stufe erneut erhält. Dabei wird der Backup-Knoten zum neuen Hauptknoten und setzt die Verarbeitung fort. Wenn der ehemalige Hauptknoten wieder in Betrieb ist, wird er zum neuen Backup-Knoten und hat einen leeren Zustand. Die Schwierigkeit bei dieser Technik ist es, festzustellen, welche Eingabe-Tupel aus der Ausgabe-Schlange der vorgelagerten Stufe entfernt werden können [HBR+05].
Grad der Recovery: Upstream-Backup unterstützt das Rollback-Recovery. Der Backup-Knoten startet aus einem leeren Zustand und führt Operatoren erneut aus. Über entsprechende Mechanismen kann es zum Precise Recovery ausgebaut werden.
Recovery Zeit: Die Recovery-Zeit ist beim Upstream-Backup am höchsten, da der Backup-Knoten aus einem leeren Zustand heraus die Verarbeitung erneut starten muss. Die Anzahl der dadurch entstehenden redundanten Ausgabe-Tupel entspricht dabei der Anzahl, der Tupel, die der Backup-Knoten erneut verarbeiten muss.
Overhead: Der Overhead für die Verarbeitung und die Bandbreite sind gering, da lediglich übermittelt werden muss, welche Tupel aus der Ausgabe-Schlange des übergeordneten Knotens entfernt werden können.
Anwendung: Upstream Backup wird in *QGSA-DQP*, einem System für verteilte Prozessverarbeitung eingesetzt [SW05].

2.9.3.5 Zusammenfassung

Die vorgestellten Recovery-Techniken eignen sich für unterschiedliche Anwendungsfälle. Amnesia mit seiner Gap Recovery, der maximalen Verfügbarkeit sowie keinem Overhead eignet sich beispielsweise für Sensornetzwerke, in denen die Sensoren ununterbrochen Datenströme liefern und es nicht relevant ist, wenn einzelne Eingaben verloren gehen, wie etwa bei der Messung der Luftfeuchtigkeit oder Temperatur über einen längeren Zeitraum.

Bei den anderen drei Techniken ist es möglich die Eingabe verlustfrei zu verarbeiten. Sie unterscheiden sich vor allem in der Recovery-Zeit und dem Overhead bei der Verarbeitung. Active-Standby hat einen großen Overhead und eine sehr kurze Recovery-Zeit. Daher ist es für Systeme geeignet, die hohe Laufzeitkosten im Gegenzug für ein schnelles, verlustfreies Recovery akzeptieren können. Ein Einsatz wäre beispielsweise für Finanzdienste, wie an der Börse, oder beim Militär mit ihren hohen Kapazitäten denkbar. Passive Standby hat hierbei eine längere Recovery-Zeit und einen noch höheren Overhead als Active-Standby, lässt sich jedoch am einfachsten in ein Precise Recovery überführen und ist somit für Anwendungen im medizinischen Bereich, wo genaue Ergebnisse benötigt werden, geeignet [HBR+05]. Ein Beispiel wäre die Auswertung eines EKGs. Upstream-Backup hat eine etwas längere Recovery-Zeit und einen geringen Overhead. Damit ist es für Systeme geeignet, die nicht die maximale Verfügbarkeit benötigen und kurze Verzögerungen erlauben können, die jedoch dennoch möglichst genaue Ergebnisse benötigen.

2.9.4 Zusammenfassung

Dieses Teilkapitel zeigt, dass Recovery in DBMS und DSMS mit verschiedenen Herausforderungen einhergeht. Recovery in DBMS muss vor allem die Einhaltung der ACID Eigenschaften und insbesondere die der Konsistenz absichern. Dies geschieht auf Transaktionsebene durch Methoden wie das Logging und das Setzen von Speicherpunkten und bei Medienfehlern über Techniken wie Replikati-

on. Diese Ansätze sind schon sehr weit ausgereift. In DSMS bedeutet Recovery eine Abwägung zwischen schneller Erholung, Grad der Recovery und dem mit der Absicherung verbundenen Overhead. Ziel ist, die Anfrageverarbeitung schnell und zuverlässig fortführen zu können. Hierzu wurden Techniken vorgestellt, die Recovery bei einem Ausfall eines Prozessknotens in einem verteilten DSMS behandeln. Die Wahl der Technik sollte hierbei auf Grundlage des geplanten Einsatzfeldes getroffen werden. Weitere Herausforderungen wie Netzwerkfehler wurden genannt. Die Forschung im Bereich der Recovery in (verteilten) DSMS ist noch relativ neu und bietet immer noch sehr viel Potenzial zur Optimierung.

2.10 Peer Computing

Big Data ist ein aktuelles Schlagwort, mit dem die Speicherung und Analyse großer Datenmengen bezeichnet wird [FKU14]. Diese stammen z.B. aus Unternehmensanwendungen oder Sensornetzwerken. Um der Herausforderung der Verarbeitung solcher Datenmengen zu begegnen gibt es einige Ansätze. Einer ist das Peer-to-Peer (P2P)-Computing. Hierbei verbinden sich mehrere Computer, Peers genannt, miteinander und lösen die Aufgaben in einem Netzwerk gemeinsam. Dies hat einige Vorteile verglichen mit dem Ansatz, die Aufgaben auf nur einem, aber dafür sehr leistungsstarken Rechner auszuführen. Die Kosten für die Hardware können auf die Besitzer verteilt werden, außerdem sind die Computer günstiger, da sie zum einen nicht die aktuellste Hardware benötigen und zum anderen häufig schon existierende Rechner genutzt werden können. Die Rechenkapazität ist skalierbarer, da einfach weitere Peers in das Netzwerk eintreten können. Aufgrund der Verteilung ist das Netzwerk zudem ausfallsicherer als ein einzelner Rechner. Wegen dieser Vorteile bieten sich P2P-Systeme für Big Data Anwendungen an.

Dieses Teilkapitel soll einem Überblick über P2P geben. Dazu werden im folgenden Abschnitt die grundlegenden Merkmale von P2P-Systemen erläutert und im Abschnitt 2.10.2 unterschiedliche Architekturtypen untersucht. Der 2.10.3. Abschnitt beschäftigt sich mit Routing-Algo-rithmen, die in P2P-Netzwerken zum Suchen von verteilten Ressourcen genutzt werden. In Abschnitt 2.10.4 werden die Grundlagen für verteiltes Rechnen erläutert und auf verteiltes Data Mining im speziellen eingegangen.

2.10.1 Merkmale

P2P-Systeme haben einige Eigenschaften und Merkmale, die je nach Umsetzung unterschiedlich stark ausgeprägt sind. Wichtig für das Verständnis von Peer-to-Peer ist der Begriff *Peer*. Ein Peer ist ein Teilnehmer des Netzwerkes aus vielen Peers, der *gleichberechtigt* mit anderen Peers Ressourcen der anderen nutzt und selbst Ressourcen zur Verfügung stellt.

Dynamischer Aufbau P2P-Systeme sind sehr dynamisch. Das Netzwerk muss damit umgehen können, dass die Peers häufig und unvorbereitet Verbindungen aufbauen und abbrechen, auch wenn gerade auf einem Peer eine Berechnung ausgeführt wird [GS02].

Gleichberechtigung Alle Peers sind gleichberechtigt. Kein Peer nimmt eine gesonderte Rolle ein, jeder Peer stellt Ressourcen zur Verfügung und nutzt die Ressourcen von anderen. Je nach Architektur kann dies jedoch abweichend umgesetzt sein [GS02].

Skalierbarkeit Das Ziel eines P2P-Netzes ist es, sehr skalierbar zu sein, um problemlos wachsen bzw. sich verkleinern zu können. Der dynamische Aufbau begünstigt gute Skalierbarkeit im Gegensatz zu Ansätzen mit zentralen Servern [GS02].

Ausfallsicherheit Ein P2P-Netzwerk sollte möglichst ausfallsicher sein. Durch die Abwesenheit von zentralen Komponenten und den dynamischen Aufbau ist dies eher gewährleistet als bei zentralen Servern [GS02].

Fehlertoleranz Verhält sich ein Peer fehlerhaft, z.B. indem er falsche Ergebnisse zurückliefert, so sollte dies das Netzwerk möglichst wenig beeinträchtigen [GS02].

Interoperabilität Die Peers, die sich in dem Netzwerk verbinden, sind unterschiedlich und stellen unterschiedliche Ressourcen zur Verfügung [GS02].

Cost of Ownership Die Kosten des Netzwerkes sind verteilt. Jeder Peer trägt einen Teil zu dem Netzwerk bei, (teure) zentrale Komponenten sind nicht notwendig [GS02].

Anonymität Je nach Aufbau kann in dem Netzwerk weitgehende Anonymität in verschiedenen Ausprägungen gewährleistet werden. So kann z.B. der anfragende Peer einer Ressource anonym bleiben [GS02].

Effektivität und Effizienz Die Aufgaben sollten in dem P2P-Netzwerk möglichst Effektiv und Effizient gelöst werden. Zum Beispiel sollten Suchen in dem Netzwerk schnell und mit möglichst wenig Datenverkehr ausgeführt werden können [GS02].

Die vorgestellten Merkmale sind je nach P2P-System unterschiedlich stark ausgeprägt. Die Art der Systemarchitektur ist entscheidend, um die Merkmale eines P2P-Systems zu bestimmen. Die gängigen P2P-Architekturtypen werden im folgenden Abschnitt vorgestellt.

2.10.2 P2P-Architekturen

P2P-Systeme reichen von zentralisierten bis hin zu vollständig dezentralisierten Architekturen. Zentralisierte Architekturen verbinden die Eigenschaften von Client-Server Architekturen mit den Vorteilen von P2P-Systemen. Dezentralisierte P2P-Architekturen hingegen kommen ohne zentrale Instanz aus und sind damit besonders skalierbar, robust und immun gegen den Ausfall von zentralen Komponenten. Die im folgenden beschriebenen Probleme dieser Architekturansätze sorgten für die Entwicklung von hybriden Systemen, die versuchen, die Vorteile beider Varianten zu vereinen. Da einige Peers die Rolle von kleinen Servern einnehmen, liegen sie zwischen zentralen und dezentralen Systemen [VLO10].

2.10.2.1 Zentrale P2P-Systeme

In zentralen P2P-Systemen gibt es einen oder mehrere zentrale Server, die das Netzwerk verwalten. Im Folgenden wird von einem einzelnen Server ausgegangen, wie in Abbildung 2.27 zu sehen. Jeder Peer verbindet sich mit dem zentralen Server und teilt diesem die eigenen Ressourcen mit, die von ihm zur Verfügung gestellt werden. In der Abbildung wird dies mit den durchgezogenen, geraden Verbindungen dargestellt. Sucht ein Peer eine Ressource, so fragt er beim Server an. Dieser teilt dem Anfragenden mit, wer die gesuchte Ressource zur Verfügung stellt. Anschließend kommunizieren

die Peers direkt miteinander, ohne auf den Server angewiesen zu sein. Nutzt man P2P für verteiltes Rechnen, kann der Server z.B. Aufgaben an die Peers verteilen, die diese dann selbstständig lösen [VLO10].

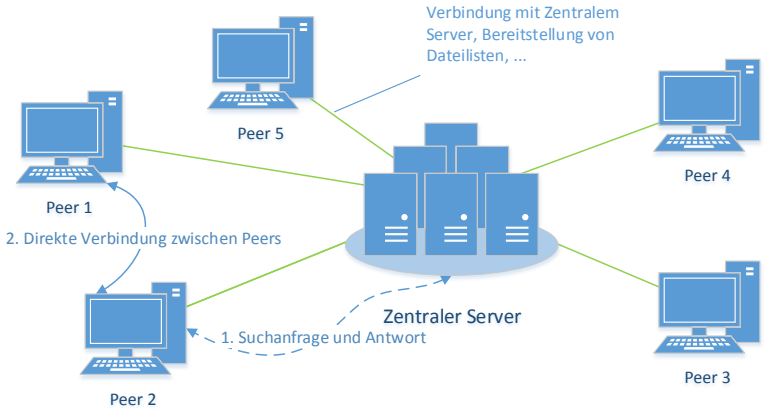


Abbildung 2.27: Zentrale P2P-Architektur

Der Vorteil dieser Architektur liegt darin, dass die Suche nach Ressourcen und die Verwaltung des Netzwerkes einfach ist. Der zentrale Server kennt alle Peers und alle Ressourcen und kann somit schnell die richtigen Peers vermitteln. Jedoch hat das Netzwerk damit einen Schwachpunkt bei der Skalierbarkeit und der Ausfallsicherheit. Der zentrale Server muss höhere Lasten in einem wachsenden Netzwerk an Peers bewältigen können. Fällt der zentrale Server aus, funktioniert das Netzwerk nicht mehr und die Peers können nicht arbeiten. Als Beispiel für ein solches Netzwerk kann das frühere Musiktauschportal Napster sowie verteiltes Rechnen wie SETI@home[13] herangezogen werden [VLO10].

2.10.2.2 Dezentrale P2P-Systeme

In völlig dezentralen P2P-Systemen sind alle Peers gleichberechtigt, es gibt keine zentralen Server oder besondere Peers zur Verwaltung. Die Peers verbinden sich untereinander und kennen jeweils einige andere Peers, die sogenannten Nachbarn. Um sich mit einem solchen Netzwerk zu verbinden, ist es notwendig, mindestens einen Peer zu kennen, der bereits Teil des Netzwerkes ist. Abbildung 2.28 zeigt ein Beispiel einer dezentralen Struktur.

Kritisch an einem vollständig dezentralen System ist vor allem das Suchen von Ressourcen. Da es keine Stelle gibt, die von jeder Ressource weiß, auf welchem Peer sie zur Verfügung gestellt wird, gestaltet sich das Auffinden von Ressourcen deutlich aufwendiger und zeitintensiver als z.B. bei einem zentralen P2P-System. Auf die Suchalgorithmen, die versuchen, trotzdem in kurzer Zeit vollständige Ergebnisse zu liefern, wird in Abschnitt 2.10.3 eingegangen. Die Vorteile eines dezentralen Systems liegen darin, dass sie besonders robust und vor allem beim Aspekt Informationsaustausch vor Zensur geschützt sind. Ohne zentrale Komponenten ist ein Ausfall des Systems unwahrscheinlich. Außer-

[13] `http://setiathome.ssl.berkeley.edu/`, Zuletzt abgerufen am 23.04.2014

dem kann es problemlos wachsen, ist also sehr skalierbar - was jedoch auch die Geschwindigkeit der Suche beeinträchtigen kann.

2.10.2.3 Hybride P2P-Systeme

In hybriden P2P-Systemen gibt es einige Peers, die erweiterte Aufgaben wahrnehmen. Häufig werden sie *Super-Peers* genannt. Dies sind für gewöhnlich einige wenige Peers, die z.B. eine höhere Rechenleistung und eine schnellere Netzwerkverbindung haben. Sie übernehmen Verwaltungsaufgaben für einfache Peers, meistens um die Suche der Ressourcen zu beschleunigen. Im Gegensatz zu einem zentralen Server kennen sie dabei aber nicht die Ressourcen aller Peers, sondern nur die einer Gruppe, für die sie zuständig sind. Im Gegensatz zu einem reinen Server haben sie zudem die Aufgaben und Möglichkeiten eines normalen Peers - sie stellen also Ressourcen zur Verfügung und nutzen Ressourcen von anderen. Es gibt einige unterschiedliche Möglichkeiten, Super-Peers zu nutzen. Sie können z.B. nur für seltene Ressourcen im Netzwerk zuständig sein. Dann werden häufige Ressourcen über einen normalen Suchalgorithmus gesucht, seltene Ressourcen jedoch mit Hilfe der Super-Peers. Auch gibt es Ansätze, in denen sie nicht die Suche beschleunigen, sondern dafür sorgen, dass die Nachbarschaft für Peers möglichst optimal aufgebaut ist, um Anfragen schnell zu beantworten (vgl. „BestPeer“, [AD02]) [VLO10].

Die Vorteile eines hybriden Systems liegen darin, dass die Suche beschleunigt werden kann, ohne die dezentrale Struktur von P2P-Systemen aufzugeben. Damit haben sie eine niedrigere Ausfallwahrscheinlichkeit als zentrale P2P-Systeme und sorgen trotzdem für schnellere und umfangreichere Suchergebnisse. Fallen die Super-Peers z.B. aus, kann trotzdem ohne sie in dem Netzwerk gesucht werden - nur eben langsamer, wie bei einem vollständig dezentralen P2P-System.

2.10.3 Routing

Gibt es keine zentrale Stelle, die von jeder Ressource weiß, auf welchem Peer sie zu finden ist, müssen zum Suchen die Peers des Netzwerkes befragt werden. Um dies schnell, mit wenig Bandbreitenaufwand und geringem Speicherplatzaufkommen zu ermöglichen, gibt es unterschiedliche Ansätze. Diese unterscheiden sich zuerst in der Netzwerktopologie, auf der sie aufbauen. In diesem Zusammenhang wird häufig der Begriff *Overlay-Netzwerk* genutzt. Aufbauend auf dem genutzten technischen Netzwerk, z.B. dem Internetprotokoll, wird ein logisches Netzwerk gebildet. Dieses Netzwerk bestimmt z.B. die Namen der Peers (die dann z.B. anstelle der IP-Adressen zur Kommunikation genutzt werden) und die Struktur der Nachbarschaften. Das Overlay-Netzwerk kann strukturiert oder unstrukturiert sein. In unstrukturierten Netzwerken haben Peers beliebige Ressourcen, die sie selbst bestimmen können, z.B. eine Menge von Dateien. Außerdem bestimmen sie meistens selbst, mit welchen anderen Peers sie sich verbinden. In strukturierten Netzwerken hingegen werden die Ressourcen den Peers zugeordnet, was zusätzlichen Aufwand zum Herstellen und Erhalten der Struktur bedeutet, jedoch die Suche deutlich beschleunigen kann [VLO10].

2.10.3.1 Routing in unstrukturierten Netzwerken

In unstrukturierten P2P-Netzwerken gibt es zwei grundsätzliche Suchalgorithmen, auf die auch die fortgeschritteneren Varianten aufbauen: Die Tiefen- und Breitensuche. Dabei wird die Suchanfrage in unterschiedlicher Reihenfolge an die Peers in der Nachbarschaft weitergeleitet. Um zu verhindern,

dass eine Suchanfrage endlos weitergeschickt wird, hängen die meisten Algorithmen ein *Time-to-live (TTL)* an die Suchanfrage an, das bestimmt, wie lange eine Suchanfrage weitergeschickt wird. Das TTL gibt z.B. die Anzahl an Sprüngen (Hops) im Netzwerk an, die die Suchanfrage maximal weitergeschickt wird. Jeder Knoten verringert das TTL um 1. Ist der Wert 0 erreicht, wird die Anfrage nicht mehr weitergeleitet. Das Ziel von Routing-Strategien ist es, die Ergebnisse mit einem beschränkten TTL zu optimieren [VLO10].

Breitensuche

Bei der Breitensuche wird eine Suchanfrage von jedem Knoten jeweils ausgeführt und an alle Nachbarknoten weitergeleitet, bis ein Knoten ein Ergebnis zurückliefern kann. Die Anzahl der Knoten, die eine Suchanfrage ausführen, wächst mit der Tiefe exponentiell an. In Abbildung 2.28 wird eine solche Suche ausgeführt. Peer 1 stellt eine Anfrage für eine Ressource, die auf Peer 5 zu finden ist. Peer 1 befragt seine Nachbarschaft (Peer 2 und Peer 4) und diese jeweils wieder ihre Nachbarn. Über Peer 2 kommt die Anfrage bei Peer 5 an, dieser sendet die Antwort über den rückwärtigen Weg der Anfrage zurück. Wie der rückwärtige Weg konkret vonstatten geht, unterscheidet sich je nach System und muss nicht zwangsläufig über die zuvor genutzt Peers gehen. Der suchende Peer kann z.B. auch direkt von dem Peer mit der Ressource angesprochen werden [VLO10].

Ein Beispiel für ein unstrukturiertes Overlay mit einfacher Breitensuche ist das Gnutella-System[14]. Ein bekannter Client, der mit diesem arbeitete, war z.B. LimeWire. Beim Beitreten des Netzwerkes verbindet sich ein Peer mit zufälligen anderen Peers, die auf ein Ping-Signal antworten. Die Verwendung der einfachen Breitensuche hat zur Folge, dass etwa 70% des Datenverkehrs nur für die Suche genutzt werden [VLO10].

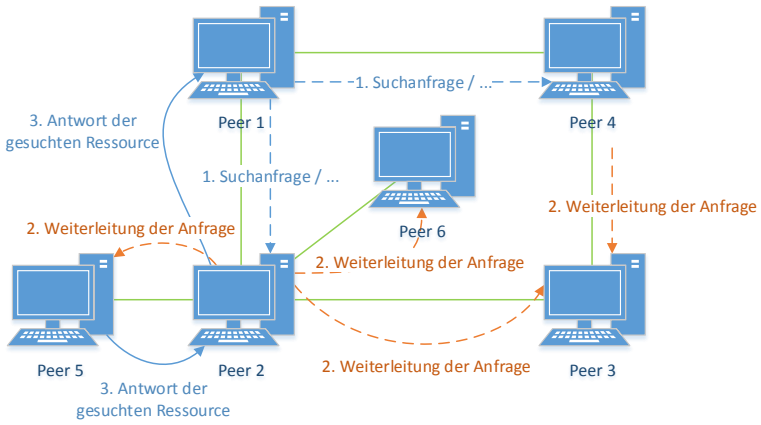


Abbildung 2.28: Routing in einem dezentralen, unstrukturierten P2P-System mit Breitensuche

[14] `http://rfc-gnutella.sourceforge.net/`, Zuletzt abgerufen am 27.04.2014

Tiefensuche

Hierbei wird die Suchanfrage nicht an alle Nachbarn weitergeleitet, sondern nur an einen bestimmten. Welcher Nachbar ausgewählt wird, hängt von der implementierten Strategie ab. Die Suchanfrage wird so lange an jeweils den nächsten Nachbarn weitergeleitet, bis die TTL abgelaufen ist. Liefert der ausgewählte Nachbar innerhalb einer bestimmten Zeit kein Ergebnis oder meldet, dass die Ressource nicht gefunden werden konnte, wird die Anfrage an den nächsten Nachbarn weitergeleitet [VLO10]. Ein Problem bei dieser Methode ist, dass Anfragen zwischen Peers im Kreis laufen können. Im dem vorzubeugen, können z.B. mit der Anfrage Listen mitgeschickt werden, wer die Anfrage bereits bearbeitet hat. Diese Peers erhalten die Anfrage dann kein zweites Mal.

Heuristische Ansätze

Einige Algorithmen setzen auf Heuristiken, um möglichst nur Peers zu fragen, die mit hoher Wahrscheinlichkeit eine positive Antwort liefern.

Iterative Deepening Hierbei wird die Tiefe der Anfrage (TTL) zuerst niedrig angesetzt und nur erhöht, wenn kein Ergebnis geliefert wurde. Der erste Peer sendet die Anfrage z.B. mit einem TTL von 2. Erhält er kein Ergebnis, so wird die Anfrage mit einem höheren TTL, etwa 4, wiederholt. Die Knoten der Tiefe bis 2 leiten die Anfrage nur weiter, da sie sie bereits ausgeführt haben. Neu ausgeführt werden sie ab den Knoten der Tiefe 3. Dieses Vorgehen spart Rechenzeit und Bandbreitenaufwand, wenn die gesuchte Ressource nah bei dem anfragenden Peer liegt [VLO10] .

Interessen-basierte Verbindungen Dieser Methode liegt die Annahme zu Grunde, dass Peers, die eine Anfrage eines anderen Peers in der Vergangenheit beantworten konnten, auch in Zukunft mit höherer Wahrscheinlichkeit Anfragen beantworten können. Die beiden Peers haben dann ähnliche Interessen, sodass es sinnvoll ist, solche Peers in die eigene Nachbarschaft aufzunehmen [VLO10].

Random Walk Bei diesem Ansatz wird mit Hilfe des Zufalls nach der Ressource gesucht. Der Startknoten sucht sich einen Peer aus seiner Nachbarschaft und sendet diesem die Anfrage. Dieser führt sie aus und sendet sie an genau einen zufällig gewählten Peer weiter. Die Anfrage, die weitergeleitet wird, wird *Walker* genannt. In der Standardvariante gibt es genau einen Walker. Es sind einige Abwandlungen möglich: Anstatt die Anfrage zu Beginn nur an einen zufälligen Knoten zu senden, kann sie an k zufällige Peers gesendet werden. Diese senden die Anfrage dann jeweils wieder nur an einen Peer weiter, es gibt also k Walker. Der Vorteil dieser Methode besteht darin, dass die Anzahl an Nachrichten sowie den besuchten Peers linear wächst anstatt exponentiell [VLO10].

2.10.3.2 Routing in strukturierten Netzwerken

In unstrukturierten P2P-Netzwerken kann es sehr aufwendig sein, eine Ressource zu finden. Außerdem kann nicht zugesichert werden, dass eine Ressource, die vorhanden ist, auch tatsächlich gefunden wird. Um diese Nachteile auszubessern, wurden strukturierte P2P-Systeme entwickelt. Sie eignen sich vor allem, wenn nur selten neue Peers hinzukommen bzw. Peers das Netzwerk verlassen, denn die Neuordnung des Netzwerkes ist aufwendig. Dafür kann die Suche stark beschleunigt werden. Viele Verfahren geben eine feste obere Schranke für den Suchaufwand an, häufig ist dieser

sogar bei $O(logN)$, wobei N die Anzahl der Teilnehmer ist. Die vorhandenen Netzwerke, die auf ein strukturiertes Overlay aufbauen, verwenden Distributed Hashtables (DHT), bilden eine Baumstruktur oder nutzen Sprunglisten (Skip lists) [VLO10]. Verteilte Hashtabellen werden im Folgenden genauer beschrieben.

Distributed Hashtables

Bei diesem Verfahren ist jeder Peer für einen festen Bereich der Werte einer Hashfunktion zuständig. Jeder Datei wird ein eindeutiger Hash-Wert zugewiesen, z.B. über den SHA-1-Algorithmus[15]. Dann wird die Datei bei dem Peer gespeichert, der für den Wertebereich dieser Datei zuständig ist. Sucht ein Peer eine Datei, nutzt er die Hash-Funktion und kann so ermitteln, welcher Peer für die Datei zuständig ist. Die Suche nach einer Datei kann also sofort mit einem exakten Ergebnis abgeschlossen werden, indem der entsprechende Peer nach der Ressource gefragt wird. Die genutzte Hash-Funktion sollte möglichst uniform sein. Dann werden die Dateien gleichmäßig über die Peers verteilt und die Performance des Netzwerkes steigt [VLO10]. Ein Nachteil an diesem Verfahren ist, dass Anfragen, die keine spezielle Ressource suchen, sondern einen ganzen Bereich (range queries), nicht funktionieren. Hash-Werte, die durch eine uniforme Hash-Funktion entstanden sind, verlieren die Information zur Ordnung der Daten.

Ein Beispiel für ein dezentrales Netzwerk mit strukturiertem Overlay ist Chord[16]. Dies nutzt verteilte Hashtabellen und ein spezielles Ring-Overlay-Netzwerk. Jeder Peer ist für einen bestimmten Bereich der Hash-Werte zuständig. Durch Querverbindungen innerhalb der Ring-Struktur ist eine besonders schnelle Suche möglich.

2.10.4 Verteiltes Rechnen

Die bisher beschriebenen Architekturen und Algorithmen sind für P2P-Systeme im allgemeinen wichtig. In diesem Abschnitt wird nun ein Teilbereich von P2P, das verteilte Rechnen, etwas genauer betrachtet. Die bekannteste Anwendung von verteiltem Rechnen in einem P2P-Netzwerk ist vermutlich SETI@home. Das Projekt beschäftigt sich damit, in Radiosignalen aus dem All nach Anzeichen von außerirdischer Intelligenz zu suchen. Das System ist dabei jedoch nur teilweise ein P2P-System und erinnert stark an eine Client-Server-Architektur. Ein zentraler Server verteilt Arbeitspakete an die Peers. Diese lösen die Rechenaufgaben selbst und schicken das Ergebnis zurück an den Server. Untereinander tauschen sich die Peers nicht aus. Bei diesen Art von Arbeitsverteilung wird auch vom *Master-Worker-Paradigma* gesprochen [KGJ08]. Trotzdem wird es meistens zu den P2P-Systemen gerechnet [VLO10] [GS02]. Dieses Vorgehen, das auch in anderen Systemen zum Einsatz kommt, funktioniert nur unter gewissen Voraussetzungen, die mögliche Einsatzgebiete eingrenzen:

Es sollte den Peers möglich sein, die Aufgaben selbst zu lösen, ohne viel mit anderen Peers kommunizieren zu müssen. Zu viel Kommunikation würde aufgrund der meistens geringen Bandbreite der Peers zu starken Geschwindigkeitseinbußen führen. Außerdem arbeiten solche Systeme meistens nach dem *Single Process Multiple Data-Modell*. Dabei wird auf allen Peers dasselbe Programm ausgeführt. Nur die Daten, mit denen es ausgeführt wird, unterscheiden sich. Eine weitere Voraussetzung ist eine, je nach Netzwerkbandbreite, begrenzte Größe der Arbeitsaufgaben, die vom Server zu den

[15] `http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf`, Zuletzt abgerufen am 23.04.2014

[16] `https://github.com/sit/dht/wiki`, Zuletzt abgerufen am 27.04.2014

Peers geschickt werden sowie der Antworten, die die Peers zurücksenden. Zu den Anfangszeiten von SETI@home war die Internetverbindung des Servers ein Flaschenhals. Die 30 MB/s reichten nicht aus, um allen Nutzern die 350 KB großen Arbeitspakete zuzuschicken [GS02].

2.10.4.1 Distributed Data Mining

P2P-Netzwerke für verteiltes Rechnen können auch dazu genutzt werden, Data Mining durchzuführen, was häufig „Distributed Data Mining“ genannt wird. Die Verwendung von herkömmlichen Data-Mining-Algorithmen in verteilten Netzwerken ist jedoch nicht ohne Weiteres möglich. Trotzdem gibt es einige Algorithmen, die in P2P-Netzen exakte Ergebnisse erzielen können. Des Weiteren wurden Algorithmen entwickelt, die die Ergebnisse annähern [DBG+06]. Zu letzteren zählt eine Entwicklung aus [DBG+06]. Diese versucht, Datenelemente zu Clustern und nutzt dazu eine verteilte Variante des verbreiteten K-Means Algorithmus. Dabei berechnen die Peers durch Kommunikation mit den Nachbarn die Mittelpunkte der Cluster. Die Genauigkeit der Ergebnisse liegt bei 90%, wenn ein zentral ausgeführter K-Means-Algorithmus als Vergleichsbasis genutzt wird.

Einen Schritt weiter gehen verteilte Netzwerke, die Data Mining auf Datenströmen durch-führen. Dies wird vor allem bei Sensornetzwerken genutzt, z.B. mit verteilten Temperatursensoren. Sie verbinden sich untereinander, leiten ihre Sensordaten weiter und warnen z.B., wenn an einigen Sensoren Durchschnittswerte überschritten werden. Je nach Sensortyp können die Sensoren selbst Teil des verarbeitenden Netzwerkes sein. Alternativ werden Rechner, die von einigen Sensoren die Daten erhalten, zu einem P2P-Netzwerk zusammengeschlossen. Das Netzwerk könnte z.B. eine Warnung ausgeben, wenn die Durchschnittstemperatur einen gewissen Schwellwert überschreitet [PPKG03].

2.10.4.2 Ziele von P2P-Algorithmen

Bei der Entwicklung von verteilten Data-Mining-Algorithmen gibt es einige Aspekte, die besonders beachtet werden müssen. Dazu zählt vor allem die Skalierbarkeit. Auch bei vielen teilnehmenden Peers oder Sensoren soll das Netzwerk noch die gewünschten Berechnungen ausführen können. Die Algorithmen sollten also unabhängig von der Größe des Netzwerkes funktionieren oder zumindest nur vom Logarithmus der Größe abhängig sein. Das wird vor allem durch *lokale Algorithmen* erreicht. Diese berechnen das Ergebnis nur mit den Daten, die sie selbst haben und mit denen ihrer Nachbarn. Es ist nicht nötig, die Daten aller Peers oder eines bestimmten Anteils zu erfragen oder gar die Daten aller Peers zu synchronisieren. Die Peers arbeiten also asynchron [DBG+06].

Gerade bei Netzwerken, die Datenströme z.B. von Sensoren auswerten, ist die Berechnung zeitkritisch. Muss der Algorithmus die Berechnung vollständig neu starten, wenn sich die Daten ändern, ist er für die Verwendung in diesem Kontext nicht geeignet. Außerdem sollte der Algorithmus in der Lage sein, zu jeder Zeit ein (approximatives) Ergebnis zu liefern. Diese Algorithmen werden *anytime algorithms* genannt [DBG+06]. Die Algorithmen sollten zudem auf einen zentralen Server verzichten und damit umgehen können, wenn Peers das Netzwerk unerwartet verlassen [DBG+06].

2.10.5 Zusammenfassung

Zur Begegnung der Herausforderungen, die sich durch steigende Datenmengen in vielen Bereichen der Informationstechnik ergeben, ist P2P ein geeignetes Mittel. Durch die Verteilung der Last auf viele Teilnehmer eines Netzwerkes können umfangreiche Aufgaben gelöst werden. Jedoch bringen P2P-

Systeme auch selbst neue Herausforderungen mit, deren Lösungsansätze in dieser Arbeit beschrieben werden.

Zunächst wird auf die Merkmale wie der sehr dynamische Aufbau und die Skalierbarkeit von P2P-Systemen eingegangen. Außerdem werden die unterschiedlichen Architekturtypen thematisiert. In dezentralen Systemen gibt es keine zentralen Komponenten, was solche Netzwerke besonders ausfallsicher und skalierbar macht. Netzwerke mit zentralen Komponenten haben dafür den Vorteil, dass sie leichter verwaltet werden können und die Suche nach Ressourcen deutlich schneller möglich ist. In hybriden Systemen wird versucht, durch einige *Super-Peers* die Vorteile beider Architekturvarianten zu verbinden. Anschließend werden einige Algorithmen zum Finden von Ressourcen im P2P-Netzwerk vorgestellt und hinsichtlich ihrer Vor- und Nachteile analysiert. Vor allem in unstrukturierten Netzwerken ist es schwierig, Ergebnisse zu liefern, ohne dabei das Netzwerk durch zu viele Anfragen zu überlasten. Mit speziellem Blick auf das verteile Analysieren von Daten werden die Grundlagen von Algorithmen für die Verwendung in P2P-Netzwerken beschrieben.

2.11 JXTA

JXTA ist eine plattformunabhängige Open Source Protokollsammlung für P2P-Netzwerke. Das JXTA Projekt wurde 2001 von Sun Microsystems ins Leben gerufen [jxt14]. Der Name JXTA leitet sich aus dem englischen *juxtapose* ab, was *nebeneinanderstellen* bedeutet und darauf abzielt, dass JXTA nicht die Client/Server-Architektur ablösen, sondern als Alternative nebenher existieren soll [GOT02]. JXTA ist somit keine Abkürzung im herkömmlichen Sinne. Das Ziel von JXTA ist dabei P2P-Programmierung auf möglichst vielen Endgeräten zu unterstützen – sei es auf einem Desktop PC oder auf Mobiltelefonen [jxt14]. Zudem ist JXTA auch netzwerkunabhängig, da JXTA nicht nur TCP/IP, sondern auch HTTP, Bluetooth und weitere Netzwerktypen unterstützt [GOT02].

Die JXTA-Protokolle helfen Entwicklern von P2P-Anwendungen grundlegende Eigenschaften von P2P-Netzen in ihren Anwendungen abzubilden. Dazu zählt zum Beispiel das Anbieten von Diensten auf mehreren verteilten Systemen. Auch treten Peers spontan in ein Netz ein und verlassen es auch wieder und müssen andere Peers und Inhalte finden können. Dabei fungieren Peers nicht immer nur als Konsument des Dienstes, sondern bieten ihn auch gleichzeitig an. Zudem werden häufig unterschiedliche Systeme an unterschiedlichen Orten verwendet [GOT02].

Es existieren verschiedene Bibliotheken, die die JXTA Protokolle als API bereitstellen – zum Beispiel JXSE[17] für Java-Applikationen oder JXTA-C[18] als C/C++/C# Implementierung.

In diesem Teilkapitel wird ein Überblick über die Grundbegriffe und Technologien von JXTA gegeben. Dazu werden zunächst die Konzepte von JXTA erklärt und die Architektur veranschaulicht. Zum Schluss werden Kern- und Standard-Protokolle vorgestellt und deren Funktionsweise beschrieben.

2.11.1 Konzepte

In diesem Abschnitt werden grundlegende Begriffe von JXTA erklärt. Dabei wird auf die verschiedenen Ressourcen (siehe Abbildung 2.29), die Bestandteil eines JXTA-Netzwerks sind, eingegangen.

[17] https://jxse.kenai.com/
[18] https://java.net/projects/jxta-c

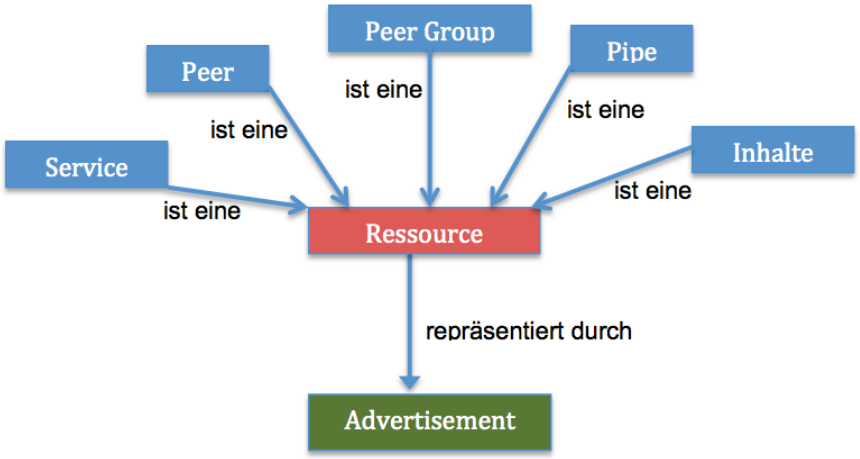


Abbildung 2.29: Ressourcen im JXTA-Netzwerk [Ver10]

2.11.1.1 Peer

Ein Peer ist ein Kommunikationspunkt, welcher JXTA unterstützt [Ver10]. Dieser ist jedoch nicht mit einem Benutzer gleichzusetzen, da ein Benutzer mehrere Peers auch auf mehreren Endgeräten betreiben kann – z.B. auf seinem Smartphone und gleichzeitig auf seinem Desktop-PC [BGKS02]. Der Peer kann unterschiedliche Rollen im JXTA-Netzwerk einnehmen:

Minimal Peer Der Minimal Peer bietet lediglich das Senden und Empfangen von Nachrichten an [Sun07a].

Simple Peer Der Simple Peer (oder auch Full-Featured Peer) kann zusätzlich Daten in einem lokalen Speicher vorhalten – z.B. für das Routing [Wer03].

Rendezvous Peer Die Aufgabe des Rendezvous Peer ist es, das Auffinden von Ressourcen den anderen Peers zu erleichtern. Er legt sich Listen mit Advertisements (siehe Abschnitt 2.11.1.6) an um so Suchanfragen anderer Peers beantworten zu können [Sun07a].

Relay Peer Der Relay Peer ermöglicht es durch z.B. Firewalls, Router und Proxies geschützte Simple Peers zugänglich zu machen. Er befindet sich i.d.R. am Rand eines Netzwerks und leitet Anfragen an Peers weiter, die sich hinter diesen Netzbarrieren befinden[Ver10, BLMM04].

2.11.1.2 Peer Group

Eine *Peer Group* ist eine Menge von Peers, die die gleichen Interessen verfolgen und sich selbst in einer Gruppe organisieren [Ver10]. Diese Gruppen können hierarchisch angeordnet werden [Ver10] und zudem public oder private – also mit einem Passwortschutz versehen – sein [Gra02]. Jede Gruppe stellt dabei Services bereit, die die Peers nutzen können [Gra02]. Folgende Peer Groups sind in einem JXTA-Netzwerk vorhanden:

- World Peer Group

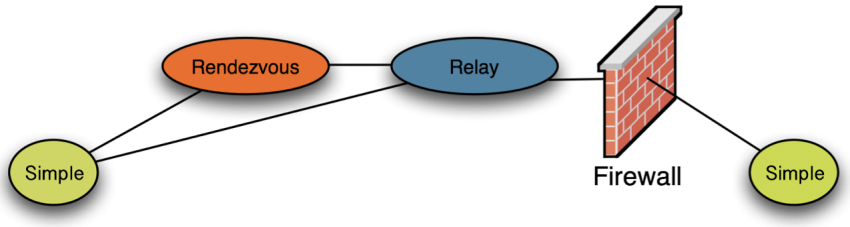


Abbildung 2.30: Arten von Peers in JXTA [Sun07a]

- User Peer Groups

Die *World Peer Group* (oder auch *Net Peer Group*) ist eine fest definierte übergeordnete Gruppe, in die jeder Peer automatisch eintritt. Dadurch stehen dem Peer spezielle Services zur Verfügung, die im weiteren Verlauf dieses Abschnitts erklärt werden (Discovery, Advertisments, Pipes) [Gra02]. Entwickler können zudem eigene Peer Groups – sogenannte *User Peer Groups* – erstellen, die spezielle Services in ihren Anwendungen bereitstellen. Die User Peer Groups sind der World Peer Group untergeordnet [GOT02].

2.11.1.3 Endpoint

Um einen Peer im Netzwerk zu erreichen, besitzt es einen oder mehrere Endpoints. Ein *Endpoint* ist die Methode zur Adressierung eines Peers. Beispielsweise kann ein Endpoint eine IP-Adresse sein. Da JXTA jedoch unabhängig von Transport-Protokollen arbeitet, kann ein Peer auch über Bluetooth oder andere Übertragungsprotokolle an das Netzwerk angebunden sein. Endpoints abstrahieren also das Übertragungsprotokoll, sodass auf einfachere Art und Weise ein Peer erreicht werden kann – u.U. auch über mehrere Peers hinweg [BGKS02].

2.11.1.4 Pipes

Damit Peers untereinander Informationen austauschen können, existieren sogenannte *Pipes*. Sie verbinden zwei Endpoints miteinander und sind die Kommunikationskanäle zwischen den Peers. Ein Peer kann über *Input Pipes*, die Daten empfangen, oder über *Output Pipes*, die Daten senden, verfügen. Pipes besitzen zwei Kommunikationsmodi:

Point-to-point Pipes Point-to-point Pipes verbinden genau zwei Endpoints miteinander, wobei ein Peer eine Input Pipe und der andere Peer über eine Output Pipe verfügt [Sun07a].

Propagate Pipes Eine Propagate Pipe verbindet eine Output Pipe mit mehreren Input Pipes. Die Nachrichten fließen dann von der Output Pipe an die verschiedenen Input Pipes [Sun07a].

JXTA unterstützt eine Reihe von unterschiedlichen Pipe-Typen. Eine einfache Pipe ist zunächst einmal unidirektional. Um eine Kommunikation in beide Richtungen zu gewährleisten, müssen zwei Pipes kombiniert werden. JXTA bietet jedoch auch eine einfache Möglichkeit eine bidirektionale Pipe zu erzeugen. Um Video, Audio oder auch Sensordaten zu übertragen, existieren spezielle Streaming Pipes, die Datenströme effizient übertragen können [BGKS02].

2.11.1.5 Services

Ein Peer oder eine Peer Group stellt verschiedene Services bereit. Die JXTA-Platform beinhaltet sieben grundlegende Services. Außer dem *Endpoint Service* und dem *Resolver Service* müssen nicht alle Services verwendet und implementiert werden, in der Regel stellt eine Peer Group jedoch zumindest den Membership Service bereit, damit Peers der Gruppe beitreten können [GOT02]:

Endpoint Service Der Entpoint Service wird verwendet um zwischen den Peers Nachrichten senden und empfangen zu können. Dieser Service implementiert gewöhnlich das Endpoint Routing Protocol (siehe Abschnitt 2.11.3.5) und muss von einem Peer angeboten werden.

Resolver Service Der Resolver Service wird benutzt um generische Suchanfragen an andere Peers zu senden. Z.B. wird der Service verwendet um Statusinformationen eines Peers abzufragen. Auch dieser Service muss von einem Peer angeboten werden.

Discovery Service Der Discovery Service dient dazu, dass veröffentlichte Ressourcen im Netzwerk gefunden werden können. Der Dienst ist ein Standard-Discovery-Service in jeder Peer Group und kann auch durch alternative Discovery Services ausgetauscht werden.

Membership Service Der Membership Service wird benutzt um Peers eine Identität zu geben, die von der Anwendung genutzt werden kann um festzustellen, wer eine Anfrage sendet und ob er die benötigten Rechte dafür besitzt.

Access Service Der Access Service wird benutzt um festzustellen, ob ein Peer die benötigten Rechte besitzt auf eine bestimmte Ressource zuzugreifen. Dazu werden sowohl die Art der Anfrage als auch die Anmeldedaten des Peers überprüft.

Pipe Service Der Pipe Service wird benutzt um die Pipes zwischen den Gruppenmitgliedern zu erstellen und zu verwalten.

Monitoring Service Der Monitoring Service erlaubt es einem Peer andere Peers der Gruppe zu überwachen (Monitoring) [Sun07a].

2.11.1.6 Advertisement

Advertisements sind XML-Dokumente, die die Ressourcen (Pipes, Peers, Peer Groups, Services) eines JXTA-Netzwerks beschreiben. Sie helfen den Peers die gewünschte Ressource im Netzwerk zu finden, indem sie bei Erstellung der Ressource im Netzwerk verteilt und somit bekannt gemacht werden. JXTA definiert verschiedene Arten von Advertisements:

- Peer Advertisement
- Peer Group Advertisement
- Pipe Advertisement
- Service Advertisement
- Content Advertisement

- Endpoint Advertisement

Die Advertisements enthalten einen Namen und die ID der entsprechenden Ressource, sodass über das Advertisement die Ressource gefunden und identifiziert werden kann. Wenn z.B. eine Peer Group erzeugt wird, wird das Peer Group Advertisment an alle Peers im Netz gesendet. Dabei erreicht es auch die Rendezvous Peers, die das Advertisement speichern und somit das Suchen und Auffinden ermöglichen. Möchte nun ein Peer der Gruppe beitreten, sucht er nach dessen Namen und erhält über den Rendezvous Peer die ID der Peer Group zurück, sodass er dieser beitreten und dessen Services nutzen kann [BGKS02, Gra02, GOT02]. Um keine Ressourcen anzubieten, die veraltet sind und nicht mehr existieren, besitzen Advertisements ein Verfallsdatum. Die Advertisements werden dann nach Ablauf dieser Zeit gelöscht. Um eine Ressource weiterhin im Netzwerk zu veröffentlichen, muss sie erneut veröffentlicht werden. Dies kann auch vor Ablauf dieser Frist geschehen [Sun07a].

2.11.1.7 Discovery

Wenn ein Peer eine Ressource beziehen möchte – zum Beispiel eine Textdatei – muss er den Peer, der diese Daten bereitstellt zunächst finden. Dieser Prozess wird als *Discovery* bezeichnet. Es wird zwischen der *Dynamic Discovery* und der *static Discovery* unterschieden. Bei der *Dynamic Discovery* wird das Peer Discovery Protocol (siehe Abschnitt 2.11.3.1) verwendet und durch Rundnachrichten (multicast) mehrere Peers angefragt, ob sie die gewünschte Ressource besitzen. Die Peers antworten dann direkt an den anfragenden Peer. Bei der static Discovery fragt der Peer bei den Rendezvous Peers (siehe Abschnitt 2.11.1.1) nach. Sie halten eine Liste mit ihnen bekannten Ressourcen vor und können dem anfragenden Peer eine Antwort mit der Ressource liefern [GOT02]. Dieses Vorgehen ist besonders von Vorteil, wenn sich der anbietende Peer außerhalb des lokalen Netzwerks befindet, da die Rundnachrichten der *Dynamic Discovery* nur innerhalb eines Netzwerks funktionieren [Gra02].

2.11.1.8 Module

Wenn zu einer Peer Group zusätzliche Funktionalitäten hinzugefügt werden sollen, die nicht von JXTA abgedeckt werden, werden diese durch *Module* beschrieben. Ein Modul enthält eine Beschreibung über diese Funktionalität und wie diese von Peers zu nutzen ist. Sie wird dann im P2P-Netz mittels Advertisements verbreitet, sodass andere Peers diese Funktion finden und bei Bedarf nutzen können [Gra02].

2.11.2 Architektur

Die Architektur von JXTA besteht aus drei Schichten, die aufeinander aufbauen (siehe Abbildung 2.31). Der *JXTA Core* bildet dabei mit Peer Groups und Pipes die Basis. Er stellt grundlegende Services wie das Erstellen von Peer Groups oder das Verbinden von Pipes bereit [Gra02].

Die zweite Schicht, die *Services Layer*, baut darauf auf und erlaubt komplexere Services wie Indexierung und Suchen. Zudem stellt die Services Layer einen einfachen Dateitransfer zwischen Peers bereit. Die bereitgestellten Services werden dabei sowohl von der Community als auch von Sun entwickelt [BLMM04].

Die *Application Layer* ist die dritte Schicht und ist die Schnittstelle zu einer konkreten P2P-Applikation. Zwischen der Application- und Services-Schicht befindet sich noch die JXTA-Shell. Sie

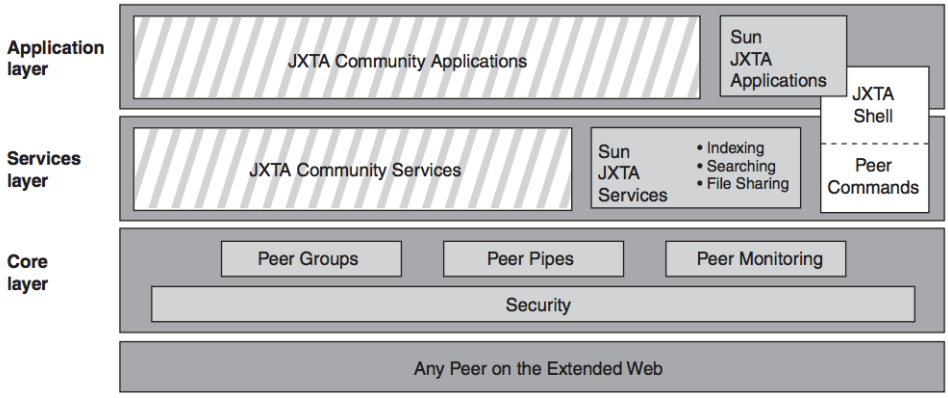


Abbildung 2.31: JXTA P2P Architektur [Gra02]

ist eine Konsolen-Anwendung, mit der sich die JXTA-Services nutzen lassen. So kann beispielsweise über die Shell Peer Groups oder Pipes erzeugt und die Funktionsweise getestet werden [Gra02].

2.11.3 Protokolle

JXTA definiert mehrere Protokolle in Form von XML-Nachrichten, die die Kommunikation zwischen den Peers gewährleisten. Die Peers verwenden diese Protokolle um sich gegenseitig zu finden, Advertisments zu verteilen, Ressourcen zu finden und zu kommunizieren. Die Protokolle werden in den Peers als Service angeboten. Im Folgenden Abschnitt werden sechs Standard-Protokolle vorgestellt, die von den Peers implementiert werden können. Es müssen allerdings keine Protokolle implementiert werden, die nicht verwendet werden. Jedes dieser Protokolle basiert auf dem Query/Response Model. Dabei werden Anfrage-Nachrichten asynchron versendet, die mit Response-Nachrichten erwidert werden [Sun07a].

2.11.3.1 Peer Discovery Protocol

Das *Peer Discovery Protocol (PDP)* wird von den Peers verwendet um ihre eigenen Ressourcen anzubieten oder Ressourcen von anderen Peers zu finden. Dazu wird jede Ressource durch ein Advertisement beschrieben und veröffentlicht. Möchte ein Peer eine Ressource finden, erzeugt er *Discovery Query Message*, die in der Peer Group verteilt wird. Wenn ein oder mehrere Peers die gesuchte Ressource kennen, erhält der suchende Peer eine Antwort in Form einer *Discovery Response Message*. Dieses Protokoll findet bei der dynamic Discovery (siehe Abschnitt 2.11.1.7) Anwendung [Sun07a].

2.11.3.2 Peer Information Protocol

Das *Peer Information Protocol (PIP)* kann von Peers genutzt werden um Status-Informationen wie Uptime, Status, Auslastung etc. von anderen Peers zu erfragen. Das Peer Information Protocol stellt einige Nachrichten bereit um den Peer Status zu erfragen. So gibt es z.B. eine *ping message*, mit der geprüft werden kann, ob der Peer noch existiert oder nicht. Als Antwort auf die ping message wird eine *peer info message* erzeugt, die die Peer ID, Ziel ID, Uptime und Advertisement des Peers enthält [Sun07a].

2.11.3.3 Peer Resolver Protocol

Das *Peer Resolver Protocol (PRP)* erlaubt dem Peer eine generische Anfrage an einen oder mehrere Peers zu stellen. Die Anfragen können an alle Peers in der Gruppe gerichtet sein oder an bestimmte Peers innerhalb der Gruppe. Im Gegensatz zu PDP und PIP, die auf dem PRP aufbauen, kann dieses Protokoll von Peers genutzt werden, um beliebige Informationen auszutauschen, die sie benötigen – also nicht nur vordefinierten Informationen.

Zur Verbreitung der Anfragen (Resolver Query Message) im Netzwerk nutzt PRP den Rendezvous Service. In der Nachricht ist ein Handler-Name enthalten. Jeder Service in der Peer Group kann einen Handler am Resolver Service registrieren. Nur wenn der Service einen passenden Handler registriert hat, kann er auf die Nachricht reagieren und diese beantworten oder weiter verbreiten. [Sun07a, Wer03].

2.11.3.4 Pipe Binding Protocol

Die Kommunikation zwischen Peers erfolgt über Pipes. Um die Verbindung über Pipes herzustellen, wird das *Pipe Binding Protocol (PBP)* genutzt. Ein Peer erstellt zunächst eine Input Pipe und nutzt Advertisments um die Eigenschaften der Pipe zu beschreiben und anderen Peers bekannt zu machen. Möchte ein anderer Peer eine Verbindung aufbauen, sendet er eine Anfrage (Pipe Binding Query Message) ab. Daraufhin erhält der anfragende Peer das passende Advertisement zurück. Nun sendet der anfragende Peer an den Peer, zu dem er eine Pipe aufbauen möchte, eine *Pipe Resolver Message*. Schließlich kann mit dem PBP die passende Output Pipe erzeugt werden [Sun07a, Gra02].

2.11.3.5 Endpoint Routing Protocol

Das *Endpoint Routing Protocol (ERP)* erlaubt einem Peers das Senden von Nachrichten an einen anderen Peer, ohne dass eine direkte Verbindung zwischen den Peers besteht. Dazu wird die Nachricht über zwischengeschaltete Peers übertragen, um so den Ziel-Peer zu erreichen. Das Protokoll definiert ein Anfrage- und Antwort-Protokoll um Routing Informationen zu erhalten – also über welchen Weg die Nachricht zum Ziel-Peer gelangt. Zudem kapselt das Protokoll die Nachrichten in eigene Nachrichten, die die Routing Informationen enthalten um die Nachricht so über mehrere Peers hinweg zu senden.

Wenn ein Peer eine Nachricht senden möchte, prüft er zunächst im eigenen Cache, ob er eine Route zum Ziel-Peer bereits kennt. Ist dies nicht der Fall, sendet er einen *Route Resolver Query Request* an die Peers in seine Gruppe. Darin fragt er die anderen Peers nach einer Route zu seinem gewünschten Ziel-Peer. Jeder Peer, der diese Request erhält, prüft, ob er er eine Route zu dem gewünschten Peer kennt und antwortet dem anfragenden Peer ggf. mit den Routing Informationen. Diese Antwort enthält unter anderem eine Liste mit Peer IDs, die sich auf dem Pfad zum Ziel-Peer befinden. Mit dieser Liste kann der Peer nun eine Nachricht über diesen Pfad an den Ziel-Peer senden [Sun07a]

2.11.3.6 Rendezvous Protocol

Das Rendezvous Protocol (RvP) wird genutzt, um Nachrichten innerhalb einer Peer Group zu verteilen. Es nutzt dabei Techniken um die Nachrichten kontrolliert und effizient zu verteilen.

Das RvP teilt sich in drei Unterprotokolle auf:

PeerView Protocol Das PeerView Protocol ist optional und wird von Rendezvous Peers verwendet um sich selbst zu organisieren. Eine *PeerView* ist eine Liste von Peers, die zur Zeit als Rendezvous Peer arbeiten. Jeder Rendezvous Peer verwaltet diese PeerView lokal und nutzt das PeerView Protocol um diese Liste mit anderen Rendezvous Peers abzugleichen [Sun07b].

Rendezvous Lease Protocol Das Rendezvous Lease Protocol ist optional und erlaubt nicht-Rendezvous Peers die verbreiteten Nachrichten zu empfangen. Peers, die nicht als Rendezvous Peer agieren, können ihr Interesse an Nachrichten bekunden, die eigentlich nur bei den Rendezvous Peers eintreffen. Diese Nachrichten leitet der Rendezvous Peer an die abonnierenden Peers weiter [Sun07b].

Rendezvous Propagation Protocol Das Rendezvous Propagation Protocol ist das einzige Protokoll, das für Peers, die den Rendezvous Service nutzen, nicht optional ist. Das Protokoll erlaubt Peers individuelle Nachrichten innerhalb der Peer Group zu verbreiten und zu verwalten. So kann u.a. die Herkunft der Nachricht und die Peers, über die die Nachricht bisher den Weg genommen hat, von den Peers bestimmt werden.

2.11.4 Zusammenfassung

JXTA ist eine plattform- und netzwerkunabhängige Protokollsammlung. Es existieren verschiedene Bibliotheken, die die Protokolle für Entwickler bereitstellen – z.B. JXSE für Java-Applikationen.

In einem JXTA-Netzwerk existieren u.a. Peers, Peer Groups, Pipes, Services und Inhalte, die als Ressourcen bezeichnet werden. Jede Ressource wird mit einem speziellen Advertisement im Netzwerk verbreitet und somit anderen Peers bekannt gemacht.

Rendezvous Peers speichern diese Advertisements und können Suchanfragen anderer Peers zu einer Ressource beantworten. Relay Peers sorgen dafür, dass Peers hinter Netzwerkbarrieren wie Routern und Firewalls erreicht werden können.

Peers sind in Gruppen organisiert, die verschiedene Services anbieten. JXTA definiert bereits Standard-Services damit sich z.B. Peers finden oder Gruppen verwalten können. Peers müssen allerdings nicht zwangsläufig alle Services implementieren und können auch anwendungsspezifische Services bereitstellen. Protokolle definieren die Austauschformate um die Funktionalität der Services zu gewährleisten.

2.12 Android

Android ist ein Betriebssystem für mobile Endgeräte, das erstmals im November 2007 von Google vorgestellt wurde. Da der Quellcode frei zugänglich ist, kann es von Herstellern für Smartphones und Tablets kostenfrei eingesetzt und modifiziert werden. Der freie Zugang führte dazu, dass im vierten Quartal 2014 Android einen weltweiten Marktanteil von 76,6% besitzt und gegenüber iOS[19] und anderen mobilen Betriebssystemen eine wesentlich höhere Verbreitung hat [20]. Entwickler können für Android zusätzliche Programme (sog. Apps) in der Programmiersprache Java entwickeln. Dazu wird

[19] https://www.apple.com/de/ios/what-is/
[20] http://www.idc.com/prodserv/smartphone-os-market-share.jsp

ein Software Development Kit (SDK) angeboten, das bereits grundlegende Funktionen und Schnittstellen zu Ressourcen bereitstellt [Kün12].

In den folgenden Abschnitten werden grundlegende Konzepte der Android Programmierung beschrieben. Zunächst werden Ressourcen sowie die grafische Oberflächengestaltung mit Layouts & Views erläutert. Anschließend wird auf Activities und Fragments eingegangen.

2.12.1 Ressourcen

Android Projekte besitzen eine bestimmte Struktur, durch die die Programmlogik von der Oberflächengestaltung und sonstigen Ressourcen getrennt wird. Zu den Ressourcen zählen unter anderem

- layouts (beschreiben Views und Aussehen der App),
- strings (sprachenabhängige Texte) und
- drawables (Grafiken klassifiziert nach Auflösung).

Während sich der Java Programmcode für gewöhnlich in einem gleichnamigen Ordner befindet, sind die Ressourcen[21] einzelne XML-Dateien, die in speziellen Unterordnern des *res* Verzeichnisses abgelegt sind. Werden neue Ressourcen angelegt oder bestehende verändert, wird automatisch eine sogenannte R-Klasse generiert, die Identifier jeder einzelne Ressource beinhaltet. Mithilfe dieser Klasse kann auch aus der Java-Programmlogik heraus auf die Ressourcen referenziert werden [Kün12].

2.12.2 Layouts & Views

Die Oberflächenbeschreibung findet in einzelnen Layout-Dateien[22] statt. Darin wird zunächst eine grundlegende ViewGroup definiert, die weitere ViewGroups oder Views enthalten kann (Listing 2.6 zeigt ein Beispiel eines solchen Layouts).

Je nach gewählter ViewGroup werden einzelne Views darin unterschiedlich angeordnet. Die geläufigsten ViewGroups sind LinearLayout und RelativeLayout. Die in einem LinearLayout befindlichen Views werden entweder horizontal nebeneinander oder vertikal untereinander angeordnet. In einem RelativeLayout werden Views relativ zueinander angeordnet, indem jede View ihre Nachbarn referenziert. Views sind bestimmte Elemente wie Texte, Eingabefelder oder Schaltflächen. Sie können über die R-Klasse im Java-Code referenziert werden, sodass z.B. Eingabewerte ausgelesen werden können [Kün12].

2.12.3 Activities

Die eigentliche Programmlogik befindet sich in sog. Activities[23], die in Java geschrieben werden. Jede Activity repräsentiert bestimmte Aufgabenteile einer App, sodass eine App aus mindestens einer aber auch – je nach Funktionsumfang – aus mehreren Activities bestehen kann. Jede konkrete Activity

[21] `http://developer.android.com/guide/topics/resources/providing-resources.html`
[22] `http://developer.android.com/guide/topics/ui/declaring-layout.html`
[23] `http://developer.android.com/guide/components/activities.html`

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/
       ↪ android"
3                android:layout_width="match_parent"
4                android:layout_height="match_parent"
5                android:orientation="vertical" >
6      <TextView android:id="@+id/text"
7                android:layout_width="wrap_content"
8                android:layout_height="wrap_content"
9                android:text="Hello, I am a TextView" />
10     <Button android:id="@+id/button"
11             android:layout_width="wrap_content"
12             android:layout_height="wrap_content"
13             android:text="Hello, I am a Button" />
14 </LinearLayout>
```

Listing 2.6: Beispiel eines LinearLayouts und zwei Views

wird von der Activity-Klasse abgeleitet und besitzt ein eigenes Layout. Untereinander können die Activities andere Activities aufrufen, sodass durch die Funktionen der App navigiert werden kann.

Dabei unterliegen Activities einem bestimmten Lebenszyklus, auf den über verschiedene Callback-Methoden in der Activity reagiert werden kann. Wird eine Acitivity (Activity A) gestartet, werden zunächst die Methoden `onCreate()`, `onStart()` und `onResume()` durchlaufen. In ihnen kann das Layout geladen und initialisiert werden. Wenn nun eine andere Activity (Activity B) gestartet wird, wird die aktuelle verlassen, sodass `onPause()` und `onStop()` aufgerufen werden. Mitunter kann es vorkommen, dass die Activity vom Betriebssystem komplett beendet wird. In dem Fall wird zuletzt `onDestroy()` aufgerufen. Kehrt der Benutzer aus der Activity B zurück in die Activity A, wird je nachdem, ob die Activity bereits vom System beendet wurde, die Activity komplett mit `onCreate()` etc. neu erstellt, oder, wenn sie noch nicht beendet wurde, mit `onResume()` fortgefahren. Für einen kompletten Einblick in den Lebenszyklus von Activities sei auf die Android Dokumentation[24] verwiesen [Kün12].

2.12.4 Fragments

Seit der Android Version 3.0, die erstmals auf Tablets zugeschnitten war, existieren die sog. Fragments[25]. Auf Grund des größeren Bildschirms bieten Tablets mehr Platz für Funktionalitäten. Da Activities jedoch jeweils nur eine Funktionalität widerspiegeln, wird auf den Bildschirmen viel Platz nicht genutzt. Aus diesem Grund wurden Fragments in das Android SDK mit aufgenommen. Sie sind Bausteine innerhalb einer Activitiy und besitzen einen eigenen Lebenszyklus. Eine Activity kann mehrere Fragmente enthalten und somit mehrere Funktionalitäten beherbergen. Jedes Fragment wird dabei von der Fragment-Klasse abgeleitet und besitzt, wie eine Activity auch, ein eigenes Layout. Sie sind somit eigenständig und wiederverwendbar. Um ein Fragment zu instanziieren, gibt es mehrere

[24] http://developer.android.com/guide/components/activities.html
[25] `http://developer.android.com/guide/components/fragments.html`

Möglichkeiten: Zum einen kann ein Fragment in einem Layout über ein Fragment-Element erzeugt werden. Zum anderen kann es auch programmatisch erzeugt und eingebunden werden. Damit Fragments mit ihrer Activity oder anderen Fragments kommunizieren können, stellen sie i.d.R. Interfaces bereit, die von der Activity implementiert werden, so dass die Activity den Datenaustausch kontrolliert [Kün12].

2.13 Scrum

Bei der Softwareentwicklung werden seit langem Vorgehensmodelle verwendet. Diese beschreiben eine festgelegte Reihenfolge von Aktivitäten, welche abgearbeitet werden müssen um Projekte durchzuführen. Hierbei können zwei unterschiedliche Arten von Modellen unterschieden werden. Dies sind zum einen klassische und zum anderen agile Vorgehensweisen [NTM12].

Zu den klassischen Modellen gehören unter anderem das Wasserfall-, das Spiral- und das V-Modell. Bei diesen werden die Projekte in Abschnitte unterteilt, welche sequenziell abgehandelt werden. Tätigkeiten werden dann in diese Abschnitte eingeordnet. Durch die sequenzielle Abarbeitung der einzelnen Abschnitte, steigen die Kosten für etwaig erforderliche Änderungen in späteren Projektphasen exponentiell an. Da dem Kunden erst in späten Projektphasen ein Produkt vorgeführt werden kann, ist das Risiko groß, dass Änderungswünsche sehr teuer werden. Um dieses Risiko zu minimieren, muss meist sehr viel Zeit in eine detaillierte Dokumentation investiert werden [NTM12].

Agile Vorgehensmodelle setzen genau bei diesem Problem an. „Agilität ist die Fähigkeit, auf Veränderungen in einer turbulenten Umwelt, in der sich Unternehmen befinden, zu reagieren“ [Hig02]. Im *Manifest für agile Softwareentwicklung*[26] sind vier Werte festgehalten, die die Agilität auszeichnet:

1. *Individuen und Interaktionen* sind wichtiger als Prozesse und Werkzeuge
2. *Funktionierende Software* ist wichtiger als umfangreiche Dokumentation
3. *Kooperation mit Projektbetroffenen* ist wichtiger als Vertragsverhandlungen
4. *Reaktion auf Änderungen* ist wichtiger als Festhalten an einem starren Plan

Bekannte Vorgehensmodelle, die diese Werte widerspiegeln, sind z.B. *Extreme Programming* und *Scrum* [Bra13b]. Im Folgenden wird sich diese Arbeit auf Scrum konzentrieren und dieses Modell vorstellen.

2.13.1 Scrum

„Scrum ist ein Rahmenwerk, innerhalb dessen Menschen komplexe adaptive Aufgabenstellungen angehen können, und durch das sie in die Lage versetzt werden, produktiv und kreativ Produkte mit dem höchstmöglichen Wert auszuliefern“ [SS13].

Scrum wurde Anfang der 1990er Jahre von Ken Schwaber und Jeff Sutherland entwickelt und basiert auf der Theorie der *Empirie*. Diese besagt, dass unsere Entscheidungen auf Basis unserer Erfahrungen getroffen werden. Zur Optimierung der Prognosesicherheit und Kontrolle von Risiken,

[26] http://agilemanifesto.org/iso/de/

nutzt Scrum einen iterativen, inkrementellen Ansatz. Die drei Säulen *Transparenz*, *Überprüfung* und *Anpassung* bilden dabei das Fundament. Transparenz bedeutet in diesem Zusammenhang, dass alle Ergebnisse und Fortschritte sichtbar festgehalten werden. Um dies sicherzustellen gibt es die Scrum Artefakte. Die anderen beiden Säulen bedeuten, dass durch eine regelmäßige Kontrolle und Anpassung ein möglichst effektiver Projektverlauf sichergestellt werden soll [SS13].

2.13.1.1 Prinzipien

Damit ein Scrum Projekt erfolgreich durchgeführt werden kann, müssen sich die Beteiligten an drei grundlegende Prinzipien halten:

1. Das Pull-Prinzip
2. Timebox und Pünktlichkeit
3. Selbstorganisation

Das *Pull-Prinzip* besagt dabei, dass das Entwicklungsteam autonom bestimmen muss, wie viele Aufgaben es, in der zur Verfügung stehenden Zeit, erledigen kann. Das zweite Prinzip besagt, dass jede Besprechung pünktlich beginnt und vor jedem Ereignis ein fester Zeitraum für dieses definiert werden muss, welcher dann strikt eingehalten wird. Das Entwicklungsteam soll während eines Sprints autonom entscheiden, wie und in welcher Reihenfolge es die Aufgaben abarbeitet. Es besteht somit ein Zwang zur *Selbstorganisation* [Glo13].

2.13.1.2 Rollen

Ein Scrum Team besteht aus dem *Product Owner*, dem *Scrum Master* und dem *Entwicklungsteam*. Im erweiterten Kreis gibt es noch den Kunden, den Manager und die Anwender. Diese gehören nicht dem Scrum Team an. Sie sollen aber an bestimmten Besprechungen trotzdem teilnehmen, um für Rückfragen zur Verfügung zu stehen und Unklarheiten schnell zu beseitigen. Im Folgenden werden die drei primären Rollen näher vorgestellt [SS13].

Product Owner

In Scrum gibt es immer genau einen Product Owner, welcher die Vision vorgibt. Er kann, muss aber nicht unbedingt gleichzeitig der Kunde des Produktes sein. Seine Hauptverantwortung ist dafür zu sorgen, dass der Return on Investment (ROI) maximiert wird. Dies kann je nach Projekt entweder bedeuten, dass der Gewinn oder der Nutzen im Bezug auf die dafür benötigen Kosten maximiert wird. Er definiert die Funktionalitäten für das Produkt und priorisiert diese kontinuierlich neu. Aufgrund seiner Priorisierung werden dann jeweils die Funktionalitäten für den nächsten Sprint ausgewählt [Sut10].

Entwicklungsteam

Das Entwicklungsteam ist selbst-organisierend und interdisziplinär tätig. Das Team besteht aus Profis, die aus verschiedenen Fachbereichen kommen, und muss alle Fähigkeiten zur Herstellung des Produktes besitzen. Während der Entwicklung organisiert es sich selbst und entscheidet selbstständig,

wie es aus dem Product Backlog ein auslieferbares Produkt erstellt. Innerhalb des Entwicklungsteam gibt es keine weitere Untergliederung in Rollen. Das Team als Ganzes ist rechenschaftspflichtig für die umgesetzten Funktionalitäten [SS13].

Die Teamgröße ist ein wichtiger Faktor für die Effektivität des Teams. Bei zu kleinen Teams fehlen oft die benötigten Fähigkeiten um ein fertiges Produkt zu erstellen. Außerdem besteht die Gefahr, dass einzelne größere Funktionen nicht innerhalb eines Sprints fertiggestellt werden können. Die Teams sollten allerdings auch nicht zu groß sein, da mit zunehmender Größe auch die Koordinationsaufwände steigen. Eine optimale Teamgröße liegt zwischen drei und neun Profis, wobei nur Mitglieder zählen, welche Aufgaben aus dem Sprint Backlog erledigen [Glo13].

Scrum Master

Die Hauptaufgabe des Scrum Masters ist es, die Einhaltung und Aufrechterhaltung des Scrum Prozesses sicherzustellen. Weiterhin soll er die Zusammenarbeit innerhalb des Teams optimieren und auch als Kontaktperson für externe Personen fungieren. Damit ein reibungsloser Projektverlauf sichergestellt werden kann, muss er eng mit den beteiligten Personen zusammenarbeiten und verschiedene Dienste für diese erbringen [SS13].

Den Product Owner muss er bei der Erstellung des Product Backlog unterstützen. Dies umfasst sowohl die Vermittlung von Techniken für die Verwaltung als auch die Unterstützung bei der klaren Formulierung und Priorisierung der Einträge. Weiterhin kann er ihn auch bei der Durchführung von Ereignissen unterstützen [SS13].

Das Entwicklungsteam unterstützt er bei der Ein- und Durchführung der Selbstorganisation und bei der funktionsübergreifenden Teamarbeit. Es ist weiterhin seine Aufgabe Hindernisse zu beseitigen, welche das Team bei der Produkterstellung behindern. Auch das Entwicklungsteam kann er bei der Durchführung von Ereignissen unterstützen [SS13].

2.13.1.3 Artefakte

Artefakte dienen in Scrum in erster Linie dazu, Transparenz bezüglich der wesentlichen Informationen und damit Möglichkeiten zur Überprüfung und Anpassung zu schaffen. Die wesentlichen Artefakte sind dabei das *Product Backlog*, das *Sprint Backlog*, das *Impediment Backlog* und das *Product Inkrement* [SS13]. Diese sollen im Folgenden vorgestellt werden.

Product Backlog

Das *Product Backlog* ist eine Sammlung von Anforderungen. Es wird vom Product Owner verwaltet und kontinuierlich aktualisiert und erweitert. Es kann immer nur die bekannten Anforderungen zeigen und ist daher niemals vollständig. Im Laufe des Entwicklungsprozesses und auch noch während des gesamten Produktlebenszyklus kommen stetig neue Anforderungen hinzu. Das Product Backlog existiert somit so lange, wie das dazugehörige Produkt [SS13].

Jeder Eintrag wird dabei als *Product Backlog-Item* bezeichnet. Es kann sowohl ein Feature, eine Funktionalität, eine Verbesserung als auch eine Fehlerbehebung sein. Jeder Eintrag enthält eine Beschreibung, die Reihenfolge, die Schätzung und den Wert. Die Reihenfolge der Einträge gibt gleichzeitig die strategische Bedeutung der Anforderungen an. Die oberen Einträge sind besonders bedeutend und meist auch deutlich detaillierter beschrieben als untere Anforderungen. Während des

Entwicklungsprozesses wird diese Reihenfolge, wie auch alle anderen Details, stetig überarbeitet und verfeinert [SS13].

Um die Aufwände der Product Backlog-Items zu schätzen, kann die Methode des *Planning Poker* verwendet werden. Hierbei handelt es sich um eine relative Schätzung. Der Product Owner kann das Scrum Team hierfür zu einem *Estimation Meeting* einladen. Dabei stellt er die einzelnen Items der Reihe nach vor und lässt diese dann jeweils vom Team schätzen. Da es sich dabei um relative Werte handelt, wird ein Referenzwert des Teams benötigt, der angibt wie leistungsfähig dies ist. Dieser Wert wird aus Erfahrungswerten ermittelt und heißt *Velocity*. Jedes Team besitzt einen individuellen Velocity-Wert [Max13].

Sprint Backlog

Das *Sprint Backlog* wird jeweils im ersten Teil des *Sprint Planning* aus dem Product Backlog mit jenen Items erstellt, welche zur Erreichung des Sprint Ziels notwendig sind. Im zweiten Teil des Sprint Planning werden die Einträge dann verfeinert und eine detailliertere Schätzung des Aufwands erstellt. Nur Mitglieder des Entwicklungsteams dürfen Änderungen am Sprint Backlog vornehmen. Es wird von ihnen während des Sprints kontinuierlich angepasst und zeigt den aktuellen Fortschritt des Sprints an.

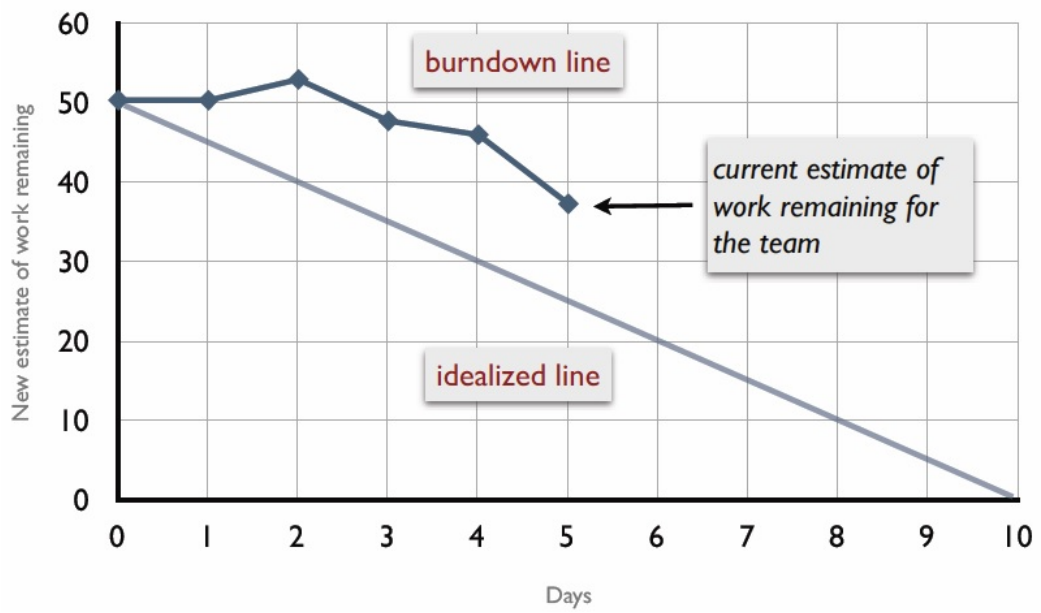


Abbildung 2.32: Burndown-Diagramm [Sut10]

Zur Visualisierung des Fortschritts werden häufig Burndown-Diagramme eingesetzt. Abbildung 2.32 zeigt wie ein solches Diagramm aussehen kann. Der aktuelle Projektfortschritt wird dabei mit einer idealisierten Linie verglichen [Glo13].

Impediment Backlog

Das *Impediment Backlog* wird vom Scrum Master verwaltet und stetig aktualisiert. In diesem werden alle Hindernisse gesammelt, welche die Produktivität des Scrum Teams vermindern. Das Team prio-

risiert die Hindernisse, welche ihren Ursprung sowohl im Team als auch in der Organisation haben können. Für die Beseitigung der Hindernisse ist der Scrum Master verantwortlich [Glo13].

Product Inkrement

Das Ergebnis eines jeden Sprints ist ein *Product Inkrement*. Das Inkrement eines folgenden Sprints baut immer auf dem des vorherigen auf. Am Ende eines Sprints muss es immer in einem potenziell auslieferbaren Zustand sein. Noch unvollständig implementierte Funktionen dürfen darin somit nicht enthalten sein [SS13].

In diesem Zusammenhang spielt die sogenannte *Definition of Done* eine wichtige Rolle. Diese wird von jedem Scrum Team individuell erstellt und gepflegt. Es handelt sich hierbei um eine interne Vereinbarung welche Voraussetzungen ein Backlog-Item erfüllen muss, damit es als *Done* bezeichnet werden darf [SS13].

2.13.1.4 Ereignisse

Bei den in Scrum beschriebenen Ereignissen handelt es sich hauptsächlich um Besprechungen. Die einzige Ausnahme bildet hierbei der Sprint, welcher als Container für die anderen Ereignisse dient. Die Hauptaufgabe der Besprechungen ist es, dem Team regelmäßig die Möglichkeit zur Überprüfung und Anpassung zu geben. Um den bestmöglichen Erfolg von Scrum und des Projektes nicht zu gefährden, darf keines der Ereignisse gestrichen werden. Um die Zeit dabei möglichst effizient zu nutzen, wird für jedes Ereignis im vornherein eine zeitliche Obergrenze festgelegt. Diese darf, einmal festgelegt, nicht wieder verändert werden und muss strikt eingehalten werden. Sobald ein Ereignis seinen Zweck erfüllt hat, darf es sofort beendet werden [SS13]. Im Folgenden werden die verschiedenen Ereignisse kurz beschrieben.

Sprint

Der *Sprint* ist ein besonderes Ereignis in Scrum. Es handelt sich hierbei nicht um eine Besprechung, sondern um ein kleines Projekt mit einem festen Zeithorizont. Innerhalb dieser Zeit soll ein fertiges, potenziell auslieferbares *Product Inkrement* entwickelt werden. Jeder Sprint innerhalb eines Scrum Projektes sollte gleich lang sein und nicht länger als einen Monat dauern. Bei zu großen Zeitintervallen der einzelnen Sprints besteht die Gefahr, dass sich die Anforderungen während eines Sprints ändern oder die Komplexität ansteigt. Ein Sprint beinhaltet immer ein bestimmtes Ziel, welches am Anfang festgelegt wird. Dieses Ziel darf danach nicht geändert werden. Einzig der Anforderungsumfang kann in Absprache zwischen dem Product Owner und dem Entwicklungsteam neu festgelegt werden. Sollte das Sprint-Ziel obsolet werden, kann der Sprint zu jederzeit abgebrochen werden. Einen Abbruch kann allerdings nur der Product Owner veranlassen [SS13].

Sprint Planning

Das *Sprint Planning* findet jeweils am Anfang eines jeden Sprints statt und besteht aus zwei separaten Treffen. Bei diesen Treffen kommt jeweils das gesamte Scrum Team zusammen, um den kommenden Sprint zu planen. Auch Zuschauer können diesen Besprechungen beiwohnen, allerdings ist es ihnen nicht erlaubt aktiv in diese einzugreifen. Die Treffen sollten jeweils nicht länger als vier Stunden dauern und finden am selben Tag statt [SS13].

Bei der ersten Besprechung wird festgelegt, welche Funktionalitäten im Product Inkrement des kommenden Sprints enthalten sein sollen. Der Product Owner stellt zunächst das Ziel vor, welches er mit dem Sprint erreichen will und benennt die Funktionalitäten, die zur Erreichung des Ziels notwendig sind. Bei der Bestimmung der Anzahl der ausgewählten Einträge hat das Entwicklungsteam immer das letzte Wort, da nur sie wissen was in der limitierten Zeit machbar ist. Anschließend arbeitet das Scrum Team gemeinschaftlich die genauen Anforderungen für diese Funktionen heraus und definiert ein *Sprint-Ziel*. Dieses bildet die Messlatte, was im kommenden Sprint erreicht werden soll. Es dient weiterhin auch als Leitfaden für das Entwicklungsteam. Es soll das Team motivieren zusammen auf dieses Ziel hinzuarbeiten [Glo13].

Bei der zweiten Besprechung geht es darum, wie die ausgewählte Arbeit erledigt wird. Das Entwicklungsteam soll dies selbständig erarbeiten. Alle weiteren Akteure sind weiterhin anwesend, sollen aber nur bei Rückfragen aktiv in das Treffen eingreifen. Das Entwicklungsteam erarbeitet einen Plan, wie die einzelnen ausgewählten Funktionen in Teilaufgaben herunter gebrochen werden können. Die Teilaufgaben können sich bezüglich des geschätzten Aufwands unterscheiden, sollten aber jeweils nicht länger als einen Tag dauern. Wenn das Team während der Planung feststellt, dass es entweder zu viel oder zu wenig Arbeit für den aktuellen Sprint ausgewählt hat, kann es in Absprache mit dem Product Owner die ausgewählten Funktionen neu aushandeln. Am Ende der zweiten Besprechung hat das Entwicklungsteam einen genauen Plan zur Erreichung des Sprint-Ziels [Glo13].

Daily Scrum

Das *Daily Scrum* ist ein tägliches Zusammenkommen des Teams, indem die laufenden Aktivitäten synchronisiert werden und ein Plan für den nächsten Tag aufgestellt wird. Die Besprechung findet täglich am gleichen Ort und zur gleichen Zeit statt und ist ein Pflichttermin für das gesamte Team. Dadurch wird eine Kontinuität in den Ablauf gebracht. Das Meeting dauert maximal 15 Minuten. In diesen beantwortet jedes Mitglied des Entwicklungsteams die folgenden drei Fragen:

1. Was habe ich seit dem letzten Treffen erreicht?
2. Was will ich bis zum nächsten Treffen erreichen?
3. Was steht mir dabei im Weg?

Während des Treffens sammelt der Scrum Masters die genannten Hindernisse im Impediment-Backlog und das Entwicklungsteam aktualisiert das Sprint Backlog. Im Anschluss an die Besprechung wissen alle wie der aktuelle Stand der Projektes ist und woran ihre Teamkollegen arbeiten. Es ist eine tägliche Möglichkeit zur Überprüfung und Anpassung. Im Anschluss an das Treffen finden häufig noch zusätzliche Treffen statt, in denen einzelne Details diskutiert, Hilfestellungen ausgetauscht oder Umplanungen der Arbeit besprochen werden. Bei diesen müssen nicht mehr alle Mitglieder anwesend sein [SS13, Glo13].

Sprint Review

Das *Sprint Review* findet jeweils am Ende eines Sprints statt. An diesem, maximal vierstündigen Treffen, nehmen sowohl das Scrum-Team als auch die Stakeholder teil. Das Scrum-Team präsentiert das Product Inkrement, welches im zurückliegenden Sprint erstellt wurde. Dabei dürfen nur Funktionen gezeigt werden, die potentiell auslieferbar sind. Mit dem präsentierten Produkt muss sich das Team

an seinen, im Sprint Planning formulierten, Sprint-Zielen messen lassen. Die Stakeholder haben im Anschluss an die Präsentation die Möglichkeit, Fragen zu dem vorgestellten Product Inkrement zu stellen [SS13, Glo13].

Im nächsten Schritt legt der Product Owner den Fortschritt des gesamten Projektes dar. Hier steht vor allem das Product Backlog im Fokus der Betrachtungen. Es wird auch die aktuelle Marktsituation betrachtet. Auf der Grundlage der aktuellen Projektstands und der Marktsituation wird die Priorisierung der Product Backlog-Items überprüft und gegebenenfalls angepasst. Somit erarbeiten die Teilnehmer gemeinschaftlich einen ersten groben Plan für den nächsten Sprint. Dieser kann dann als Grundlage für das Sprint Planning dienen [SS13, Glo13].

Sprint Retrospektive

Zum Abschluss eines jeden Sprints findet eine *Sprint Retrospektive* statt. Dies ist ein maximal dreistündiges internes Treffen des Scrum Teams. In diesem werden nochmal speziell die positiven sowie negativen Erfahrungen des Teams zusammengetragen. Diese können sich auf die beteiligten Menschen, Beziehungen, Prozesse oder auch auf benutzte Werkzeuge beziehen. Für die negativen Punkte wird nach möglichen Verbesserungen gesucht und sie werden anschließend vom Team nach ihrer Wichtigkeit sortiert. Der Scrum Master sammelt alle genannten Hindernisse im Impediment-Backlog. Aus den Erkenntnissen kann abschließend ein Plan zur Umsetzung von den Verbesserungen für die folgenden Sprints erstellt werden. Bei der späteren Umsetzung ist die Fähigkeit des Teams zur Selbstorganisation und Anpassung gefragt. Die Sprint Retrospektive am Ende eines jeden Sprints stellt die kontinuierliche Verbesserung in der Arbeitsweise des Scrum Teams sicher [SS13, Glo13].

2.13.2 Prozess

Abbildung 2.33 zeigt den Ablauf eines Scrum Projektes. Am Anfang steht der *Product Owner* mit einer *Vision*. Diese wird von ihm in ein *Product Backlog* umgewandelt. Danach startet der iterative Entwicklungsprozess. Vor jedem Sprint findet das *Sprint Planning* statt. Bei diesem wird das *Sprint Backlog* erstellt. Es folgt die Abarbeitung der im Sprint Backlog festgehaltenen Anforderungen im *Sprint*. Währenddessen findet jeden Tag ein *Daily Scrum* zur Überprüfung und Anpassung des Fortschritts statt. Am Ende des Sprints steht das *Product Inkrement*, welches in diesem erstellt wurde. Bevor der nächste Sprint startet, finden dann noch zwei weitere Treffen statt. Zum einen das *Sprint Review*, indem die Ergebnisse präsentiert werden und zum anderen die *Sprint Retrospektive*, indem versucht wird die Arbeitsbedingungen für die folgenden Sprints zu optimieren. Diese beiden Treffen sind ebenso ein Teil der kontinuierlichen Überprüfung und Anpassung [Sut10].

2.13.3 Scrum in LSOP

Liveanalysen im Sport mit Odysseus P2P (LSOP) ist ein Projekt, welches im universitärem Rahmen von einer zwölfköpfigen Projektgruppe bearbeitet wird. Die Projektdauer beträgt ein Jahr bei einer wöchentlichen Arbeitszeit von etwa zwei Tagen. Auf den ersten Blick scheint eine Eignung von Scrum unter diesen Voraussetzungen fragwürdig. Ein Daily Scrum ist nicht möglich und die Teamgröße liegt oberhalb des optimalen Bereichs.

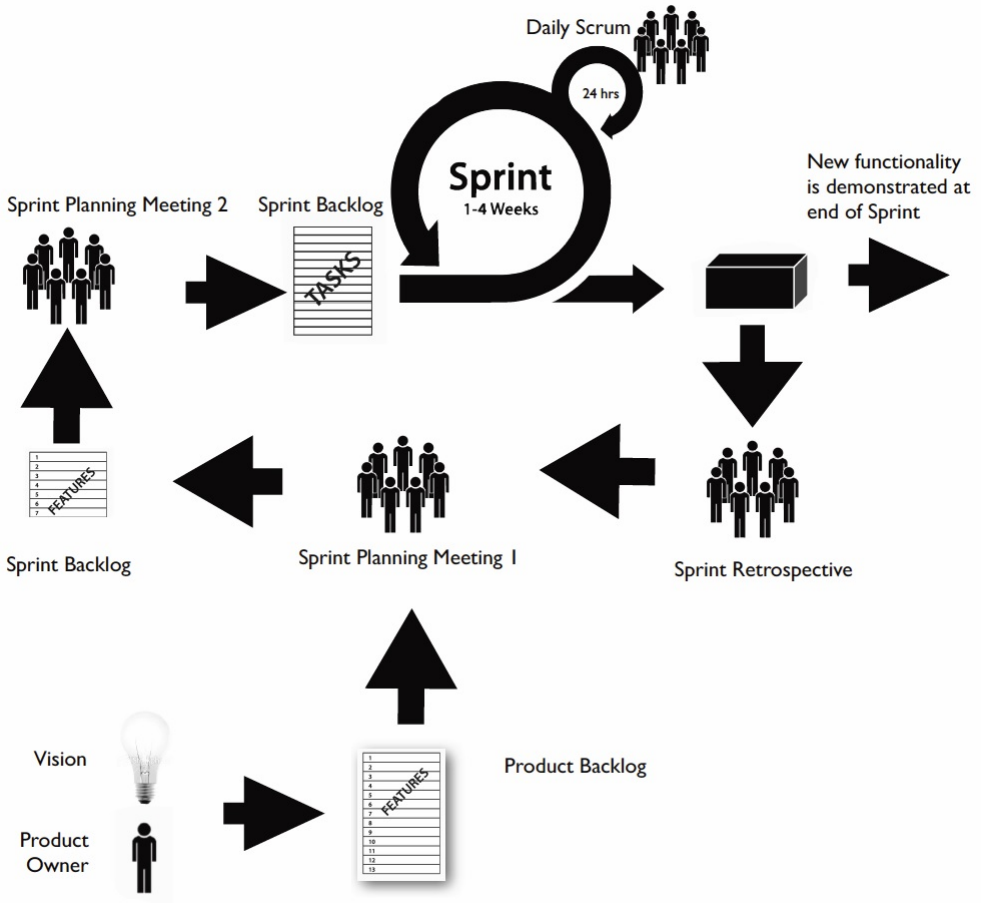


Abbildung 2.33: Scrum Prozess [Sut10]

Um Scrum dennoch im Projekt anwenden zu können, müssen einige Anpassungen an dem Rahmenwerk vorgenommen werden. Da es sich nicht um ein Vollzeitprojekt handelt müssen die Zeitrahmen vergrößert werden. So könnte man beispielsweise *Weekly Scrums* statt der täglichen Treffen ansetzen. Um weiterhin sicherzustellen, dass mit der limitierten Arbeitszeit innerhalb eines Sprints ein potenziell auslieferbares Produkt hergestellt werden kann, sollte die Sprintdauer ebenfalls verlängert werden. Diese beiden Anpassungen sind machbar, haben allerdings den Nachteil, dass die iterativen Überprüfungs- und Anpassungszyklen deutlich verlängert werden und somit das Risiko erhöht wird. In einem früheren Projekt haben wir aber schon gute Erfahrungen mit ähnlichen Anpassungen gemacht.

Der zweite kritische Punkt ist die Teamgröße. Bei einem Entwicklungsteam von zwölf Leuten ist der Koordinierungsaufwand relativ hoch. Erschwerend kommt noch hinzu, dass es dabei auch nur einen eingeschränkten persönlichen Kontakt gibt. Um die Koordination zu erleichtern sollten daher Tools eingesetzt werden. Ein Beispiel dafür ist *Jira*[27]. Dies kann helfen den Koordinationsaufwand in Grenzen zu halten.

[27] https://www.atlassian.com/software/jira

Ein großer Vorteil von Scrum ist, dass Ergebnisse schon zu einem frühen Projektzeitpunkt sichtbar sind. Dies steigert die Motivation und den Zusammenhalt des Teams. Im universitärem Umfeld ist dies besonders nützlich, da die Studenten meistens noch keine große Projekterfahrung besitzen.

2.13.4 Zusammenfassung

Scrum ist kein striktes Vorgehensmodell, sondern vielmehr ein Rahmenwerk. Es bietet die Rahmenbedingen um ein komplexes Produkt zu entwickeln. Das Modell definiert innerhalb des Scrum Teams die drei Rollen Product Owner, Scrum Master und Entwicklungsteam. Weiterhin werden fünf Ereignisse und vier Artefakte sowie Regeln definiert, welche diese Komponenten miteinander verbinden. Die Softwareentwicklung findet dabei in iterativen Zyklen statt. Durch das Rahmenwerk werden regelmäßige Möglichkeiten zur Überprüfung und Anpassung sowohl des Projektverlaufs als auch der Arbeitsweise ermöglicht. Mit einigen Anpassungen ist Scrum auch im Rahmen des Projektes LSOP anwendbar.

Jedes Projektumfeld ist unterschiedlich und somit müssen auch immer Kompromisse bei der Vorgehensweise eingegangen werden. Bei Scrum stehen die Teammitglieder in der Hierarchie über den verwendeten Prozessen und Werkzeugen. Wenn das Projektteam die Grundüberzeugung von Scrum teilt, dann ist es auch mit den notwendigen Anpassungen für den Einsatz im Projekt LSOP geeignet.

2.14 Zusammenfassung

Mit den Grundlagen wurde die Basis geschaffen, um Herakles zu verwirklichen. Dazu gehörten die Statistiken, die in der Sportanalyse typischerweise verwendet werden. Außerdem wurden die Grundlagen für Sensorsysteme erklärt, auf deren Basis die Positionsdaten der Spieler ermittelt werden könnten. Des Weiteren wurden andere Werkzeuge für Sportanalyse, die sich am Markt befinden, betrachtet, um andere Umsetzungsmöglichkeiten aufzuzeigen, mit denen Herakles grundsätzlich verglichen werden kann.

In den folgenden Grundlagenabschnitten wurden Themen zur Datenanalyse in DSMS behandelt. Nach einer Einführung in das Konzept von Datenströmen und dem DSMS Odysseus wurden Techniken vorgestellt, die in diesem Kontext verwendet werden können. Dazu gehören Admission Control, Replikation und Fragmentierung, Load-Balancing und Recovery. Da Herakles in einem P2P-Netzwerk genutzt werden soll, wurden auch die Grundlagen für diesen Themenbereich vorgestellt. Dazu gehört allgemein das Peer Computing und im speziellen JXTA, das zum Aufbau des P2P-Netzwerkes genutzt wird.

Zum Schluss wurde Android und die Entwicklung für diese Plattform vorgestellt, da die Anwendung für den Trainer auf Android-Basis realisiert werden soll.

3 Anforderungen

Im folgenden Kapitel werden die Anforderungen an die Projektgruppe erläutert. Diese wurden von dem Product Owner, in diesem Fall dem Projektgruppenbetreuer, vorgegeben. Die Anforderungen wurden im Detail als User Stories beschrieben und einem Epic zugeordnet. Diese wurden durch die gewählte agile Vorgehensweise (vgl. Kapitel 8) erarbeitet. Im Laufe des Projekts wurden die Epics und User Stories kontinuierlich an veränderte Anforderungen angepasst. Dies ermöglicht eine flexible Handhabung von äußeren Einflüssen auf die Projektbedingungen.

3.1 Fachliche Anforderungen

Die Projektgruppe definiert zusammen mit dem Product Owner die Anforderungen an die Software, die im Laufe des Projekts angepasst und erweitert werden. Das Hauptziel des Projekts besteht darin, Odysseus P2P in einem konkreten Anwendungsfall einzusetzen und zu evaluieren. Dafür wurde als Szenario eine Liveanalyse von Sportdaten gewählt. Diese gilt es effizient, effektiv und zuverlässig in einem dezentralen, heterogenen, dynamischen und autonomen P2P-Netzwerk zu erstellen. Die Ergebnisse der Analyse sollen auf einer grafischen Oberfläche dargestellt werden. Um dieses Ziel zu erreichen wurden verschiedene Bereiche identifiziert, die jeweils eigene Anforderungen erfüllen müssen. Diese sind in den nachfolgenden Tabellen 3.1 bis 3.7 aufgeführt.

Um Odysseus P2P stabiler zu machen, wird das bestehende System um die beiden Funktionen des Load-Balancings und der Recovery erweitert. Damit die Auslastung auf den einzelnen Peers möglichst gleichmäßig verteilt ist, muss es einen Mechanismus geben, der die Lastverteilung automatisch regelt. Dies soll den Flaschenhals, der sich bei überlasteten Peers im Netzwerk ergibt, minimieren. Zum Load-Balancing wurden folgende Anforderungen erhoben:

Kennziffer	Priorität	Anforderung
LB01	Hoch	Odysseus P2P muss Grundlegende Load-Balancing Konzepte realisieren
LB02	Hoch	Odysseus P2P muss eine Einfache Load-Balancing Strategie (Round Robin) integrieren
LB03	Hoch	Odysseus P2P muss die Zustandsübertragung durch die Parallel-Track-Strategy ermöglichen
LB04	Mittel	Odysseus P2P soll die Zustandsübertragung durch Moving-State-Strategy ermöglichen
LB05	Gering	Odysseus P2P kann das Kräftemodell für das Load-Balancing realisieren

Tabelle 3.1: Anforderungen Load-Balancing

Da im P2P-Netzwerk dynamisch Peers das Netzwerk betreten und wieder verlassen können, muss das P2P-Netzwerk robust gegenüber Ausfällen von einzelnen Peers sein. Hierfür muss es ein Recovery-Protokoll geben, welches die Anfragen des ausgefallenen Peers neu im Netzwerk verteilt und so die Fortführung der Bearbeitung von Anfragen gewährleistet:

Kennziffer	Priorität	Anforderung
RE01	Hoch	Odysseus P2P muss Grundlegende Recovery Prozesse durchführen
RE02	Mittel	Odysseus P2P soll Active-Standby unterstützen
RE03	Gering	Odysseus P2P kann Upstream-Backup unterstützen
RE04	Gering	Das Zusammenspiel zwischen Active-Standby und Upstream-Backup funktioniert

Tabelle 3.2: Anforderungen Recovery

Odysseus P2P muss mit den Clients, insbesondere der Coach App, kommunizieren können. Die folgenden Anforderungen ergeben sich daher für die Kommunikation zwischen der Coach App und Odysseus P2P.

Kennziffer	Priorität	Anforderung
KO01	Hoch	Odysseus P2P muss eine Zwischensprache (SportsQL) unterstützen
KO02	Hoch	Die Kommunikation (Daten) zwischen Coach App und Odysseus P2P muss durch Sockets realisiert sein
KO03	Hoch	Die Kommunikation (Steuerung) zwischen Coach App und Odysseus P2P muss über REST realisiert sein
KO04	Mittel	Es muss eine Zwischenkomponente vorhanden sein, die die Übersetzung und Verarbeitung sowie das Stellen der Anfragen ermöglicht und als Grundlage für andere Client Anwendungen dient
KO05	Mittel	Die Zwischenkomponente soll durch ein Continuous Integration (CI) System kompiliert sein
KO06	Mittel	Die Zwischenkomponente soll automatisch als Artefakt den Client Anwendungen zur Verfügung stehen
KO07	Mittel	Die statischen Daten (Metainformationen) sollen in Client Anwendungen angefragt und genutzt werden
KO08	Gering	Client Anwendungen können über Peer oder Socket-Ausfälle informieren, sodass die Verarbeitung auch nach dem Ausfall eines Peers weitergeführt werden kann, der die Daten an den Client liefert

Tabelle 3.3: Anforderungen Kommunikation

Zur Darstellung der Analyseergebnisse und die Aufbereitung für die Trainer als Zielgruppe hat die Projektgruppe sich für eine Android Applikation entschieden und die folgenden Anforderungen aufgestellt:

Kennziffer	Priorität	Anforderung
CA01	Hoch	Die Coach App muss Anfragen automatisch starten
CA02	Hoch	Die Coach App muss SportsQL nutzen
CA03	Hoch	Die Coach App muss mehrere Sportarten unterstützen
CA04	Hoch	Die Coach App muss die Bewegungen der Spieler auf dem Spielfeld anzeigen
CA05	Hoch	Die Coach App muss die Analysezeit synchron zu den Bewegungen der Spieler darstellen
CA06	Hoch	Die Coach App muss alle relevanten Team-Statistiken anzeigen
CA07	Hoch	Die Coach App muss alle relevanten Spieler-Statistiken anzeigen
CA08	Mittel	Die Coach App soll einfach auf weitere Sportarten erweiterbar sein
CA09	Mittel	Die Coach App soll die Möglichkeit bieten Anfragen manuell zu starten, stoppen und pausieren
CA10	Mittel	Die Coach App soll die Möglichkeit bieten, Ketten farblich zu kennzeichnen, die auch während der Spielerbewegungen dargestellt werden
CA11	Mittel	Die Coach App soll die Möglichkeit bieten, die Laufwege der Spieler anzuzeigen
CA12	Gering	Der Trainer kann in der Coach App den Spielverlauf anhalten und Notizen vornehmen
CA13	Gering	Die Spielerpositionen können mit einer Heatmap visualisiert werden

Tabelle 3.4: Anforderungen Coach App

Um selbstständig Daten für Analysen erzeugen zu können und damit die Funktionstüchtigkeit des Konzepts zu beweisen, müssen Sensoren eingesetzt werden. Folgende Anforderungen wurden dabei aufgestellt:

Kennziffer	Priorität	Anforderung
RE01	Hoch	Die Projektgruppe muss verschiedene Sensorsysteme und Techniken evaluieren
RE02	Mittel	Die Projektgruppe soll eigene Sensordaten mit der GPS-Technik aufnehmen
RE03	Gering	Die Projektgruppe kann eigene Sensordaten mit der WLAN-Triangulations-Technik aufnehmen
RE04	Gering	Die statischen Informationen können über die GPSender App manipuliert werden

Tabelle 3.5: Anforderungen Sensorik

Zur zentralen Überwachung und Steuerung des dezentralen P2P-Netzwerks soll eine Monitoring Application entwickelt werden.

Kennziffer	Priorität	Anforderung
RE01	Hoch	Die Monitoring Application gibt eine Übersicht über alle verfügbaren Odysseus-Peers in dem Netzwerk
RE02	Hoch	Innerhalb der Monitoring Application ist es möglich, die Peers über eine Konsole zu steuern
RE03	Hoch	Die Monitoring Application bietet die Möglichkeit, die Ping-Map eines Peers anzeigen zu lassen
RE04	Mittel	Der Log eines Peers kann mit der Monitoring Application angezeigt werden
RE05	Gering	Standardfunktionen (z.B. Load-Balancing einschalten) der Peers können innerhalb der Monitoring Application über Menüs ausgewählt werden
RE06	Gering	Verteilte Anfragen können über die Monitoring Application visualisiert werden

Tabelle 3.6: Anforderungen Monitoring Application

Um zeitnah Statistiken ermitteln zu können, müssen kontinuierliche Anfragen auf die Daten, welche durch verschiedene Datenquellen geliefert werden, durchgeführt werden. Diese Anfragen unterliegen den folgenden Anforderungen.

3.2 Epics und User Stories

Nachfolgend werden die Epics, die sich während der Arbeit an Herakles ergeben haben, genannt und kurz beschrieben.

Load-Balancing dient als Erweiterung von Odysseus P2P um eine Lastverteilung.

Recovery dient als Erweiterung von Odysseus P2P um die Ausfallsicherheit zu erhöhen.

Kommunikation Client zu P2P baut die Brücke zwischen Odysseus P2P und den Clients.

Sensorik enthält alles, was mit der Erhebung von eigenen Sensordaten zutun hat.

Schnittstelle zur Sensorik beschreibt den Teil der Anbindung der Sensoren an Herakles.

Data-Mining auf Sensordaten enthält die Verarbeitung der Daten aus den Datenquellen.

Coach App bereitet die Ergebnisse der Anfragen für den Trainer auf.

Monitoring Application kümmert sich um die Kontrolle und Steuerung des P2P-Netzwerks.

Developer Application dient als Entwicklungswerkzeug.

Kennziffer	Priorität	Anforderung
CA01	Hoch	Das System muss verschiedene Sportarten unterstützen
CA02	Hoch	Die Anfragen müssen von den Sensoren / Quellen unabhängig sein
CA03	Hoch	Grundlegende Anfragen für die Sportart Fußball müssen realisiert sein
CA04	Hoch	Gängige Statistiken müssen für die Sportart Fußball realisiert sein und funktionieren zuverlässig, allerdings abhängig von der Genauigkeit des Datenstroms
CA05	Hoch	Statische Informationen (Spielfeldabmessungen, Spielernamen usw.) müssen auf allen Peers genutzt werden können, ohne dass die Informationen auf allen Peers hinterlegt werden müssen
CA08	Mittel	Anfragen sollen für Spielunterbrechungen manuell gestartet, gestoppt und pausiert werden
CA09	Mittel	Grundlegende Anfragen für die Sportart Basketball sollen realisiert sein
CA10	Mittel	Ausgewählte Anfragen sollen die vorhandene Fragmentierung nutzen
CA11	Gering	Anfragen können nach dem Survey Verfahren auf verschiedene Peers des Netzwerks verteilt werden
CA12	Gering	Seitenwechsel können während der Ausführung der Anfragen durchgeführt werden
CA13	Gering	Spielunterbrechungen können automatisch erkannt werden und Anfragen werden entsprechend gestoppt und gestartet

Tabelle 3.7: Anforderungen Anfragen

Projektdokumentation beschreibt alles, was mit der Dokumentation und Präsentation der Projektergebnisse zutun hat.

In den folgenden zwei Unterabschnitten finden sich die umgesetzten User Stories, welche nach Epics geordnet sind, sowie die nicht umgesetzten User Stories nach Bezeichner geordnet.

3.2.1 Umgesetzte User Stories

Load-Balancing

Bezeichner: LSOP-22
Name: Zustandslose-Operatoren verschieben
Beschreibung: Als Zustandsloser-Operator möchte ich über verschiedene Peers verschoben werden können, damit Load-Balancing möglich wird.
Kriterien:

- Das Verschieben vom SELECT-Operator von einem Peer zu einem anderen ist möglich.
- Das Verschieben vom PROJECT-Operator von einem Peer zu einem anderen ist möglich.
- Das Verschieben vom RENAME-Operator von einem Peer zu einem anderen ist möglich.

- Das Verschieben von zustandlosen WINDOW-Operatoren von einem Peer zu einem anderen ist möglich.

Bezeichner: LSOP-23
Name: Zustandsbehaftete-Operatoren verschieben
Beschreibung: Als Zustandsbehafteter-Operator möchte ich über verschiedene Peers verschoben werden können, damit Load-Balancing möglich wird.
Kriterien:

- Das Verschieben vom JOIN-Operator von einem Peer zu einem anderen ist möglich.
- Das Verschieben vom UNION-Operator von einem Peer zu einem anderen ist möglich.
- Das Verschieben vom AGGREGATE-Operator von einem Peer zu einem anderen ist möglich.
- Das Verschieben von zustandbehafteten WINDOW-Operatoren von einem Peer zu einem anderen ist möglich.

Bezeichner: LSOP-24
Name: Einfache Load-Balancing-Strategie integrieren
Beschreibung: Als System möchte ich durch eine einfache Load-Balancing Strategie in der Lage sein, Last umzuverteilen, damit die Operatoren-Verschiebung getestet werden kann.
Kriterien:

- Eine einfache Load-Balancing Strategie ist implementiert, so dass ein Load-Balancing durchgeführt werden kann.

Bezeichner: LSOP-38
Name: Grundlegendes Verschieben von Anfragen ermöglichen
Beschreibung: Als System möchte ich das Load-Balancing nutzen, um den Datenstrom auf andere Peers umleiten zu können ohne dass Daten des Stroms verloren gehen, sodass andere Peers die Last übernehmen können.
Kriterien:

- JXTA Sender und Receiver sind richtig gesetzt.
- Start und Ende des Load-Balancing Vorgangs sind gekennzeichnet.

Bezeichner: LSOP-178
Name: Load-Balancing Protokoll erweitern
Beschreibung: Als System möchte ich auf den Peers ein Load-Balancing Protokoll verwenden, welches auch Fehlerfälle berücksichtigt. Derzeit setzt das Load-Balancing-Protokoll nur den Idealfall um (im LoadBalancingListener). Dieses Protokoll ist um die Berücksichtigung von Fehlerfällen zu erweitern. Im Konzept sind bereits viele mögliche Fehlerfälle beschrieben, weitere mögliche Fehler sollen identifiziert werden.
Kriterien:

- Das Load-Balancing Protokoll ist durch eine Fehlerbehandlung erweitert.

Bezeichner: LSOP-219
Name: Dokumentation des (neuen) Load-Balancing Protokolls
Beschreibung: Als Entwickler möchte ich, dass das neue Load-Balancing Protokoll ausführlich dokumentiert wird um die Wartung zu erleichern.
Kriterien:

- Dokumentation ist erstellt.

Bezeichner: LSOP-323
Name: Zustandsbehaftete Operatoren verschieben
Beschreibung: Als System möchte ich auch zustandsbehaftete Operatoren verschieben können, sodass alle Arten von Anfragen auf andere Peers übertragen werden können.
Kriterien:

- Ausgewählte zustandsbehaftete Operatoren werden beim Load-Balancing übertragen.

Bezeichner: LSOP-348
Name: Zustände für weitere Operatoren implementieren
Beschreibung: Als System möchte ich, dass weitere Operatoren mit der MovingState Strategy verschoben werden können. Dazu müssen diese Operatoren die Methoden des Interfaces IStatefulPO implementieren. Diese Story stellt eine Erweiterung zu LSOP-323 dar.
Kriterien:

- Window Operatoren können beim Load-Balancing verschoben werden.
- Weitere zustandsbehaftete Operatoren können beim Load-Balancing verschoben werden.

Recovery

Bezeichner: LSOP-48
Name: Recovery-Konzept im P2P Bereich von Odysseus evaluieren
Beschreibung: Als Entwickler möchte ich evaluieren wie im P2P Bereich Recovery-Konzepte eingesetzt werden können, sodass wir für weitere Umsetzungen eine Grundlage haben.
Kriterien:

- Ergebnisse der Evaluation sind dokumentiert.

Bezeichner: LSOP-150
Name: Detailliertes Recovery-Konzept für Odysseus P2P ausarbeiten
Beschreibung: Als Entwickler möchte ich ein ausgearbeitetes Konzept besitzen, wie im P2P Bereich Recovery-Konzepte eingesetzt werden können, sodass Instruktionen für die Implementation dieser Konzepte (Klassen, Abläufe und Techniken) gegeben sind.
Kriterien:

- Die Artefakte des ausgearbeiteten Konzepts sind dokumentiert.

Bezeichner: LSOP-209
Name: Allgemeine Recovery-Konzepte Implementieren
Beschreibung: Als Nutzer möchte ich die Gewissheit haben, dass bei Ausfällen der Infrastruktur (Peer Ausfällen) Sicherheitsmaßnahmen ergriffen werden, um eine fehlerfreie Bearbeitung meiner Anfragen schnellstmöglich wieder zu erreichen.
Kriterien:

- Peer-Ausfälle können erkannt werden.
- Einfaches Auswählen eines neuen Peers zur Übernahme nach Peer-Ausfall ist möglich.
- Übertragung der Informationen über eine Anfrage auf einen neuen Peer ist möglich.

- Vorgehen ist dokumentiert.

Bezeichner: LSOP-210
Name: Active-Standby Konzepte Implementieren
Beschreibung: Als System möchte ich die Funktionalität besitzen, Query-Parts parallel von mehreren Peers bearbeiten zu lassen, so dass bei Ausfällen der Infrastruktur (Peer Ausfällen) eine fehlerfreie Bearbeitung meiner Anfragen schnellstmöglich wieder erreicht werden kann bzw. erst gar nicht gefährdet ist.
Kriterien:

- Neuen Backup-Peer wird gefunden und initialisiert.
- In der Recovering Phase wird eine möglichst korrekte Ausgabe erzeugt.
- Vorgehen ist dokumentiert. Dazu ist die Dokumentation von LSOP-209 zu erweitern.

Bezeichner: LSOP-345
Name: Recovery Umstrukturieren
Beschreibung: Als System möchte ich eine übersichtliche und leicht erweiterbare Struktur des Recoverys besitzen.
Kriterien:

- Allokatoren sind in eigene Projekte ausgelagert.
- Alle nötigen Schnittstellen sind erstellt.

Bezeichner: LSOP-367
Name: Recoveryprotokoll robuster machen
Beschreibung: Als System möchte ich ein zuverlässiges Recovery-Protokoll besitzen. Durch ein Protokoll mit Acks, etc. und evtl. weiteren Maßnahmen muss das Recovery robuster gemacht werden.
Kriterien:

- Das Nachrichtenprotokoll des Recovery ist robuster durch zusätzliche Sicherheitsmechanismen.
- Das Recovery ist insgesamt robuster gegenüber Fehlerfällen.

Bezeichner: LSOP-411
Name: Recovery Fehlerfälle behandeln und dokumentieren
Beschreibung: Als System möchte ich mehr Robustheit durch eine Fehlerbehandlung innerhalb des Nachrichten-Protokolls. Die Fehlerfälle, die beim Senden einer Nachricht auftreten können, müssen dokumentiert werden. Es muss auch beschrieben werden, wie diese Fehler behandelt werden. Die Fehlerbehandlung muss implementiert werden.
Kriterien:

- Die Fehlerfälle sind behandelt und dokumentiert.

Bezeichner: LSOP-464
Name: Dokumentation Recovery Implementierung
Beschreibung: Als Produktbesitzer möchte ich eine ausführliche Dokumentation der Recovery. Die aktuelle Implementierung unterscheidet sich grundlegend von dem ursprünglich erstellten Konzept. Hier sollen entscheidende Aspekte der Umsetzung dokumentiert werden.
Kriterien:

- Das Recovery-Konzept ist um die Implementierungsaspekte erweitert.

Bezeichner: LSOP-480
Name: Recovery dokumentieren
Beschreibung: Als Produktbesitzer möchte ich eine ausführliche Dokumentation der Recovery. In dieser Story sollen Tasks zusammengefasst werden, in denen es um die Dokumentation der Recovery geht. Weiterführung von LSOP-464.
Kriterien:

- Das Recovery-Konzept ist überarbeitet und gegebenenfalls ergänzt.

Kommunikation Client zu P2P

Bezeichner: LSOP-31
Name: Zwischensprache entwickeln
Beschreibung: Als System möchte ich innerhalb der Client-Anwendungen keine PQL Statements generieren, sondern die Anfragen an den Host über Keywords ermöglichen.
Kriterien:

- Die Syntax der Zwischensprache ist entwickelt.
- Die Syntax der Zwischensprache ist dokumentiert.

Bezeichner: LSOP-32
Name: Übersetzungskomponente für Zwischensprache <-> Odysseus P2P
Beschreibung: Als System möchte ich die Keywords des Clients in syntaktisch korrekte Anfragepläne übersetzen können, sodass der Client keine Statements generieren muss.
Kriterien:

- Keywords werden in Anfragepläne übersetzt.
- Die Erweiterbarkeit der Komponente ist im hohen Maße gegeben.

Bezeichner: LSOP-45
Name: Kommunikation zwischen P2P und Client ermöglichen
Beschreibung: Als System möchte ich meinen Datenstrom aus dem P2P Netzwerk verschiedenen Clients zur Verfügung stellen können, sodass diese Auswertungen anzeigen können.
Kriterien:

- Grundlage zur Kommunikation ist vorhanden.

Bezeichner: LSOP-46
Name: Relevante Auswertungen ermitteln
Beschreibung: Als Nutzer möchte ich für mich relevante Auswertungen angezeigt bekommen, sodass mich diese bei der Analyse des Spiels unterstützen.
Kriterien:

- Fragenkatalog ist erstellt.
- Befragung ist eines Fußballvereins ist durchgeführt.
- Ergebnisse sind dokumentiert.

Bezeichner: LSOP-89
Name: Herstellung der Kommunikation zwischen Communiation Layer (+ JXTA) und P2P

Beschreibung: Als Entwickler möchte ich, dass das Graphical User Interface (GUI) Odysseus Instanzen finden und mit diesen kommunizieren kann, damit Datenströme angezeigt werden.
Kriterien:

- PQL Statements können an Odysseus Instanz angefragt werden.
- Erweiterbarkeit ist gegeben(insbesondere Umstellung auf Zwischensprache).
- UML-Diagramme sind erstellt(Klassen-, Sequenz- und Anwendungsfalldiagramme).

Bezeichner: LSOP-134
Name: Neuausrichtung des Communication Layers
Beschreibung: Als Entwickler möchte ich einen gemeinsamen Layer haben, in dem Dienste angeboten werden, die von der Android App und der Entwickler App zusammen genutzt werden können.
Nach Feststellung, dass gerade die Kommunikation wohl nicht in dem "Communication Layer"gelöst werden kann, da Android und Java hier zu unterschiedlich sind, soll der entsprechende Layer neu ausgerichtet und passend umbenannt werden. (z.B. ServiceLayer). Außerdem muss die inhaltliche Neuausrichtung durchdacht und knapp dokumentiert sowie vorgestellt werden.
Kriterien:

- Konzept für die inhaltliche Neuausrichtung des LSOPService ist ausgearbeitet und dokumentiert.

Bezeichner: LSOP-237
Name: REST Schnittstelle erweitern
Beschreibung: Als Entwickler möchte ich die Verwaltung von Anfragen in Odysseus steuern können, sodass auch Anfragen angehalten oder gelöscht werden können, wenn diese nicht mehr benötigt werden.
Kriterien:

- Die Rest-Schnittstelle ist funktional so erweitert, dass Anfragen gestartet, gestoppt und gelöscht werden können.

Bezeichner: LSOP-277
Name: Metadatenströme sollen in der Coach App verfügbar gemacht werden
Beschreibung: Als System möchte ich auf die Metadaten zugreifen können.
Kriterien:

- Die Metadaten im DDC sind per LSOP Service abrufbar.
- Die Metadaten aus dem LSOP Service werden in der Coach-App verwendet.

Sensorik

Bezeichner: LSOP-40
Name: Evaluierung der verschiedenen Sensoren auf Basis des Vortrags
Beschreibung: Als Entwickler möchten wir wissen welche Sensoren wir verwenden werden, sodass wir diese möglichst zeitnah anschaffen können.
Kriterien:

- Verschiedene Angebote sind verglichen.
- Unternehmen sind angeschrieben.

Bezeichner: LSOP-391
Name: GPS Daten aufzeichnen
Beschreibung: Als Entwickler möchte ich das System anhand eines Testszenarios mit GPS-Sensoren testen.
Kriterien:

- Die GPS-Sensoren lassen sich an das System anbinden.
- Das Testszenario ist beschrieben.
- Die Ergebnisse sind dokumentiert.

Bezeichner: LSOP-454
Name: Sensordaten mit der WLAN-Triangulationstechnik
Beschreibung: Als Entwickler möchte ich mit der WLAN Triangulationstechnik die Sensordaten aufnehmen um damit die Anwendungen unter realen Bedingungen testen zu können. Vergleiche auch mit LSOP-391.
Kriterien:

- Die erhobenen Daten lassen sich an das System anbinden.
- Das Testszenario ist beschrieben.
- Die Ergebnisse sind dokumentiert.

Schnittstelle zur Sensorik

Bezeichner: LSOP-176
Name: Intermediate Schema Konzept für Sensordaten erstellen
Beschreibung: Als Entwickler von Anfragen möchte ich unabhängig von den Sensoren immer das selbe Datenschema nutzen können. Die Zwischenschicht, die zwischen den rohen Sensordaten und den Anfragen liegt, soll das Schema vereinheitlichen und von den Sensoren abstrahieren sowie um weitere Informationen wie Metadaten ergänzen. In dieser Story soll diese Schicht konzipiert werden.
Kriterien:

- Konzept für das Intermediate Schema ist ausgearbeitet und dokumentiert.

Bezeichner: LSOP-208
Name: Umsetzung des Intermediate Schemas
Beschreibung: Als System möchte ich das entworfene Intermediate Schema verwenden.
Dazu gehören der DDC, ein Adapter für den zur Zeit verwendeten Beispiel-Sensoren-Datenstrom, sowie ein Adapter für den Spielereignis-Datenstrom.
Kriterien:

- Intermediate Schema aus LSOP-176 ist implementiert.

Bezeichner: LSOP-266
Name: Intermediate Schema erweitern
Beschreibung: Als System möchte ich das Intermediate Schema um Zeitintervalle und Hinterlegung der Datendurchsatzrate erweitert haben.
Kriterien:

- Die Zeiteinheit ist auf einen bestimmtes Intervall eingrenzbar sein.

- Maximaler Durchsatz und Umrechnungsfaktor des Datenstroms ist im Intermediate Schema festgelegt.

Bezeichner: LSOP-291
Name: Eat The Whistle anbinden
Beschreibung: Als Entwickler möchte ich Eat The Whistle als weitere Datenquelle neben dem Distributed Event-Based Systems (DEBS)-Datenstrom anbinden können. Dies soll die Unabhängigkeit des entworfenen Konzepts verdeutlichen.
Kriterien:

- Die Daten von Eat The Whistle sind auf das Intermediate Schema übersetzt.
- Die vorhandenen Anfragen funktionieren mit Eat The Whistle.

Bezeichner: LSOP-305
Name: Überarbeiten des OperatorBuildHelpers
Beschreibung: Als System benötige ich weitere Informationen im OperatorBuildHelper. Andere sind hingegen evtl. überflüssig geworden.
Kriterien:

- Ermittlung welche Mannschaft auf welches Tor spielt ist gegeben.
- Zeitangaben sind entfernt, sofern sie nicht benötigt werden.

Coach App

Bezeichner: LSOP-33
Name: Grundlegende Struktur der Coach App (GUI)
Beschreibung: Als Nutzer möchte ich die Coach App auf einem Tablet verwenden können, sodass ich mir Informationen des Spiels anzeigen lassen kann.
Kriterien:

- Mockups sind erstellt.
- UML Diagramme sind erstellt.
- Grundlegende Struktur in Android ist vorhanden. (erweiterbare Architektur)

Bezeichner: LSOP-34
Name: JXTA und REST automatisch nutzen
Beschreibung: Als System möchte ich, dass die Coach App sich automatisch mit einem bestehenden P2P Netzwerk aus Odysseus Instanzen verbinden kann, sodass ich den Datenstrom empfangen kann und sie keine IP Adresse des Hosts benötige.
Kriterien:

- P2P Netzwerke werden automatisch gefunden und verbunden.
- IP Adresse des Hosts muss nicht bekannt sein.
- Coach App erhält angefragte Daten.

Bezeichner: LSOP-143
Name: Spielübersicht in Anwender App
Beschreibung: Als Nutzer möchte ich auf meinem Tablet eine informative, übersichtliche und inter-

aktive Spielerübersicht haben.
Kriterien:

- Das Spielfeld wird in der Coach-App angezeigt.
- Die Spieler und der Ball werden in der Coach-App angezeigt.

Bezeichner: LSOP-220
Name: Aufräumen der Coach App
Beschreibung: Als Entwickler möchte ich, dass die Coach App übersichtlich strukturiert ist, sodass ich Funktionen einfach und ohne große Probleme hinzufügen kann.
Kriterien:

- Struktur ist übersichtlicher als vorher.

Bezeichner: LSOP-281
Name: Ketten in der Coach App anzeigen
Beschreibung: Als Nutzer möchte ich Ketten in der Coach App erstellen können.
Kriterien:

- Eine Funktion zum Erstellen für Ketten basierend auf LSOP-228 ist realisiert.

Bezeichner: LSOP-289
Name: Verknüpfung der Spielerliste mit dem Spielfeld
Beschreibung: Als Nutzer möchte ich den angewählten Spieler in der Spielerliste und auf dem Spielfeld angezeigt bekommen.
Kriterien:

- Der ausgewählte Spieler ist sowohl in der Spielerliste als auch auf dem Spielfeld erkennbar.

Bezeichner: LSOP-319
Name: Statistiken in Android anzeigen (Laufstrecke, Tore und Menüstruktur)
Beschreibung: Als Nutzer möchte ich Statistiken in der Coach-App angezeigt bekommen. Es sind schon einige Anfragen zu Statistiken vorhanden, jedoch noch keine Ansichten dafür in der Coach App. In den Tasks dieser Story sollen für einige der Statistiken coole Ansichten erstellt werden. Dazu muss erstmal eine allgemeine Menüstruktur aufgebaut werden (bei welchem Klick werden die Statistiken geöffnet, wie werden sie dort angezeigt?) und dann für einige Statistiken die Ergebnisse angezeigt werden. Dort sind Zahlen aber auch hübsche (evtl. interaktive?) Diagramme möglich.
Kriterien:

- Die Menüstruktur für Statistiken in der Coach App ist umgesetzt.
- Statistiken zur Laufstrecke werden angezeigt.
- Statistiken zu erzielten Toren werden angezeigt.

Bezeichner: LSOP-310
Name: Architektur der Coach App erweitern
Beschreibung: Als Entwickler möchte ich neue Sportarten leicht hinzufügen können. Die Architektur muss dies berücksichtigen.
Kriterien:

- Erweiterung der Architektur zur Unterscheidung von Sportarten ist gegeben.

Bezeichner: LSOP-331
Name: Weitere Statistiken in Android anzeigen
Beschreibung: Als Nutzer möchte ich Statistiken in der Coach App angezeigt bekommen. Dieser Task stellt eine Erweiterung zu LSOP-319 dar.
Kriterien:

- Torschuss-Statistik wird angezeigt.
- Ballkontakt-Statistik wird angezeigt.
- Ecken-Statistik wird angezeigt.
- Pass-Statistik wird angezeigt.
- Laufwege werden dargestellt.

Bezeichner: LSOP-370
Name: Heatmap in Coach App
Beschreibung: Als Nutzer der Coach App möchte ich eine Heat-Map angezeigt bekommen, die mir anzeigt, wo die Spieler sich häufig aufhalten.
Kriterien:

- Heat-Map ist in der Coach App implementiert.

Bezeichner: LSOP-383
Name: Funktionale Erweiterung der Statistiken in Coach App
Beschreibung: Als Nutzer der Coach App möchte ich weitere Funktionen, wie einen Spielervergleich oder das Anordnen der Teamstatistiken besitzen.
Kriterien:

- Der Spielervergleich in der Coach App ist möglich.
- Das Anordnen der Teamstatistiken in der Coach App ist möglich.

Bezeichner: LSOP-388
Name: Sprintstatistik in Coach App anzeigen
Beschreibung: Als Nutzer der Coach App möchte ich Sprint-Statistiken angezeigt bekommen.
Kriterien:

- Die Coach App ist um die Sprint-Statistik erweitert.

Bezeichner: LSOP-426
Name: Optimierung und Dokumentation Coach App
Beschreibung: Als Produktbesitzer möchte ich eine ausführliche Dokumentation der Coach App besitzen. Es sollen unterschiedliche Diagrammtypen zur Dokumentation erstellt werden. Weiterhin ist eine Optimierung des Codes erforderlich.
Kriterien:

- Diagramme zur Coach-App sind erstellt.
- Optimierungen am Code sind vorgenommen.

Bezeichner: LSOP-468
Name: Optimierungen in der Coach App
Beschreibung: Als Nutzer möchte ich eine verbesserte Benutzerschnittstelle.
Kriterien:

- Verschiedene Aspekte der Benutzerschnittstelle sind verbessert.

Bezeichner: LSOP-507
Name: Konfiguration von Anfragen innerhalb der Coach App
Beschreibung: Als Entwickler möchte ich zu Testzwecken Anfragen in der Coach App konfigurieren können.
Kriterien:

- Einstellungen in der Coach-App sind um Konfiguration der Anfragen ergänzt.

Data-Mining auf Sensordaten

Bezeichner: LSOP-47
Name: Grundlegende, einfache Muster erkennen
Beschreibung: Als Entwickler möchte ich herausfinden wie ich mit den Testdaten (DEBS-Datenstrom) umgehen kann und einfache Anfragen stellen kann, sodass Wissen dazu aufgebaut wird.
Kriterien:

- Anfragen basierend auf Rohdaten der Fußballdaten von DEBS-Datenstrom sind erstellt.
- Performanz ist beachtet.

Bezeichner: LSOP-82
Name: Evaluierung für Client Applications
Beschreibung: Als Entwickler möchte ich vor der Umsetzung zunächst die notwendigen Technologien evaluieren, sodass die Bearbeitung strukturiert durchgeführt werden kann.
Kriterien:

- Client Technologien sind evaluiert.
- Ergebnisse sind dokumentiert.

Bezeichner: LSOP-112
Name: Grundlegende Client-Architektur aufsetzen
Beschreibung: Als Entwickler möchte ich eine sinnvolle Client-Architektur nutzen können.
Kriterien:

- Handlungsvorschläge aus der Evaluation in LSOP-82 sind umgesetzt.

Bezeichner: LSOP-160
Name: Anfragepläne erstellen
Beschreibung: Als Entwickler möchte ich Zugriff auf folgende Statistiken haben.
Kriterien:

- Anfrageplan für die Anzahl der Pässe ist erstellt.
- Anfrageplan für die Anzahl der Tore ist erstellt.
- Anfrageplan für die Anzahl der Ballkontakte für einen spezifizierten Spieler ist erstellt.
- Anfrageplan für die Anzahl der Ecken ist erstellt.
- Anfrageplan für die Laufwege mit Ball ist erstellt.

Bezeichner: LSOP-180
Name: Verteilung von SportsQL-Anfragen

Beschreibung: Als Nutzer der SportsQL-Anfragen möchte ich, dass diese verteilt installiert werden und nicht nur auf einem Peer. In dieser Story muss auch überlegt werden, welche Art von Verteilung sinnvoll ist. (z.B. Partitionierung, Replikation, Fragmentierung, Allokation)
Kriterien:

- Konzept zur Verteilung der SportsQL-Anfragen ist erstellt und dokumentiert.

Bezeichner: LSOP-228
Name: Weitere Anfragepläne erstellen/erweitern
Beschreibung: Als Entwickler möchte ich Zugriff auf weitere Statistiken haben. Diese sind konzeptionell zu erarbeiten.
Kriterien:

- Ein Konzept für die Auswertung von Flanken ist erarbeitet.
- Ein Konzept für die Auswertung von Pässen ist erarbeitet.
- Ein Konzept für die Auswertung von Sprints ist erarbeitet.
- Ein Konzept für die Verwendung von Ketten ist erarbeitet.

Bezeichner: LSOP-259
Name: Grundsätzliches Verarbeitungskonzept von Anfragen
Beschreibung: Als Entwickler möchte ich wissen wie es realisiert werden kann, dass Ergebnisse von Anfragen aktuell sind und ob alle Anfragen jederzeit laufen müssen.
Kriterien:

- Klassifizierung von Anfragen ist dokumentiert.

Bezeichner: LSOP-299
Name: Vorhandene Anfragen überarbeiten
Beschreibung: Als Nutzer erwarte ich von den vorhandene Anfragen möglichst korrekte Ergebnisse. Dafür ist es nötig, dass die Anfragen so überarbeitet werden, dass sie bessere Ergebnisse liefern.
Kriterien:

- Die vorhandenen Anfragen sind genauer und bestenfalls schneller.
- Unnötige Informationen, wie der Metadatenstrom, sind entfernt.

Bezeichner: LSOP-440
Name: Sportarten Unabhängigkeit auf Odysseusseite umsetzen
Beschreibung: Als Entwickler möchte ich auf Seiten von Odysseus P2P eine Struktur erstellen, die um weitere Sportarten erweiterbar ist.
Kriterien:

- Paketstruktur ist auf die Erweiterung durch neue Sportarten ausgelegt.
- Anfragen für verschiedene Sportarten sind erstellt.

Monitoring Application

Bezeichner: LSOP-358
Name: Logging Monitoring Application
Beschreibung: Als Nutzer der Monitoring Application möchte ich Log-Nachrichten des Peer-Netz-

werks in der Monitoring Application angezeigt bekommen.
Kriterien:

- Logging Nachrichten sind in der Monitoring-App abrufbar.

Bezeichner: LSOP-361
Name: Layout der Monitoring Application
Beschreibung: Als Nutzer der Monitoring Application möchte ich eine ansprechende Benutzerschnittstelle verwenden.
Kriterien:

- Mockups zum Layout sind erstellt.
- Die Mockups sind umgesetzt.

Bezeichner: LSOP-368
Name: Ping Map Monitoring Application
Beschreibung: Als Nutzer der Monitoring Application möchte ich eine Ping-Map angezeigt bekommen, die mir zeigt, wie der Ping zwischen Teilnehmern eines Netzwerks aussieht.
Kriterien:

- Ping-Map ist in der Monitoring Application implementiert.

Bezeichner: LSOP-372
Name: Console Monitoring Application
Beschreibung: Als Nutzer der Monitoring Application möchte ich die Konsole eines Peers angezeigt bekommen.
Kriterien:

- Das Anzeigen der Konsole ist in der Monitoring Application implementiert.

Bezeichner: LSOP-374
Name: Darstellung von verteilten Anfragen Monitoring Application
Beschreibung: Als Nutzer der Monitoring Application möchte ich angezeigt bekommen, wie Anfragen auf verschiedene Peers verteilt sind.
Kriterien:

- Das Anzeigen des Peer-Graphen ist in der Monitoring Application implementiert. Peers sind gekennzeichnet.

Developer Application

Bezeichner: LSOP-27
Name: Grundlegende Entwickler-GUI
Beschreibung: Als Entwickler möchte ich eine einfache Entwickler-GUI verwenden können, die jedoch zunächst keine Daten empfangen kann.
Kriterien:

- Mockups der Oberfläche sind erstellt und dokumentiert.
- UML-Diagramme (Klassen-, Sequenz- und Anwendungsfalldiagramme) sind erstellt und dokumentiert.

- Die Projektstruktur ist aufgebaut.
- Einfache Java GUI nach MVC ist umgesetzt.(keine App für das Tablet)

Projektdokumentation

Bezeichner: LSOP-181
Name: Zwischenbericht vorbereiten
Beschreibung: Als Entwickler möchte ich eine Grundlage zur Erstellung des Zwischenberichts haben. Dazu muss dieser vorbereitet werden. Dazu gehört ein schriftlicher Bericht und eine Präsentation.
Kriterien:

- Inhalte für die Präsentation sind überlegt.
- Präsentation ist erstellt.

Bezeichner: LSOP-204
Name: BTW Konferenz in Hamburg
Beschreibung: Als Produktbesitzer möchte ich, dass wir an der BTW Konferenz in Hamburg teilnehmen.
Kriterien:

- Paper für die BTW Konferenz ist erstellt.

Bezeichner: LSOP-224
Name: Zwischenpräsentation vorbereiten
Beschreibung: Als Zwischenpräsentation möchte ich mit Inhalten gefüllt werden. Zu dem jedem einzelnen Gliederungspunkt soll es einzelne Tasks geben.
Kriterien:

- Gliederungspunkte sind definiert.
- Neue Tasks sind erstellt.

Bezeichner: LSOP-337
Name: Bugsammlung Zwischenpräsentation
Beschreibung: Als Entwickler möchte ich bestehende Fehler vor der Zwischenpräsentation adressieren.
Kriterien:

- Bekannte Fehler wurden ausgeräumt.

Bezeichner: LSOP-472
Name: Vorbereitung der Evaluationen
Beschreibung: Als Entwickler möchte ich für die Evaluationen Vorkehrungen treffen.
Kriterien:

- Evaluationsszenarien sind beschrieben.
- Evaluationsinfrastruktur ist aufgebaut und getestet.

Bezeichner: LSOP-512
Name: Evaluation - Dokumentation

Beschreibung: Als Produktbesitzer möchte ich eine ausführliche Dokumentation bezüglich der verschiedenen Evaluationskriterien.
Kriterien:

- Die Evaluation des System ist dokumentiert.

Bezeichner: LSOP-523
Name: Evaluation - Aktivitäten
Beschreibung: Als Entwickler möchte ich eine Evaluation über die Funktionalität und Leistungsfähigkeit des System durchführen. Zuerst wird LSOP-541 abgeschlossen.
Kriterien:

- Verschiedene Evaluationsszenarien sind durchgeführt.

Bezeichner: LSOP-541
Name: Evaluation - Vorbereitung
Beschreibung: Als Entwickler möchte ich die geplanten Evaluations-Aktivitäten aus LSOP-523 vorbereiten.
Kriterien:

- Verschiedene Evaluationsszenarien sind entworfen.

Bezeichner: LSOP-547 bis 558
Name: Abschlussdokomentation
Beschreibung: Als Produktbesitzer möchte eine ausführliche Dokumentation des Produkts. Jede Story behandelt dabei eines der Kapitel dieser Abschlussdokumentation. Die Kapitel können aus der Gliederung im Inhaltsverzeichnis entnommen werden.
Kriterien:

- Das jeweilige Kapitel ist vollständig beschrieben.

Keinem Epic zugeordnet

Bezeichner: LSOP-151
Name: Durchstich: Spielminute anzeigen
Beschreibung: Als Nutzer möchte ich die aktuelle Spielminute auf dem Tablet angezeigt bekommen.
Kriterien:

- Der Durchstich unserer vollständigen Kommunikation ist erreicht (von Sensordaten bis Android).

3.2.2 Nicht umgesetzte User Stories

Bezeichner: LSOP-25
Name: Kräftemodell für Load-Balancing integrieren
Beschreibung: Als System möchte ich in der Lage sein, Last von einem Peer zu einen optimalen anderen Peer abzugeben, um die Last möglichst optimal zu verteilen. Dazu möchte ich das Kräftemodell nutzen.
Kriterien:

- Die Kräftemodell Strategie ist implementiert, so dass ein Load-Balancing durchgeführt werden kann.

Begründung: Das Kräftemodell wird aus Kapazitätsgründen nicht mehr umgesetzt, da es nicht zwingend benötigt wird, um die Funktionstüchtigkeit des Load-Balancings zu zeigen.

Bezeichner: LSOP-26
Name: Vergleich der Load-Balancing-Strategien
Beschreibung: Als Produktbesitzer möchte ich nachlesen können, warum welche Load-Balancing Strategien integriert werden, um die Entscheidung nachvollziehen zu können.
Kriterien:

- Ein Vergleich der umgesetzten Load-Balancing Strategien ist in der Abschlussdokumentation nachzulesen.

Begründung: Da LSOP-25 nicht mehr umgesetzt wird, ist diese Story hinfällig.

Bezeichner: LSOP-28
Name: Anbindung der Entwickler-GUI an Odysseus P2P
Beschreibung: Als Entwickler möchte ich, dass die Developer Application auch mit Odysseus Instanzen kommunizieren kann, damit Datenströme angezeigt werden.
Kriterien:

- PQL Statements können an Odysseus Instanz angefragt werden.

Begründung: Wird von den Entwicklern nicht benötigt.

Bezeichner: LSOP-29
Name: Push-Fähigkeit der API ermöglichen
Beschreibung: Als System möchte ich, dass Daten vom Host zu den Client-Anwendungen gepusht werden und diese nicht aufwendig in kontinuierlichen Abständen abfragen müssen, sodass die Belastung für den Host möglichst minimal ist.
Kriterien:

- Daten werden vom Host zum Client gepusht.

Begründung: Ist durch die Verwendung von Sockets schon gegeben.

Bezeichner: LSOP-30
Name: Developer Application kommuniziert mit Zwischensprache
Beschreibung: Als System möchte ich, dass die Developer Application mit dem Host über die Zwischensprache kommunizieren kann, sodass sie keine PQL oder CQL Syntax verwenden muss und die Generierung der Anfragen beim Host durchgeführt wird.
Kriterien:

- Abruf aller möglichen Keywords vom Host möglich und Darstellung auf der GUI.
- Senden der Keywords an den Host und Erstellung der passenden PQL / CQL Statements.

Begründung: Ist bereits in LSOP-27 umgesetzt.

Bezeichner: LSOP-35
Name: Verschiedene Perspektiven in der Coach App
Beschreibung: Als Nutzer möchte ich verschiedene Perspektiven auswählen können, sodass Ansichten, Einstellungen und eventuell relevanten Daten automatisch für mich vorbelegt werden.
Begründung: Das Konzept der Coach App ist überarbeitet worden, wodurch diese Story hinfällig ist.

Bezeichner: LSOP-37
Name: JXTA Operatoren verschieben
Beschreibung: Als System möchte ich in Odysseus P2P die JXTA Operatoren auf andere Peers verschieben können, sodass die Belastung auf andere Peers zur Laufzeit verlagert werden kann.
Begründung: Duplikat.

Bezeichner: LSOP-42
Name: Testaufbau der Sensoren
Beschreibung: Als Entwickler wollen wir die Sensoren zwecks Test aufbauen, sodass wir eventuelle Probleme im Vorfeld erkennen können.
Kriterien:

- Probeaufbau ist durchgeführt.
- Aufbau wurde inklusive Fotos dokumentiert.

Begründung: Duplikat.

Bezeichner: LSOP-43
Name: Eigene Spieldaten mit Sensoren ermitteln
Beschreibung: Als Entwickler möchten wir eigene Spieldaten mit den angeschafften Sensoren aufnehmen können, sodass wir auch andere Sportarten oder spezielle Gegebenheiten analysieren können.
Begründung: Duplikat.

Bezeichner: LSOP-44
Name: Evaluierung der Genauigkeit der Sensorik
Beschreibung: Als Entwickler möchten wir evaluieren, wie genau die GPSender App in der Realität (GPS/WLAN) wirklich ist, sodass wir das in unsere Auswertungen eventuell berücksichtigen können.
Begründung: Aus Kapazitätsgründen wegen des hohen Aufwands nicht umgesetzt.

Bezeichner: LSOP-113
Name: Netzwerkunabhängigkeit
Beschreibung: Als System möchte ich, dass ein Peer in einem beliebigen Netzwerk sein und trotzdem am Peer-Netzwerk teilnehmen kann.
Begründung: Ist bereits vorhanden.

Bezeichner: LSOP-183
Name: Intermediate Schema erstellen
Beschreibung: Als System möchte ich das Intermediate Schema verwenden. Dazu: Intermediate Schema erstellen und die vorhanden Queries anpassen.
Begründung: Ist in LSOP-208 umgesetzt.

Bezeichner: LSOP-189
Name: Verhalten des Clients bzgl. des zuständigen Peers
Beschreibung: Als Nutzer möchte ich Informationen über Peerausfälle und weitere Ereignisse im Netzwerk erhalten.
Begründung: Story ist verworfen.

Bezeichner: LSOP-206
Name: Optimierung der vorhandenen Anfragen
Beschreibung: Die vorhandenen Anfragen müssen vor allem bezüglich der Nutzung von Fenstern, sowie möglichst frühzeitig Selektion und Projektion optimiert werden.

Begründung: Aus Kapazitätsgründen ist auf eine ausführliche Anfrageoptimierung verzichtet worden, da dies nicht der Schwerpunkt unserer Arbeit ist.

Bezeichner: LSOP-211
Name: Upstream-Backup Konzepte Implementieren
Beschreibung: Als System möchte ich die Funktionalität besitzen, Query-Parts bearbeiten zu lassen und so lange Tupel Upstream vorzuhalten, bis diese nicht mehr benötigt werden. So dass bei Ausfällen der Infrastruktur (Peer Ausfällen) eine fehlerfreie Bearbeitung meiner Anfragen wieder erreicht werden kann.
Kriterien:

- Neuer Backup-Peer wird gefunden und initialisiert.
- Zwischenspeichern von Tupeln Upstream ist möglich.
- Löschen von Tupeln aus dem Speicher, wenn sie fertig verarbeitet sind, ist möglich.
- Vorgehen ist dokumentiert. Dazu ist die Dokumentation von LSOP-209 zu erweitern.

Begründung: Aus Kapazitätsgründen ist die Arbeit auf die Verwendung des Active-Standbys beschränkt. Dieses bietet hinreichende Ausfallsicherheit.

Bezeichner: LSOP-212
Name: Koordination von Active-Standby und Upstream-Backup
Beschreibung: Als System möchte ich die passendste Recovery-Technik für ein gegebenes Szenario wählen können, damit ein Trade-Off zwischen Bearbeitungsaufwand und Wiederherstellungsgeschwindigkeit gelingt.
Kriterien:

- Kombination von Active-Standby und Upstream-Backup ist umgesetzt. Es wird jeweils die günstigste Strategie gewählt.(Operatorenliste)
- Vorgehen ist dokumentiert. Dazu ist die Dokumentation von LSOP-209, LSOP-210 und LSOP-211 zu erweitern.

Begründung: Da LSOP-211 nicht umgesetzt wird, kann LSOP-212 auch nicht umgesetzt werden.

Bezeichner: LSOP-245
Name: Developer Application erweitern und Bedienung verbessern
Beschreibung: Als Entwickler möchte ich die Developer Application besser benutzen können, damit die Verwendung schneller und einfacher möglich ist.
Begründung: Die Verwendung der Developer Application wird nicht mehr fortgeführt, daher ist die Story hinfällig.

Bezeichner: LSOP-285
Name: Erweiterung der Ketten in der Coach App
Beschreibung: Als Nutzer möchte ich Statistiken über meine erstellten Ketten in der Coach-App angezeigt bekommen.
Begründung: Diese Story hätte das dynamische Erstellen von Anfragen zu von Nutzern erstellten Ketten erfordert. Der Aufwand ist im Verhältnis zum Ertrag zu hoch.

Bezeichner: LSOP-384
Name: Verteilungsstrategie optimieren
Beschreibung: Es kann sein, dass eine spezielle Verteilungsstrategie (u.a. für das Recovery) Vorteile gegenüber den bisherigen Strategien hat. Hier soll eine solche Strategie erdacht und erstellt werden.

Begründung: Aus Kapazitätsgründen wird diese Story nicht mehr berücksichtigt, da es nicht der Schwerpunkt der Arbeit ist.

Bezeichner: LSOP-433
Name: DDC Einträge über die Coach App anpassen
Beschreibung: Als Coach oder Techniker möchte ich die Möglichkeit haben die DDC Einträge, wie Spielfeldabmessungen oder Sensor-ID's direkt über die Einstellungen der Coach App anpassen zu können, sodass ich hierfür nicht extra die Textdatei in Odysseus öffnen muss.
Begründung: Wird nicht mehr benötigt.

3.3 Zusammenfassung

Die Anforderungen wurden zunächst mit dem Product Owner definiert und im Laufe des Projekts angepasst und erweitert. So ergeben sich fachliche Anforderungen für die Bereiche Load-Balancing, Recovery, Kommunikation zwischen Coach App und Odysseus P2P, Coach App, Sensorik, Monitoring Application und Anfragen. Die einzelnen Anforderungen sind mit Prioritäten von gering bis hoch bewertet. Dadurch wird eine Reihenfolge bei der Umsetzung der Anforderungen implizit vorgegeben.

Auf Basis der ermittelten Anforderungen wurden im Detail User Stories beschrieben und einem Epic zugeordnet. Die folgenden Epics haben sich auf Grund der Anforderungen und im Laufe des Projekts ergeben: Load-Balancing, Recovery, Kommunikation Client zu P2P, Sensorik, Schnittstelle Sensorik, Data-Mining auf Sensordaten, Coach App, Monitoring Application, Developer Application, Projektdokumentation.

4 Systemarchitektur

Innerhalb dieses Kapitels werden die Architektur und die Bestandteile von Herakles erläutert. Dazu gehört zunächst eine Übersicht des gesamten Systems. Anschließend wird auf die einzelnen Bestandteile von Herakles genauer eingegangen. Dies umfasst die Sensorik, das P2P-Netzwerk, die Datenabstraktion, Odysseus, SportsQL, die mobile App und die Monitoring Application.

4.1 Systemübersicht

In diesem Abschnitt wird eine grobe Übersicht über das System Herakles gegeben, um im Anschluss auf die einzelnen Komponenten genauer einzugehen. Die Abbildung 4.1 zeigt die drei Bestandteile, in die Herakles eingeteilt werden kann: Die Sensorik, die Datenanalyse im P2P-Netzwerk und das Handheld, auf dem die Statistiken visualisiert werden.

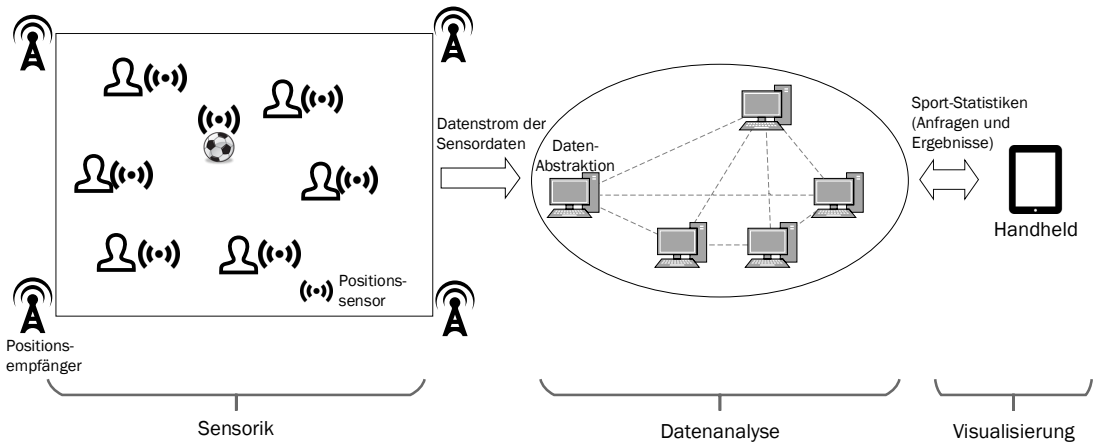


Abbildung 4.1: Systemübersicht von Herakles

Von dem Sensorsystem aus fließen die Positionsdaten der Spielobjekte, wie etwa die der Spieler und des Balls, an das P2P-Netzwerk. Mindestens ein Peer aus diesem Netzwerk muss den Datenstrom der Sensoren empfangen. Innerhalb dieses Netzwerkes werden die Daten analysiert und Statistiken berechnet. Die fertig berechneten Statistiken werden anschließend an das Handheld gesendet, das die Statistiken visualisiert und dem Nutzer Funktionen bietet, um mit diesen zu interagieren.

Die gesamte Verarbeitung der Daten im P2P-Netzwerk basiert auf dem DSMS Odysseus und der Erweiterung Odysseus P2P. Da P2P-Netzwerke, wie in 2.10 beschrieben, einige Herausforderungen mit sich bringen, muss Odysseus P2P um Mechanismen zur Lastverteilung und zum Umgang mit Peer-Ausfällen erweitert werden. Das Load-Balancing sorgt für eine ausgeglichene Last auf den Peers im Netzwerk, um die Überlastung einzelner Peers zu vermeiden. Das Recovery schaltet sich im Falle des Ausfalls einzelner Peers ein, um möglichst nahtlos weiter Ergebnisse zum Handheld liefern zu können.

In Abbildung 4.1 ist zudem eine Datenabstraktion eingezeichnet. Diese ist notwendig, um Herakles von spezifischen Sensorsystemen unabhängig zu machen, sodass die Sensorik leicht ausgetauscht werden kann.

Da die Peers auch Geräte ohne grafische Oberfläche sein können, wie etwa ein Netzwerk von Raspberry Pis, bietet Herakles eine Monitoring Application an, die in einem Browser aufgerufen werden kann und mit der die Peers des Netzwerkes überwacht und gesteuert werden können.

4.2 Sensorik

Um die Position der Spielelemente, wie etwa die Spieler und der Ball, feststellen zu können, wird ein Sensorsystem benötigt, das regelmäßig die Position der Spielobjekte misst. Diese Systeme werden „Real-Time Location Systems“ (RTLS) genannt (siehe 2.2). In Frage kommen zum Beispiel Systeme, die auf der RFID-Technik basieren oder GPS-Signale nutzen. Für Herakles ist es wichtig, dass die verwendeten Sensoren ihre Positionsdaten aktiv versenden können, sodass das P2P-Netzwerk in der Lage ist, diese Daten zu analysieren. Außerdem müssen sie klein und robust genug sein, um im Sport eingesetzt werden zu können.

Herakles bietet eine Android-Anwendung, die sogenannte „GPSender App“, welche die Position über den eingebauten GPS-Sensor eines Smartphones bestimmt und an einen Server sendet. Dieser sendet die Daten an mindestens einen Peer des P2P-Netzwerk weiter. Die Anwendung hat einige Nachteile, sodass sie nur für Testzwecke verwendet werden sollte. Beim Sport kann es bspw. hinderlich sein, ein Smartphone mit sich führen zu müssen. Außerdem sind die Positionen, welche durch die GPS-Sensoren ermittelt werden, nur auf wenige Meter genau. Des Weiteren ist das System nur unter freiem Himmel und nicht in einer Halle nutzbar. Für Testzwecke reicht die entwickelte Anwendung jedoch aus und liefert Daten, die genau genug sind, um auf diesen aufbauend Statistiken zu berechnen.

4.3 P2P-Netzwerk

Die große Menge an aufgezeichneten Positionsdaten soll durch ein P2P-Netzwerk verarbeitet werden. Gegenüber einer monolithischen Lösung mit einem großen leistungsstarken Server kann das P2P-Netzwerk aus mehreren kostengünstigen Rechnern (z.B. Raspberry Pis) bestehen. Darüber hinaus wird hierdurch eine höhere Verfügbarkeit und Skalierbarkeit erreicht.

Aus der Verwendung eines P2P-Netzwerkes entstehen unterschiedliche Herausforderungen. Zum Einen kann durch die Autonomie des Netzwerkes ein Peer jederzeit das Netzwerk verlassen. Daher ist ein Recovery-Mechanismus notwendig, um einen ausfallenden Peer zu kompensieren und eine Neuverteilung der verloren gegangenen Query-Parts auf andere Peers zu ermöglichen. Dadurch soll eine möglichst nahtlos Verarbeitung der Daten gewährleistet werden. Zum Anderen kann die Berechnungslast des Netzwerkes unterschiedlich stark auf die verschiedenen Peers aufgeteilt werden. Durch die Heterogenität des Netzwerkes kann dieser Effekt noch verstärkt werden, wenn ein Peer mit geringen Ressourcen einen großen Teil der Berechnungen durchführen muss.

Zur Verarbeitung der Daten wird auf jedem Peer eine Odysseus-Instanz ausgeführt. Das sehr anpassbare und generische DSMS bietet die Möglichkeit, Datenströme mit bereits vorhandenen Operatoren (Fenster, Selektionen etc.) zu verarbeiten. Von Odysseus wird dabei nicht die monolithische

Standardversion verwendet, sondern die Erweiterung Odysseus P2P, mit der die Möglichkeit geboten wird, Datenströme verteilt in P2P-Netzwerken zu verarbeiten.

4.4 Datenabstraktion

Da es nicht das perfekte Sensorsystem gibt, das für Herakles genutzt werden kann, ist das System so aufgebaut, dass es unabhängig von spezifischen Sensoren ist. Um diese Unabhängigkeit zu erreichen, bietet das System eine Datenabstraktionsschicht, welche Sensordaten in ein einheitliches Zwischenschema überführt. Auf diesem Zwischenschema basieren die Anfragen für die einzelnen Sportereignisse. Durch das Zwischenschema kann das Sensorsystem ausgetauscht werden, ohne dass die Anfragen angepasst werden müssen. Lediglich die Schnittstelle zwischen Sensor-Rohdaten und dem Zwischenschema muss im Falle eines neuen Sensorsystems angepasst werden. Das erleichtert es enorm, Herakles mit unterschiedlichen Sensoren zu nutzen, da ein Großteil der Arbeit in die Entwicklung der Anfragen für Sportereignisse geflossen ist und diese nun nicht immer neu entwickelt werden müssen.

4.5 SportsQL

In der Übersichtsabbildung 4.1 ist zu erkennen, dass das Handheld bei dem P2P-Netzwerk aktiv Statistiken anfordert. Um möglichst wenig Logik auf dem Handheld implementieren zu müssen und einen größtmöglichen Anteil der Logik im P2P-Netzwerk zu haben, wird die domänenspezifische Anfragesprache SportsQL verwendet. Mit dieser simpel gehaltenen Anfragesprache müssen auf dem Tablet keine komplexen Anfragen, z.B. in der Anfragesprache PQL, generiert werden. Die SportsQL-Anfragen werden stattdessen in dem P2P-Netzwerk zu einem, abhängig von der Anfrage, komplexen Anfrageplan übersetzt.

4.6 Mobile App

Die Anzeige der fertig berechneten Statistiken findet auf einem Handheld statt. Herakles bietet dazu eine App für Android-Geräte an, die auf die Größe von Tablets optimiert ist. Das Ziel ist es, dass ein Trainer die Statistiken direkt am Rand des Spielfeldes nutzen kann. Darüber hinaus bietet die App zahlreiche weitere Möglichkeiten. Dies umfasst z.B. das Anzeigen von Ketten oder das Aufmalen von Taktik-Anweisungen.

4.7 Monitoring Appliciation

Die Monitoring Application ist eine Webanwendung, mit der eine Überwachung und Steuerung der Peers möglich ist. Sie bietet bspw. die Möglichkeit, den Graphen einer verteilten Anfrage zu visualisieren. Darüber hinaus können die Peers über eine Konsole gesteuert und über eine Log-Ausgabe beobachtet werden.

4.8 Zusammenfassung

In diesem Kapitel wurde die Systemarchitektur von Herakles vorgestellt. Sie basiert im Wesentlichen auf den Komponenten Sensorik, Datenanalyse und Visualisierung. Die Sensorik ist dafür zuständig, die Positionen von Spielobjekten (Spieler, Bälle etc.) zu messen und diese in einem Datenstrom an mindestens ein Odysseus-Peer zu senden. Innerhalb der Datenanalyse verarbeiten mehrere Odysseus-Peers innerhalb eines P2P-Netzwerkes den Datenstrom, um sportarten-spezifische Ereignisse (z.B. Torschüsse) in den Daten zu erkennen. Dabei spielen Load Balancing und Recovery eine wichtige Rolle, um überlastete Peers oder Ausfälle von Peers zu berücksichtigen. In der Visualisierungskomponente werden die erkannten Ereignisse in den Daten als Statistiken innerhalb einer mobilen App auf einem Handheld angezeigt. Des Weiteren lassen sich mit einer Monitoring Application die Odysseus-Peers von einem Netzwerk steuern und überwachen.

5 Konzept

Im Folgenden werden die Konzepte für die einzelnen Teilbereiche des Projektes beschrieben. Dabei wird zunächst mit der Ermittlung von Sensordaten begonnen. Basierend auf den Daten wird die Datenabstraktion und die Analyse auf dem Datenstrom genauer erläutert. Weiterhin werden zentrale Aspekte wie Load-Balancing und Recovery in Bezug auf das P2P-Netzwerk vorgestellt. Abgeschlossen wird das Konzept mit den beiden Client-Anwendungen für die Darstellung der Statistik-Ergebnisse und die Überwachung des P2P-Netzwerkes.

5.1 Echtzeitlokalisation

Zur Lokalisierung der Spieler und anderer Spielobjekte musste eine Lösung gefunden werden, die die Positionsdaten genau genug ermittelt, um daraus Statistiken berechnen zu können. Zudem müssen diese Positionsdaten direkt weitergesendet werden, damit die Analyse live stattfinden kann. Letztlich müssen die verwendeten Sensoren zudem robust genug sein, um während eines Spiels getragen werden zu können.

Bei der Suche nach Lösungen für genaue Positionsbestimmung von Spielern werden besonders häufig Real Time Kinematic (RKT) Systeme genannt. Mit solchen Systemen werden Fehler, die in den Signalen von Navigationssatelliten, wie etwa denen von GPS, auftreten, gemessen und bereinigt. Somit sind Genauigkeiten im einstelligen Zentimeter-Bereich möglich, was im Vergleich zu einer Genauigkeit von fünf Metern eines gewöhnlichen GPS-Empfängers sehr gut ist. Dazu ist ein zweiter Empfänger in der Nähe des eigentlichen Positionsempfängers notwendig, der Korrekturdaten erstellt und an den (oder die) eigentlichen Positionsempfänger sendet. Der Empfänger, der die Korrekturdaten sendet, ist die Referenzstation; der mobile, sich bewegende Empfänger wird *Rover* genannt.

Da die Systeme jedoch verhältnismäßig teuer sind und in den gewöhnlichen Bauformen für die Verwendung im Sport ungeeignet sind, wurde darauf verzichtet, ein solches System zu nutzen und stattdessen auf die bereits in Abschnitt 2.2 beschriebenen Möglichkeiten zur Positionsbestimmung zurückgegriffen. Aufbauend auf diesen Methoden wird eine App für Android-Smartphones geschrieben, die die Positionsdaten an einen Server sendet, der als Quelle in Odysseus eingebunden wird. Die Smartphones ermitteln ihre Position über den eingebauten Positionssensor, der meistens mit GPS-Signalen arbeitet.

5.1.1 Lokalisierung via GPS

Um GPS-Positionsdaten im Spielfeld der Coach App darstellen zu können, muss zunächst ein Spielfeld mittels GPS abgezeichnet werden. Dazu werden die Eckpunkte des Spielfeldes ermittelt. Problem ist hierbei allerdings, dass das Spielfeld nicht unbedingt in der Richtung aufgespannt wird, in der die Längen- bzw. Breitengrade verlaufen. Um dieses Problem zu umgehen, sollte zunächst auf das Gauß-Krüger (GK) Koordinatensystem zurückgegriffen werden. Hierbei können einzelne Koordinaten mittels Rechts- und Obenwert dargestellt werden. Maßeinheit ist in diesem Koordinatensystem Meter und GPS Koordinaten lassen sich mittels einer Java Bibliothek in GK Koordinaten umwandeln. Theoretisch wäre diese Umrechnung für unseren Anwendungszweck die beste Variante, da wir keine zusätzlichen Berechnungen durchführen müssen. Das Problem ist allerdings auch hier, ähnlich

wie bei der Nutzung der rohen GPS Daten, dass die Spielfelder nicht individuell gestaltet werden können, sondern nach dem Koordinatensystem entsprechend ausgerichtet sein müssen. Aus diesen Gründen wurde auf die Verwendung dieser Verfahren verzichtet.

Eine weitere Variante ist die Nutzung eines lokalen Koordinatensystems, in die die rohen GPS Koordinaten umgewandelt werden, um mit diesen weiterarbeiten zu können. Nach der Umwandlung ist man unabhängig von den ursprünglichen GPS- oder GK Koordinaten. Das lokale System wird mittels der Abstände zwischen den GPS Koordinaten aufgespannt, wie in der Abbildung 5.1 zu sehen ist.

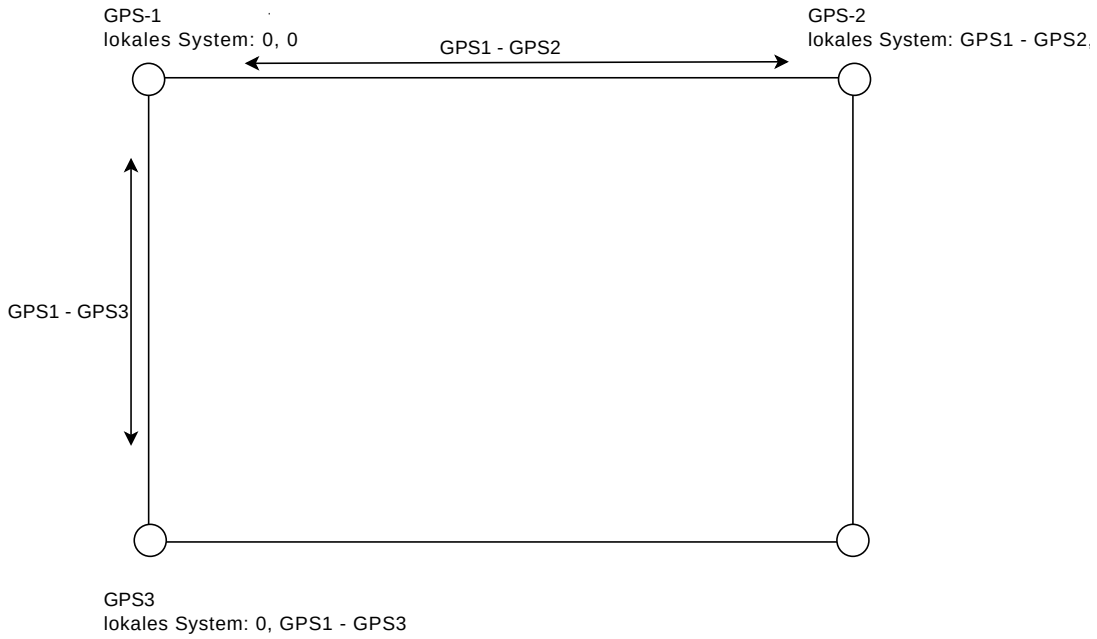


Abbildung 5.1: Bestimmung des lokalen Koordinatensystems

Um GPS-Positionsdaten im lokalen System darzustellen, ist es nun notwendig, die Abstände von mindestens drei Eckpunkten (GPS Koordinaten) zu einem gesuchten Punkt (einer GPS Koordinate) innerhalb des Spielfeldes zu bestimmen. Diese Abstände können wiederum auf die Eckkoordinaten des lokalen Systems angewendet werden, um Kreise mit den entsprechenden Radien zu bestimmen. Mittels Triangulation (siehe Abschnitt 2.2.0.1) wird dann der gesuchte Punkt im lokalen Koordinatensystem gefunden (siehe Abbildung 5.2). Dieses Vorgehen wird in der Anwendung zur Positionsbestimmung der Projektgruppe genutzt.

5.1.2 Lokalisation via WLAN

Ähnlich wie die Lokalisierung via GPS müssen zur Bestimmung der Position mindestens drei Eckpunkte des Spielfelds bekannt sein. Hier werden Accesspoints aufgebaut und die Abstände zu den jeweils anderen Accesspoints gemessen. Die Abstände können wiederum auf das lokale System übertragen werden. Zur Bestimmung einer Position wird das Verfahren „Free-space path loss (FSPL)“ genutzt.

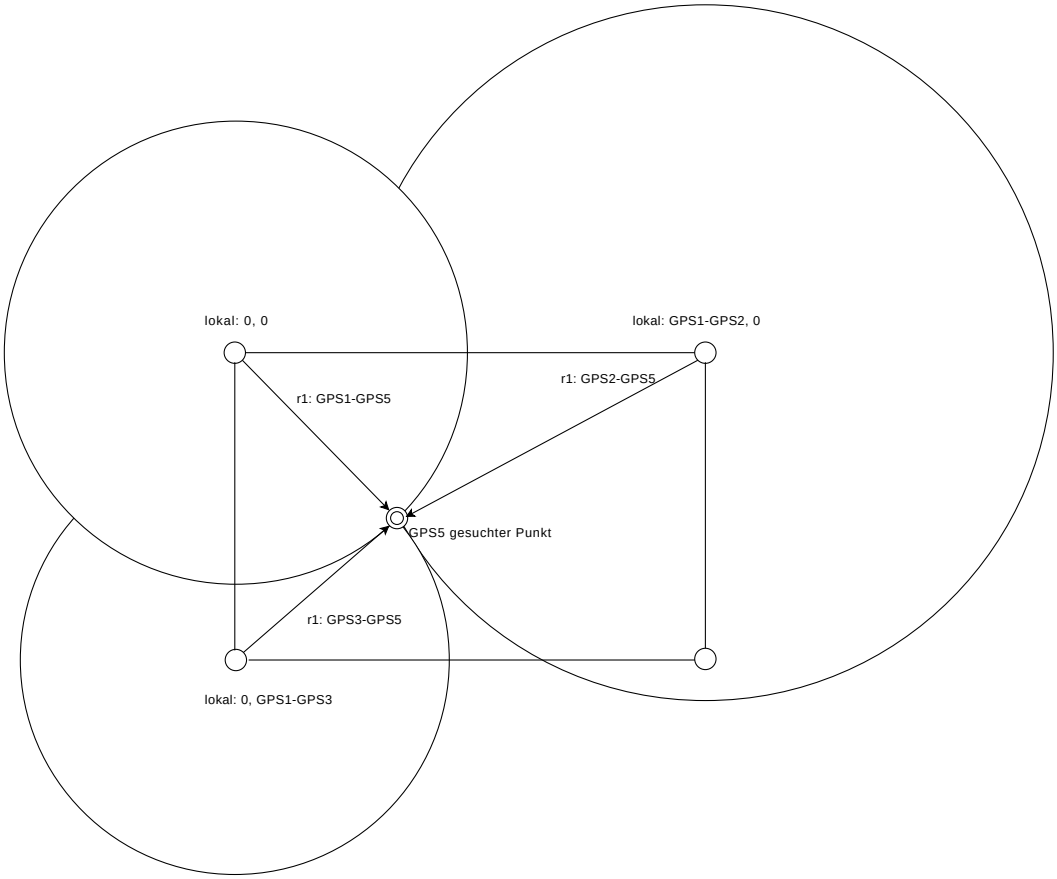


Abbildung 5.2: Bestimmung Position im lokalen Koordinatensystem

Das FSPL beschreibt hierbei den Verlust der Signalstärke einer elektromagnetischen Welle durch den freien Raum. Hierbei wird davon ausgegangen, dass sich in dem Raum keine Objekte oder sonstige Hindernisse befinden, die die Ausbreitung der elektromagnetischen Wellen beeinflussen. Je weiter der gesuchte Punkt von einem Accesspoint entfernt ist, desto schwächer ist das Signal. Hierbei kann mittels einer passenden Formel die Entfernung in Metern berechnet und somit der genaue Punkt innerhalb des lokalen Koordinatensystems bestimmt werden (siehe 2.2.1.3). Dazu wurde die „Free-space path loss“-Formel verwendet: $FSPL(db) = 20log_{10}(d) + 20log_{10}(f) - 27,55$. Die Konstante ist geeignet für die Verwendung der Einheiten Meter und Megahertz.

5.2 Datenabstraktion

Dieser Abschnitt beschreibt unser Konzept für eine Abstraktion der Daten. Die Datenabstraktion stellt sicher, dass Herakles unabhängig von den Datenquellen eingesetzt werden kann. Die abstrahierten Daten werden verwendet, um Sportanalysen auf diesen auszuführen.

Bei der Datenabstraktion werden Heterogenitäten von Sensoren berücksichtigt und bereinigt. Konkret können sich die Sensor-Rohdaten in vielen Merkmalen von System zu System voneinander unterscheiden. So ist es möglich, dass unterschiedliche Maßeinheiten für die Koordinaten genutzt werden.

Einige Sensoren liefern zudem noch weitere Informationen, wie Beschleunigung und Geschwindigkeit, andere tun dies nicht. Mit der Datenabstraktion wird sichergestellt, dass die Daten unabhängig von ihrer Quelle verarbeitet werden können. Abbildung 5.3 zeigt die Architektur für eine solche Abstraktion der Daten.

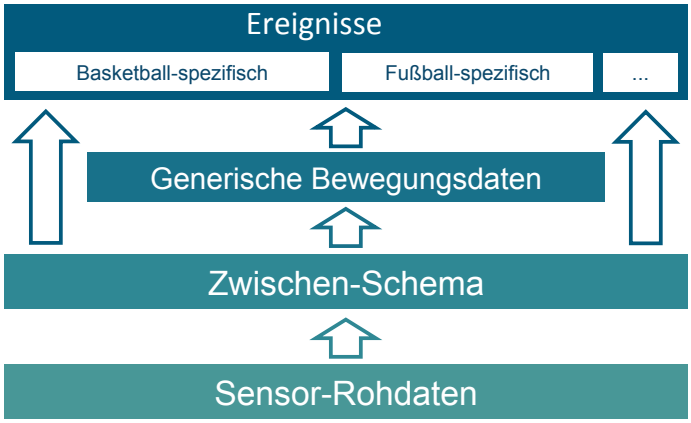


Abbildung 5.3: Architektur Datenabstraktion

5.2.1 Zwischenschema

Zunächst werden Sensor-Rohdaten in ein einheitliches Zwischenschema transformiert. Um eine möglichst große Anzahl an Sensorsystemen zu unterstützen, werden Geschwindigkeit und Beschleunigung von Herakles selbst auf Basis der Positionsdaten berechnet. Die Sensor-Rohdaten müssen lediglich die Sensor-ID, einen Zeitstempel und die Positionsdaten enthalten. Aus der Sensor-ID kann mit Hilfe des *Distributed Data Container (DDC)* (vgl. Abschnitt 5.2.2) die Entity-ID und Team-ID bestimmt werden. Anschließend können generische Bewegungsdaten, beispielsweise Geschwindigkeit und Beschleunigung, aus den Daten des Zwischenschemas berechnet werden. Das mit den Bewegungsdaten erweiterte Zwischenschema ist in Tabelle 5.1 dargestellt. Die Anfragen verwenden nun die Daten aus dem Zwischenschema und die generischen Bewegungsdaten, um sportartenspezifischen Ereignisse, wie z.B. Torschüsse, in den Daten zu erkennen.

5.2.2 Distributed Data Container

Eine einfache Speicherung von Konstanten auf einem Peer ist in einem verteilten P2P-Netzwerk nicht sinnvoll, da ein oder mehrere Peers jederzeit das Netzwerk verlassen können und im Anschluss kein Zugriff mehr auf die Daten besteht. Darüber hinaus wäre ständiger Zugriff über das Netzwerk vergleichsweise langsam. Aus diesem Grund müssen die Konstanten auf alle Peers des Netzwerkes verteilt werden. Dies stellt jedoch eine weitere Herausforderung dar, da die Daten zu jeder Zeit konsistent gehalten werden müssen. In diesem Abschnitt wird das Konzept des DDC erläutert und speziell auf die Verteilung und Synchronisation der Daten eingegangen.

Die im DDC hinterlegten Werte sollen nach einem Herunterfahren des Peers persistent in einer Datei gespeichert werden, sodass diese beim erneuten Starten des Peers wieder verfügbar sind. Diese

Name	Datentyp	Einheit
Entity-ID	Integer	-
Team-ID	Integer	-
ts	Starttimestamp	BaseTimeUnit (BTU)
x	Double	mm
y	Double	mm
z	Double	mm
v	Double	microm/s
a	Double	microm/s^2

Tabelle 5.1: Zwischenschema inkl. generischen Bewegungsdaten

Speicherung soll allerdings konfigurierbar sein. Beim Hochfahren des Peers soll die entsprechende Datei eingelesen werden und in den Arbeitsspeicher geladen werden. Wurde der Peer gestartet, werden Änderungen am DDC nicht mehr in die Datei geschrieben. Diese wird erst wieder modifiziert, wenn der Peer heruntergefahren wird. Wird ein Peer gestartet, der keine hinterlegte DDC-Datei hat, wird automatisch eine leere Datei erstellt.

Wie bereits erwähnt, müssen die Werte des DDC zwischen den verschiedenen Peers ausgetauscht werden. Hierzu ist es notwendig, dass diese durch ein festgelegtes Protokoll eine Synchronisation durchführen. Beim Hochfahren des Peers sollen alle Inhalte geladen und anschließend initial an alle im Netzwerk verfügbaren Peers weitergeleitet werden. Diese prüfen anhand des Zeitstempels, ob diese in das eigene DDC übernommen werden und schicken automatisch auch die eigene Inhalte an den neu hinzugekommenen Peer, damit auch dieser alle bereits im Netzwerk verfügbaren Inhalte erhält. Da bei einem solchen Vorgang mehrere Peers antworten können, verarbeitet der neue Peer nur eine dieser Antworten. Auf diese Weise kann sichergestellt werden, dass die Peers sich automatisch bei einem Neueintritt in das Netzwerk synchronisieren. Generell ist das DDC so konzipiert, dass Änderungen eher selten auftreten und der Synchronisationsvorgang somit eine Zeit von mehreren Sekunden benötigen darf. Darüber hinaus werden auch eventuelle Synchronisationsprobleme nicht beachtet, da in der Regel Änderungen zu einer Zeit nur von einem Peer durchgeführt werden. Kommt es zur Laufzeit zu Änderungen auf einem Peer werden alle anderen Peers automatisch darüber informiert und diese nehmen die Änderungen in den eigenen DDC auf.

5.3 Datenanalyse

In diesem Abschnitt werden die konzeptionellen Ideen für die Erstellung der Sportanfragen beschrieben. Dabei handelt es sich, vor allem wegen der zur Verfügung stehenden Daten, ausschließlich um Fußball-Anfragen. Alle Anfragen setzen auf dem in Abschnitt 5.2 beschriebenen Zwischenschema auf, damit auf einer einheitlichen Datengrundlage gearbeitet werden kann. Für eine dynamische Anpassung der Anfragen an das jeweilige Spiel oder die jeweilige Sportart verwendet eine Vielzahl der Anfragen das DDC. Im Folgenden werden die einzelnen Anfragen kurz vorgestellt.

5.3.1 Analysezeit-Anfrage

Die Analysezeit-Anfrage gibt Aufschluss darüber, seit wievielen Minuten und Sekunden die Datenanalyse dauert. Ursprünglich war diese Anfrage als Spielzeit-Anfrage gedacht. Da Herakles jedoch keine Spielunterbrechungen erkennen kann, wird mit dieser Anfrage die Analysezeit anstatt der Spielzeit berechnet.

5.3.2 Ballkontakte

Mit dieser Anfrage werden die Ballkontakte pro Spieler und Team berechnet. Ein Ballkontakt liegt dann vor, wenn sich der Ball in einem Radius von 40 cm um einen Spieler befindet. Falls der Ball sich gleichzeitig in den Radien mehrerer Spieler befindet, wird der Ballkontakt dem Spieler mit der niedrigsten Distanz zugeordnet.

5.3.3 Ecken

Die Ecken-Anfrage erkennt eine Ecke, sobald der Ball ins Toraus gespielt wird und sich anschließend innerhalb der nächsten 30 Sekunden in einem Radius von 1,5m um eine Eckfahne befindet.

5.3.4 Heatmap

Eine Heatmap dient zur Visualisierug der häufigsten Positionen eines Spielers. Dazu wird die Spielfeldübersicht eingefärbt. Um eine Heatmap erstellen zu können, wird das Spielfeld in Kacheln unterteilt. Die Anzahl der Kacheln vertikal und horizontal kann bei der Anfrage bestimmt werden. Wird das Feld z.B. in 30 x 15 Kacheln unterteilt, wird für jedes Spielobjekt, also unter anderem für jeden Spieler, ein Integer-Array dieser Größe erstellt.

Anschließend wird bei jedem eingehenden Positions-Tupel berechnet, in welcher Kachel sich diese Position befindet. Der Zähler für diese Kachel wird in dem entsprechenden Integer-Array des Spielobjekts um eins erhöht. So entsteht für jeden Spieler ein Array, in dem gespeichert ist, wie oft der Spieler wo war. Um die Einfärbung anhand der relativen Verhältnisse vornehmen zu können, wird zudem immer der maximale Wert gespeichert, der pro Spieler in dem jeweiligen Array vorhanden ist. Ein solches Array bei einer 2x2-Einteilung könnte so aussehen wie in Tabelle 5.2. Der Spieler war vor allem in der oberen linken Kachel häufig unterwegs. Der maximale Wert für diesen Spieler ist 5.

5	1
3	0

Tabelle 5.2: Heatmap-Array eines Spielers

5.3.5 Laufstrecke

Diese Anfrage ermittelt die Laufstrecke eines Spielers und eines Teams. Dazu werden jeweils die Abstände zwischen zwei aufeinander folgenden Positionen eines Spielers berechnet und zu einer Gesamtlaufstrecke aggregiert.

5.3.6 Spielerpositionen

Die Spielerpositionen-Anfrage verändert die Daten aus dem Zwischenschema nicht und leitet sie direkt weiter.

5.3.7 Pässe

Mit dieser Anfrage werden die Pässe pro Spieler und Team berechnet. Ein erfolgreicher Pass liegt dann vor, wenn zwei aufeinander folgende Ballkontakte zu unterschiedlichen Spielern des selben Teams gehören. Wenn zwei aufeinander folgende Ballkontakte zu unterschiedlichen Teams gehören, wurde ein Fehlpass gespielt. Ein Doppelpass liegt dann vor, wenn bei zwei aufeinander folgenden Pässen der Passgeber des ersten Passes gleichzeitig der Passempfänger des zweiten Passes ist. Ein Pass wird direkt gespielt, wenn nach Empfangen eines Passes der Ball innerhalb von einer Sekunde wieder weitergepasst wird. Ein kurz gespielter Pass hat eine Maximallänge von 10 Metern, ein lang gespielter Pass hat eine Länge von über 10 Metern. Zudem werden Pässe anhand des Passwinkels darin unterschieden, ob diese vorwärts, quer oder rückwärts gespielt werden.

5.3.8 Sprints

Die Sprints-Anfrage berechnet die Anzahl an Sprints pro Spieler und Team. Ein Sprint eines Spielers beginnt, wenn die Geschwindigkeit größer ist als 24 Km/h. Der Sprint endet, sobald die Geschwindigkeit in der Folge darauf unter 19 Km/h sinkt. Neben der Anzahl an Sprints kann zudem die durchschnittliche Länge und Geschwindigkeit der Sprints sowie die Maximalgeschwindigkeit berechnet werden.

5.3.9 Tore

Die Tor-Anfrage ermittelt dann ein Tor, wenn der Ball die Torlinie (innerhalb der Pfosten und unterhalb der Querlatte) passiert und anschließend innerhalb von 30 Sekunden in einem festgelegten Radius um den Mittelpunkt des Spielfeldes gelegt wird.

5.3.10 Torschüsse

Die Bewegung eines Balles muss ein bestimmtes Muster erfüllen. Zum einen muss er eine Beschleunigung von mindestens $55m/s^2$ aufweisen. Wenn ein Sensorwert dieses Kriterium erfüllt, kann parallel zu den folgenden Kriterien der Spieler, der zu dem Zeitpunkt am nächsten zum Ball ist, bestimmt werden. Dieser Spieler gilt als Schütze.

Ab dem Zeitpunkt, zu dem die Beschleunigung gemessen wurde, sind die nächsten 1,5 Sekunden und der nächste Meter in der Flugbahn entscheidend. Nur, wenn der Ball innerhalb dieser Zeitspanne mindestens einen Meter zurück legt, kann das Muster als Torschuss in Frage kommen.

Wenn die Sensordaten des Balles die beschriebenen Kriterien erfüllt und der Ball anschließend die Torlinie überquert, wird die Aktion sowohl als Torschuss als auch als Tor gewertet. Da es allerdings auch Torschüsse gibt, die nicht zu einem Tor führen, müssen auch abgelenkte Schüsse, die dadurch nicht aufs Tor gehen, erfasst werden. Dies wird durch eine Prognose der Flugbahn des Balles bewerkstelligt. Es wird in diesem Zusammenhang allerdings eine gerade Flugbahn vorausgesetzt.

5.4 SportsQL und Sportartenunabhängigkeit

In diesem Abschnitt wird das Konzept der Anfragesprache SportsQL erläutert. Dabei wird zunächst motiviert, warum eine neue Anfragesprache entwickelt wurde und welche Vorteile diese bietet. Weiterhin wird der Aufbau und die Erweiterbarkeit der Sprache beschrieben.

5.4.1 Motivation

Damit Statistiken bei einem Client angezeigt werden können, ist es notwendig, dass diese Anwendungen zunächst eine entsprechende Anfrage in Odysseus installieren. Die Erstellung dieser Anfragen kann je nach Statistik sehr kompliziert und aufwendig sein. Darüber hinaus werden eventuell odysseusspezifische Informationen benötigt, um die Anfrage zu erstellen. Aus diesem Grund wurde eine neue, domänenspezifische Anfragesprache entwickelt, die es ermöglicht, dass ausschließlich der Statistiktyp und eine minimale Anzahl an zusätzlichen Informationen angegeben werden müssen. Dabei ist es entscheidend, dass nur definiert werden muss welche Statistik benötigt wird und nicht, wie diese technisch realisiert wird. Da durch die Anfragesprache Sportstatistiken angefragt werden können, wurde als Name „SportsQL“ gewählt.

SportsQL bietet den Vorteil, dass die Logik für die Erstellung der Anfrage vom Client in die Odysseus-Instanz verlagert wird. Der Client muss somit nicht wissen, wie die angeforderte Anfrage funktioniert. Darüber hinaus kann die Logik der Anfrage zentral in Odysseus angepasst werden, ohne dass sämtliche Client-Anwendungen modifiziert werden müssen.

5.4.2 Grundlegende Funktionsweise

Im Vergleich zur Anfragesprache PQL, die sehr technisch aufgebaut ist und mit der es möglich ist, individuelle Anfragen an einen Datenstrom zu stellen, ist SportsQL sehr simpel gehalten. Die simplen SportsQL Anfragen werden in Odysseus P2P ausgewertet und anschließend in einen deutlich komplizierteren Anfrageplan umgesetzt. Dieser Anfrageplan besteht aus Operatoren, die den Datenstrom entsprechend der jeweiligen Anfrage manipulieren.

Bei SportsQL ist es ausschließlich notwendig, den Statistiknamen, beispielsweise Torschüsse, die Sportart und den Statistiktyp anzugeben. Optional ist es möglich, einen individuellen Anzeigenamen zu vergeben, sodass die Anfragen in Odysseus P2P besser unterschieden werden können. Als Statistiktyp stehen folgende Möglichkeiten zur Auswahl:

```
{
    "statisticType": "player",
    "gameType": "soccer",
    "name": "ballpossession",
    "parameters": {
        "space": {
            "space": "all"
        },
        "time": {
            "time": "always"
        }
    }
}
```

Listing 5.1: Beispielanfrage in SportsQL (Ballbesitz)

global Wird der Statistiktyp global gewählt, bezieht sich die Anfrage auf das gesamte Spiel und somit auf beide Teams und alle Spieler. Das ist vor allem bei Statistiken sinnvoll, die unabhängig von Teams oder Spielern sind, wie beispielsweise die Analysezeit.

player Bei diesem Statsitiktyp wird die Statistik auf Spielerebene ermittelt. Es wird also für jeden Spieler individuell eine eigene Statistik erstellt. Ein mögliches Beispiel hierfür ist die zurückgelegte Strecke eines Spielers. Eine Angabe bzw. eine Filterung auf einen speziellen Spieler ist nicht möglich und muss in der jeweiligen Client-Anwendung vorgenommen werden.

team Analog zum Statistiktyp Spieler ist es auch möglich, Anfragen zu stellen, die sich auf das gesamte Team beziehen. Dabei werden alle Daten der einzelnen Spieler einer Mannschaft zusammengefasst. Dies ist vor allem bei Statistiken sinnvoll, die sich auf das gesamte Team beziehen wie der Ballbesitz im Fußball.

Abhängig von der jeweiligen Anfrage ist es auch möglich, dass Anfragen mehrere Statistiktypen unterstützen, da es eventuell sinnvoll sein kann, dass die Ergebnisse in unterschiedlicher Granularität an den Client gesendet werden.

Bei SportsQL handelt es sich um einen JavaScript Object Notation (JSON)-Dialekt. Dies hat zum Vorteil, dass es über spezielle Frameworks wie GSON einfach möglich ist valide Anfragen zu erstellen und auch wieder zu parsen. Listing 5.1 zeigt die beispielhafte Verwendung von SportsQL. Die aufgeführten Parameter sind dabei optional.

5.4.3 Zeit- und Raumparameter

Neben den beschriebenen verpflichtenden Angaben gibt es noch einige weitere Parameter, die angegeben werden können, um die Anfrage an die jeweiligen Bedürfnisse anzupassen.

Mit dem Zeitparameter "time"kann der gewünschte Zeitraum angegeben werden. Wird nichts angegeben, wird „GAME“ genutzt, was bedeutet, dass nur Informationen verarbeitet werden, die während der Spielzeit passieren. Sollen alle Informationen verarbeitet werden, muss der Parameter „ALWAYS“

```
{
"parameters": {
    "time": {
        "start": 5,
        "end": 10,
        "unit": "minutes"
    }
}
```

Listing 5.2: Beispielanfrage in SportsQL (Ballbesitz)

angegeben werden. Die festen Parameter in "time"haben Priorität vor den Start- und Endzeitpunkten. Wird also ein gültiger time-Konstanten-Parameter angegeben, werden die Start- und Endzeitpunkte überschrieben. Bei einer konkreten Angabe von Zeitwerten muss darüber hinaus noch die Zeiteinheit angegeben werden. Dabei sind folgende Zeiteinheiten verfügbar:

- minutes
- seconds
- milliseconds
- microseconds
- nanoseconds
- picoseconds

Listing 5.2 zeigt die Verwendung des Zeitparameters mit der Angabe von festen Zeitwerten und der dazugehörigen Einheit.

Mit dem Raumparameter kann der gewünschte Raum angegeben werden, also z.B. Teilbereiche des Spielfeldes. Wird nichts angegeben, wird „FIELD“ genutzt, was bedeutet, dass nur die Informationen verarbeitet werden, die innerhalb des Spielfeldes liegen. Sollen alle Informationen verarbeitet werden, muss „ALL“ angegeben werden. Die festen Parameter in „SPACE“ haben Priorität vor den Koordinatenangaben. Wird also ein gültiger space-Konstanten-Parameter angegeben, werden die Koordinatenangaben überschrieben. Abbildung 5.4 zeigt alle verfügbaren Raumkonstanten, die ausgewählt werden können.

5.4.3.1 Erweiterbarkeit

Damit die Umwandlung von SportsQL in eine Sprache, die Odysseus versteht flexibel und erweiterbar ist, ist für jede Statistik (z.B. die Spielerstatistik Ballbesitz) ein Parser vorgesehen, der in einen logischen Anfrageplan übersetzt. Mittels einer Annotation kann dynamisch der richtige Parser ausgewählt werden.

Es ist eine Annotation SportsQL vorgesehen, die folgende Informationen bereithält:

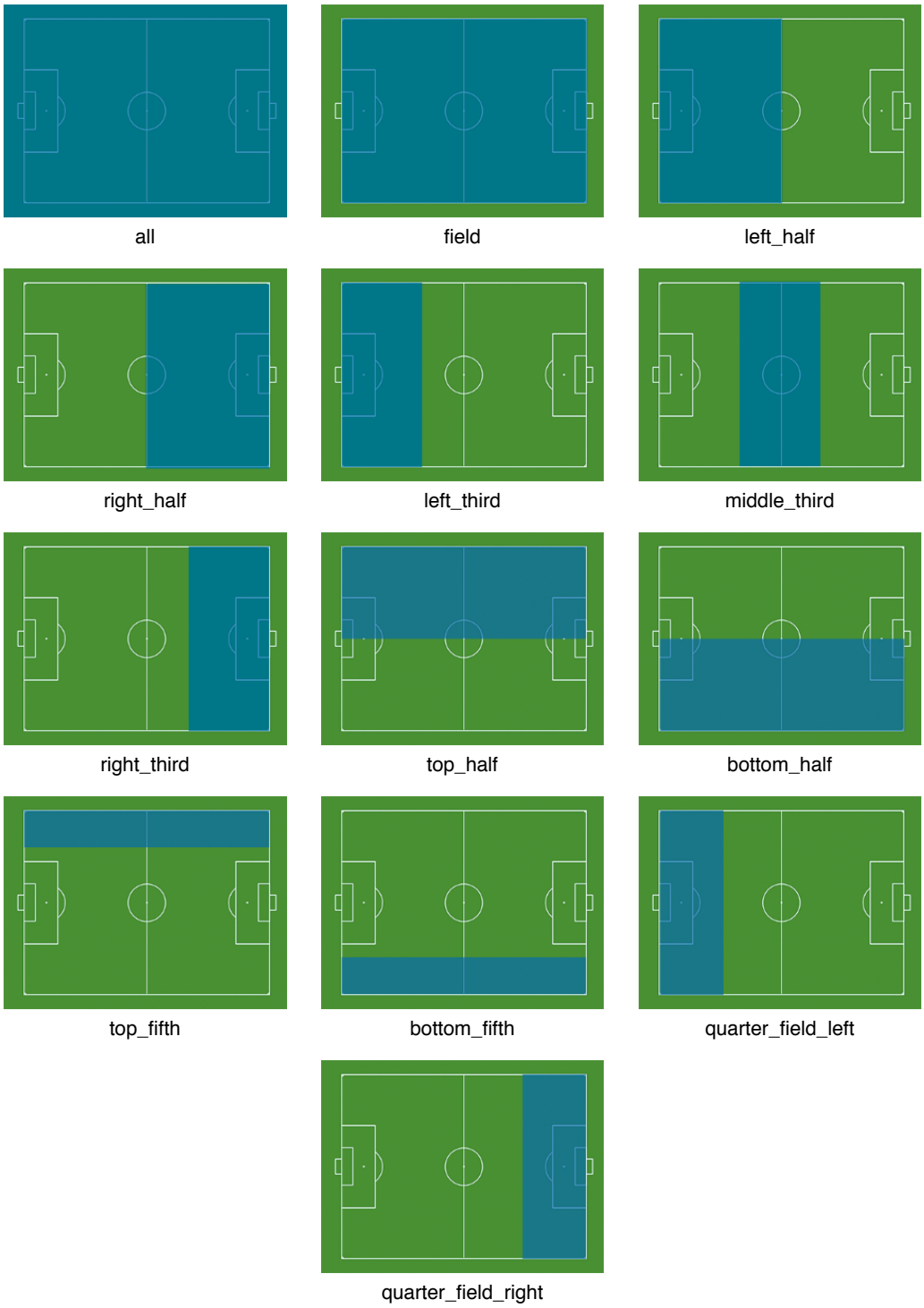


Abbildung 5.4: Mögliche Konstanten des Raumparameters in SportsQL

- Die Sportarten für die die konkrete Übersetzung geeignet ist (z.B. auch „all“ für generische Anfragen oder Parameter)
- Die Statistiktypen, denen die Statistik angehört
- Der Name der Statistik

Es ist ein Interface `ISportsQLParser` vorgesehen, das die Übersetzer implementieren.

Die Entität, die Anfragen vom Client entgegennimmt sollte alle eingebundenen Übersetzer verwalten. Somit kann sie über die Annotation bestimmen, welcher Übersetzer ausgewählt werden soll. Da Parameter bei allen Anfragen auftauchen können und deren Übersetzung ein Teil der Gesamtübersetzung ist, sollten die Parameterübersetzer vor anderen Übersetzern aufgerufen und die Ergebnisse in die Anfrage eingebaut werden.

5.5 Load-Balancing

Im Folgenden wird das Konzept zum Load-Balancing vorgestellt. Dabei wird zunächst der grundlegende Aufbau erläutert, d.h. welche Schnittstellen zur Durchführung von Load-Balancing notwendig sind. Anschließend werden mit Parallel-Track und Moving-State zwei unterschiedliche Verfahren erklärt, mit denen ein Load-Balancing durchgeführt werden kann.

5.5.1 Grundlegender Aufbau

Um ein dynamisches Load-Balancing durchzuführen, müssen zunächst vier Fragen beantwortet werden, deren optimale Antwort sich je nach Anwendungsfall unterscheiden kann:

1. Wie wird das Load Balancing ausgelöst?
2. Wie wird bestimmt, welche Anfragen oder Teilanfragen abgegeben werden?
3. Wie wird der Peer ermittelt, der die abzugebenen Anteile übernimmt?
4. Wie findet die Übertragung statt, ohne dass Informationen verloren gehen?

Deshalb wurde ein modularer Aufbau gewählt, bei dem jeweils eine eigene Schnittstelle für die Beantwortung einer oder zweier dieser Fragen zuständig ist: Eine *Load-Balancing-Strategie* überwacht jeweils einen Peer und löst ggf. das Load-Balancing aus. Außerdem wählt die Strategie auch die zu übertragene Teilanfrage. Ein *Load-Balancing-Allokator* ermittelt daraufhin einen fremden Peer, der bereit ist, diese Teilanfrage aufzunehmen. Sind der übernehmende Peer und die zu übertragene Teilanfrage gefunden, übernimmt ein *Load-Balancing-Kommunikator* die Kommunikation zwischen diesen Peers und sorgt für die eigentliche Übertragung. Zusätzlich müssen die Kommunikatoren in der Regel auch die beteiligten Peers oberhalb und unterhalb der Teilanfrage anpassen. Diese Peers bezeichnet man als Upstream- bzw. Downstream-Peers.

Um Load-Balancing in Odysseus P2P überhaupt zu ermöglichen wurden eine einfache Strategie und ein einfacher Allokator entwickelt: Die Strategie löst aus, wenn die Prozessorlast auf dem überwachten Peer über einem bestimmten Schwellwert liegt. Dieser kann je nach Bedarf gewählt werden.

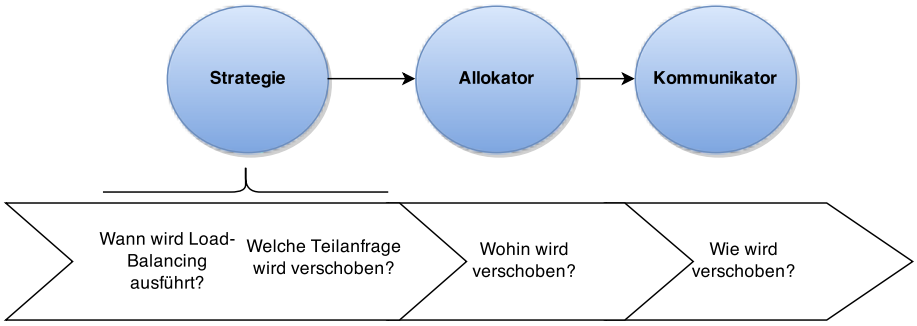


Abbildung 5.5: Aufbau Load-Balancing

Als zu verschiebende Teilanfrage wird immer die erste auf dem jeweiligen Peer installierte Anfrage gewählt. Der entwickelte Allokator sucht einfach per Round-Robin-Verfahren den nächsten bekannten Peer und schlägt diesen als Übertragungspartner vor. Da die eigentliche Übertragung von laufenden Teilanfragen zwischen Peers in Odysseus P2P bisher nicht möglich war, wurde der Schwerpunkt auf die Implementierung von Übertragungsmethoden und den dazugehörigen Kommunikatoren gelegt: Das *Parallel-Track-Verfahren* ermöglicht die Übertragung einer Teilanfrage ohne zustandsbehaftete Operatoren und mit dem *Moving-State-Verfahren* können zusätzlich auch Teilanfragen mit zustandsbehafteten Operatoren verschoben werden [ZRH04]. Beide Verfahren werden im Folgenden kurz vorgestellt.

5.5.2 Parallel-Track-Verfahren

Das Parallel-Track-Verfahren eignet sich zur Übertragung von Teilanfragen, die keinen zustandsbehafteten Operator enthalten [ZRH04]. Bei diesem Verfahren wird die auf dem Ursprungspeer (Master) laufende Anfrage verdoppelt und auf dem übernehmenden Peer (Slave) installiert. Ein nachgelagerter Synchronisations-Operator empfängt beide Datenströme, leitet aber zunächst nur den Ursprungsstrom weiter. Erst nachdem der neue Datenstrom den alten eingeholt hat, schaltet der Synchronisationsoperator auf den neuen Datenstrom um und das LoadBalancing ist beendet. Der gesamte Vorgang lässt sich in mehrere Phasen unterteilen:

5.5.2.1 Init-Phase

In der Init-Phase wird der vom Allokator ausgewählten Slave initialisiert. Kommt es während der Initialisierung des Slaves zu einem Fehler, schickt diese eine Fehlernachricht an den initiierenden Master zurück. Ist die Initialisierung erfolgreich, wird dem Master dies mit einem Init-Response (ACK) mitgeteilt. Nur wenn der Master vom Slave eine ACK-Nachricht erhält kann der Load-Balancing Vorgang fortgesetzt werden. Andernfalls, oder falls der Slave auch nach mehrmaligen Versuchen gar nicht reagiert, wird eine Fehlerbehandlung durchgeführt. Diese Phase dient vor allem der Vorbereitung des Master- und Slave-Peers auf die spätere Übertragung. Abbildung 5.6 zeigt den Zustand der Peers nach Abschluss der Init-Phase.

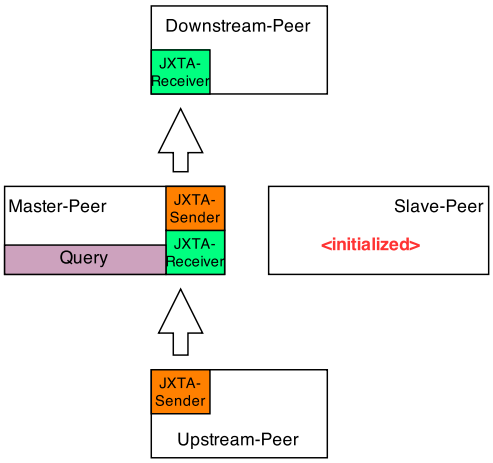


Abbildung 5.6: Parallel-Track: Zustand nach Init-Phase

5.5.2.2 Copy-Phase

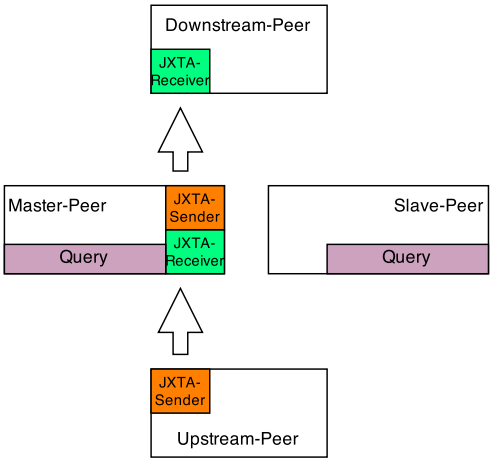


Abbildung 5.7: Parallel-Track: Zustand nach Copy-Phase

In der Copy-Phase soll die Anfrage vom Master auf den Slave übertragen werden. Dazu wird aus der Anfrage PQL-Code erzeugt und zum Slave gesandt. Der Slave installiert daraufhin das empfangene PQL und teilt dem Master mit, ob die Installation der Anfrage erfolgreich gewesen ist. Es kann vorkommen, dass der Peer nicht die Möglichkeit besitzt eine solche Anfrage zu installieren, da alle Peers in einem P2P Netzwerk heterogen sein können. Für diesen Fall würde der Slave dies dem Master mitteilen und der Load-Balancing Vorgang müsste mit einem anderen Slave von vorne gestartet werden. Dieser Fehlerfall ist stark von der „Intelligenz“ des Allokators abhängig: Bei einem Round-Robin Verfahren ist es sehr wahrscheinlich, dass ein solcher Fall eintritt. Handelt es sich jedoch um

einen Allokator, der die Möglichkeiten und Performance der einzelnen Peers zunächst beurteilt, kann der Fehlerfall nahezu ausgeschlossen werden. Nachdem die Anfrage erfolgreich auf dem Peer installiert wurde und der Master die Instruction-Response (ACK) erhalten hat, kann die nächste Phase des Protokolls gestartet werden. Das ist notwendig, da der Slave nun eine Anfrage installiert hat, allerdings wird der Datenstrom noch nicht an diesen Peer weitergeleitet. Abbildung 5.7 zeigt den Zustand der Peers nach Abschluss der Copy-Phase.

5.5.2.3 Relink-Phase

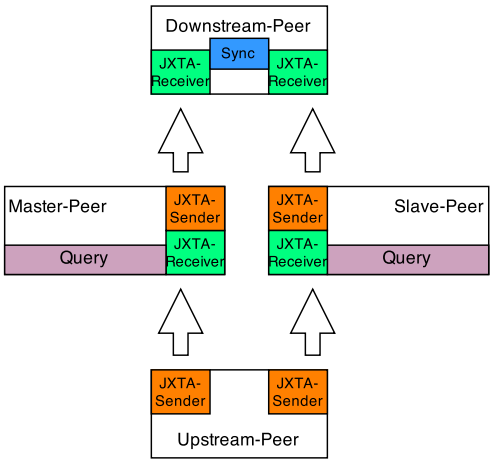


Abbildung 5.8: Parallel-Track: Zustand nach Relink-Phase

In der Relink-Phase wird der Datenstrom aufgeteilt und parallel sowohl weiterhin an den Master, als auch an den neuen Slave weitergeleitet. Dazu wird der Datenstrom beim Upstream-Peer aufgeteilt und durch das Einfügen eines neuen Senders an beide Peers weitergeleitet. Beim Downstream-Peer ist die Zusammenführung der Datenströmen etwas aufwändiger. Zum einen muss hier ein neuer Receiver eingefügt werden, der den Datenstrom des Slaves empfängt. Zum anderen muss ein Synchronisations-Operator eingesetzt werden, der die beiden ankommenden Datenströme synchronisiert. Dies ist notwendig, damit die Ergebnisse der beiden Peers nicht doppelt weitergeleitet werden, da diese eine gewisse Zeit identische Anfragen auf dem gleichen Datenstrom ausführen. Der Synchronisations-Operator verwirft dabei Ergebnisse, die bereits weitergeleitet wurden. Auch die Relink-Phase wird durch den Master initiiert, indem Instructions an den oder die Downstream- und Upstream-Peers gesandt werden. Diese Phase ist erst abgeschlossen, wenn alle Peers, die Anweisungen mit einem Relink-Response (ACK) bestätigt haben und somit die Datenströme auf beiden Peers gleichzeitig verarbeitet werden. Tritt ein Fehler auf oder ist ein beteiligter Peer nicht mehr erreichbar, wird der komplette Load-Balancing-Vorgang abgebrochen. Nachdem die Datenströme parallelisiert wurden, beginnt die Waiting-for-Sync-Phase. Abbildung 5.8 zeigt den Zustand der Peers nach Abschluss der Relink-Phase.

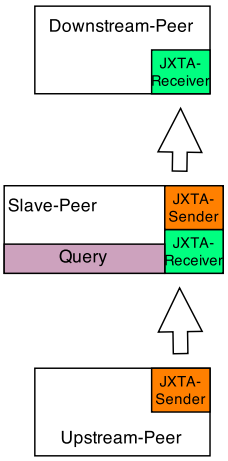


Abbildung 5.9: Parallel-Track: Zustand nach Waiting-for-Sync-Phase

5.5.2.4 Waiting-for-Sync-Phase

In dieser Phase wird zunächst darauf gewartet, dass die ankommenden Datenströme der beiden Peers (Master/Slave) identisch sind. Ist dies der Fall, wird die Verarbeitung des Datenstoms auf dem Master beendet. Dazu schaltet der Synchronisations-Operator zunächst komplett auf den neuen Datenstrom um. Dann wird beim Upstream-Peer die Aufteilung des Datenstroms aufgehoben und der alte Sender entfernt. Anschließend wird beim Downstream-Peer der alte Receiver und der Synchronisations-Operator entfernt. Dazu muss die Verarbeitung während des Entfernens kurzzeitig unterbrochen werden. Damit es hierbei nicht zu Datenverlust kommt, werden hier Buffer eingesetzt. Abschließend muss noch die Anfrage vom Master entfernt werden. Die Verarbeitung des Datenstroms wurde erfolgreich auf den Slave-Peer übertragen. Wie auch bei den vorherigen Phasen wird das Entfernen der Sender und Receiver und der Anfrage durch Anweisungen des Masters ausgelöst bzw. direkt auf dem eigenen Peer durchgeführt. Kommt es in dieser Phase zu Problemen beim Upstream oder Downstream Peers wird keine Fehlerbehandlung mehr durchgeführt, da ja bereits der neue Datenstrom erfolgreich aufgebaut wurde. Nach dem Entfernen der Operatoren, ist grundsätzlich wieder ein stabiler Zustand erreicht. Abbildung 5.9 zeigt den Zustand der Peers nach Abschluss der Waiting-for-Sync-Phase und somit nach Abschluss des Load-Balancing.

5.5.3 Moving-State-Verfahren

Im Gegensatz zum Parallel-Track-Verfahren eignet sich das Moving-State-Verfahren auch zur Verschiebung von zustandsbehafteten Operatoren [ZRH04]. Dazu wird die zu übertragende Teilanfrage zunächst kopiert und auf dem Slave-Peer installiert. Danach der Datentstrom vor dem betroffenen Teil der Anfrage angehalten. Dann werden die internen Status von allen zustandsbehafteten Operatoren vom Master auf den Slave übertragen, bevor der Datenstrom wieder gestartet und über den Slave-Peer geleitet wird. Die einzelnen Phasen dieses Verfahrens werden im Folgenden kurz beschrieben.

5.5.3.1 Init-Phase

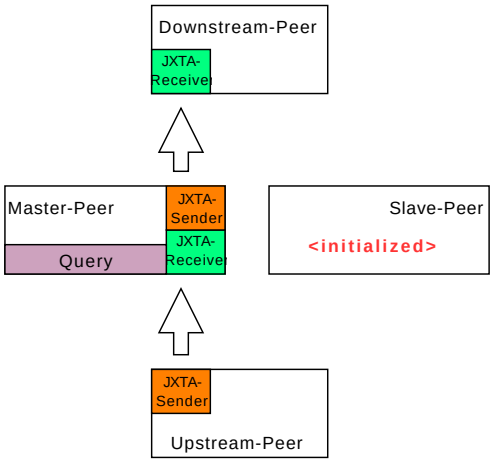


Abbildung 5.10: Moving-State: Zustand nach Init-Phase

Die Init-Phase beim Moving-State-Verfahren entspricht der Init-Phase beim Parallel-Track-Verfahren: In der Init-Phase wird der vom Allokator ausgewählten Slave initialisiert. Kommt es während der Initialisierung des Slaves zu einem Fehler, schickt diese eine Fehlernachricht an den initiierenden Master zurück. Ist die Initialisierung erfolgreich, wird dem Master dies mit einem Init-Response (ACK) mitgeteilt. Nur wenn der Master vom Slave eine ACK-Nachricht erhält kann der Vorgang fortgesetzt werden. Andernfalls, oder falls der Slave auch nach mehrmaligen Versuchen gar nicht reagiert, wird eine Fehlerbehandlung durchgeführt. Diese Phase dient vor allem der Vorbereitung des Master- und Slave-Peers auf die spätere Übertragung. Abbildung 5.10 zeigt den Zustand der Peers nach Abschluss der Init-Phase.

5.5.3.2 CopyQuery-Phase

Die Copy-Phase beim Moving-State-Verfahren entspricht ebenfalls der Copy-Phase beim Parallel-Track-Verfahren: In der Copy-Phase soll die Anfrage vom Master auf den Slave übertragen werden. Dazu wird aus der Anfrage PQL-Code erzeugt und zum Slave gesandt. Der Slave installiert daraufhin das empfangene PQL und teilt dem Master mit, ob die Installation der Anfrage erfolgreich gewesen ist. Auch hier sendet der Slave im Fehlerfall eine Nachricht an den Master. Nachdem die Anfrage erfolgreich auf dem Peer installiert wurde und der Master die Instruction-Response (ACK) erhalten hat, kann die nächste Phase des Protokolls gestartet werden. Das ist notwendig, da der Slave nun eine Anfrage installiert hat, allerdings wird der Datenstrom noch nicht an diesen Peer weitergeleitet. Abbildung 5.11 zeigt den Zustand der Peers nach Abschluss der Copy-Phase.

5.5.3.3 Buffer-And-Relink-Phase

In der Buffer-And-Relink-Phase wird zunächst der Datenstrom angehalten. Dazu sendet der Master an alle Upstream-Peers eine entsprechende Instruktion. Diese pausieren dann alle Datenströme, die Teil der entsprechenden Anfrage sind und speichern eingehende Tupel in einem Buffer. Danach werden alle betroffenen Sender durch neue Sender ausgetauscht, die den Datenstrom über den Slave statt

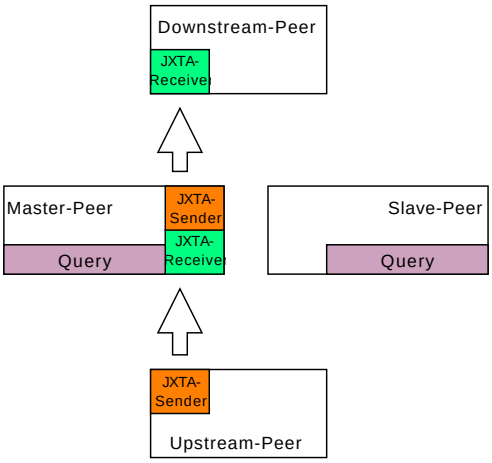


Abbildung 5.11: Moving-State: Zustand nach CopyQuery-Phase

über den Master leiten. Im Erfolgsfall bestätigen die Upstream-Peers dies, ansonsten melden sie dem Master einen Fehler. Ist kein Fehler aufgetreten, kann der Master weiter fortfahren. Er sendet nun eine Instruktion an alle Downstream-Peers, die nun analog die Receiver ersetzen. Ist dies erfolgreich, ist die Buffer-And-Relink-Phase erfolgreich abgeschlossen: Der Datenstrom ist noch pausiert, die Verbindungen laufen aber bereits über den neuen Peer. Diese Situation ist in Abbildung 5.12 dargestellt.

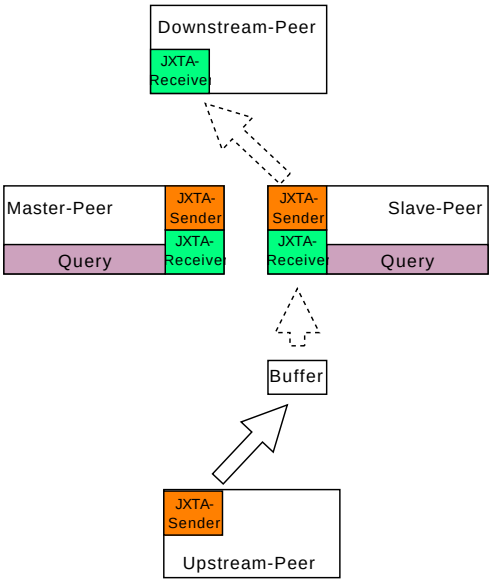


Abbildung 5.12: Moving-State: Zustand nach Buffer-And-Relink-Phase

5.5.3.4 Move-State-Phase

In der Move-State-Phase findet der eigentliche Zustandstransfer statt. Sie ist in Abbildung 5.13 dargestellt. Der Master erstellt eine Liste aller zustandsbehafteten Operatoren (diese sind durch ein spe-

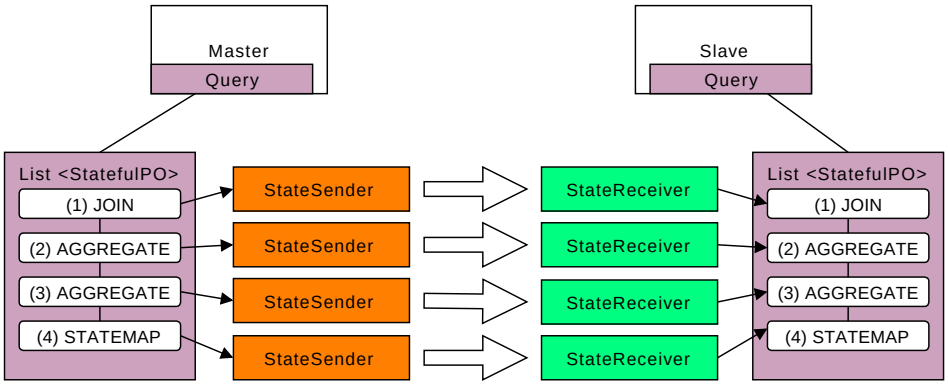


Abbildung 5.13: Moving-State: Move-State-Phase

zielles Marker-Interface gekennzeichnet). Für jeden dieser Operatoren wird nun ein eigener Sender installiert und eine Initiate-State-Copy-Nachricht an den Slave gesendet. Diese enthält Informationen über den Übertragungskanal und den betroffenen Operator. Der Slave erzeugt nun mit der gleichen Methode ebenfalls eine Liste der zustandsbehafteten Operatoren der (in der CopyQuery-Phase installierten) Anfrage. Er vergleicht für jede eingehende Initiate-State-Copy-Nachricht den mitgesendeten Operatortyp mit dem Typ des lokalen Operators. Stimmen diese nicht überein, sind die Operatoren in einer anderen Reihenfolge und der Slave sendet eine Fehlernachricht an den Master. Stimmen die Typen überein, wird je Operator ein passender Receiver installiert und eine Bestätigung an den Master gesendet. Der Master wartet nun, bis alle Operatoren vom Slave bestätigt wurden. Daraufhin werden die einzelnen Zustände vom Master an den Slave gesendet. Der Slave empfängt die Zustände, installiert diese und quittert den erfolgreichen Empfang mit einer weiteren Nachricht an den Master. Sind alle Zustandsübertragungen erfolgreich abgeschlossen, schickt der Master eine All-State-Copies-Finished-Nachricht und nach einer erneuten Bestätigung des Slaves ist die Moving-State-Phase abgeschlossen. Nun sollten alle zustandsbehafteten Operatoren auf beiden Peers über den gleichen Zustand verfügen.

5.5.3.5 Resume-Phase

In der Resume-Phase wird eine Stop-Buffering-Nachricht an die Upstream-Peers geschickt, die den Datenstrom wieder anlaufen lassen (über den Slave). Auch hier senden die Peers entweder eine Erfolgs- oder Fehlermeldung. Erst wenn alle Upstream-Peers das Wiederaufnehmen der Verarbeitung gemeldet haben, löscht der Master Peer die alte (nun unnötige) Anfrage und beendet das Load-Balancing.

5.6 Recovery

Das Recovery ist dafür zuständig, beim Ausfall eines Peers dafür zu sorgen, dass die Verarbeitung der eingehenden Tupel durch die Anfragen fortgesetzt werden kann. Dabei ist ein erstrebenswertes Ziel, dass möglichst wenige bis gar keine Tupel verloren gehen und die Ergebnisse einer Anfrage trotz des Ausfalls richtig bleiben. Um dies zu erreichen gibt es unterschiedliche Strategien, die im Folgenden konzeptionell vorgestellt werden.

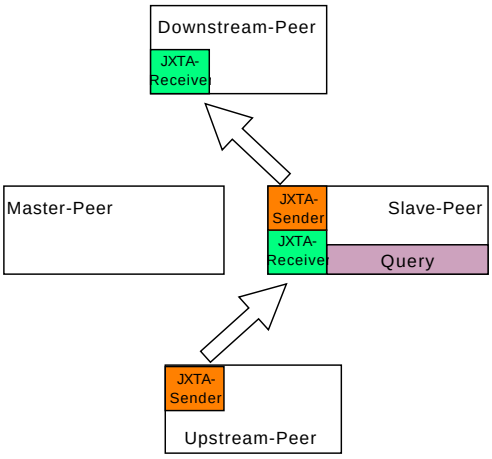


Abbildung 5.14: Moving-State: Zustand nach Resume-Phase

5.6.1 Ausgangssituation

Für das Recovery ist es wichtig, dass neben der gewöhnlichen Kommunikation in einem verteilten DSMS ein zusätzlicher Informationsaustausch stattfindet: Um bei einem Ausfall überhaupt reagieren zu können, tauschen die Peers untereinander durchgehend Informationen über ihren Zustand aus. So weiß im idealen Fall jeder Peer von jedem anderen Peer, welche Anfragen dieser gerade installiert hat. Erst so ist es möglich, dass nach dem Ausfall eines Peers entsprechend reagiert werden kann.

5.6.2 Phasen

Der Recovery-Prozess kann in mehrere aufeinander folgende Phasen eingeteilt werden, die in Abbildung 5.15 dargestellt sind. Die Phasen werden in den folgenden Punkten erläutert.

Ausfall erkennen

Zunächst einmal muss der Ausfall erkannt werden, damit die anderen Peers handeln können. Dazu müssen alle Peers im Netzwerk regelmäßig prüfen, ob die ihnen bekannten Peers noch erreichbar sind. Ist ein Peer nicht mehr erreichbar, wurde ein Ausfall erkannt. Dabei ist es möglich, dass ein Peer einen Ausfall deutlich früher erkennt als andere. Dies liegt an der dezentralen Natur eines P2P-Netzwerkes.

Um die Anzahl der durch den Ausfall verlorenen Tupel möglichst gering zu halten, werden bereits vor der Einigung, also der nächsten Phase, Nachrichten an die Peers gesendet, die Tupel an den ausgefallenen Peer gesendet haben. Diese Nachrichten informieren die Peers über den Ausfall, woraufhin diese die Tupel nicht weitersenden, sondern zwischenspeichern.

Einigung

In der zweiten Phase müssen sich die Peers im Netzwerk einigen, wer das Recovery für den ausgefallenen Peer übernimmt. Da es möglich ist, dass mehrere Peers den Ausfall gleichzeitig erkennen, müssen diese sich einigen, wer das Recovery durchführt. Melden sich mehrere Peers gleichzeitig,

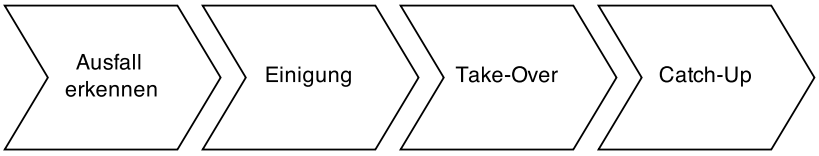


Abbildung 5.15: Phasen im Recovery

würden die mit dem ausgefallenen Peer weggefallenen Teilanfragen mehrfach wiederhergestellt werden, was es zu vermeiden gilt.

Um eine Einigung zu erreichen, kann jeder Peer, der das Recovery für einen ausgefallenen Peer übernehmen möchte, allen anderen Peers eine Nachricht schicken, dass er dies vorhat. Erhält er von keinem anderen Peer eine solche Nachricht, weiß er, dass er zu diesem Zeitpunkt der einzige ist, der das Recovery übernehmen möchte. Erhält er eine solche Nachricht von einem anderen Peer, müssen diese sich einigen. Dies kann z.B. dadurch geschehen, dass eine im Netzwerk für jeden Peer eindeutige ID verglichen wird und der Peer mit der höchsten (oder niedrigsten) ID den Zuschlag bekommt. Dabei ist zu betonen, dass der Peer, der das Recovery durchführt nicht derjenige ist, der die ausgefallenen Anfragen übernimmt, sondern die Installation dieser auf anderen Peers koordiniert. Es bieten zudem nur Peers an, das Recovery zu übernehmen, die Informationen über den ausgefallenen Peer haben.

Da ein Peer mehr als eine Teilanfrage ausführen kann und in einem P2P-Netzwerk nicht sichergestellt werden kann, dass jeder Peer alle Informationen über einen anderen Peer hat, wird die Aushandlung des Recoverys nach den Teilanfragen aufgeteilt. Hatte ein ausgefallener Peer zwei Teilanfragen installiert, so kann sich ein Peer aus dem Netzwerk darum kümmern, dass die erste Teilanfrage recovert wird und ein weiterer, dass die zweite recovert wird.

Take-Over

In der dritten Phase, dem Take-Over, werden die ausgefallenen Teilanfragen auf einem anderen Peer oder mehreren anderen Peers wieder installiert. Hierbei muss z.B. beachtet werden, dass nicht jeder Peer in der Lage sein muss, jede Teilanfrage zu übernehmen. In einem solchen Fall wird während der Installation der wiederherzustellenden Teilanfrage ein Fehler entstehen. Daraufhin muss der Peer, der das Recovery für diese Teilanfrage leitet einen neuen Peer auswählen, der die Teilanfrage übernimmt. Der Grund kann z.B. sein, dass ein Peer eine Quelle nicht hat, die notwendig ist, um eine Teilanfrage ausführen zu können.

In dieser Phase werden auch die Verbindungen wiederhergestellt. Das bedeutet, dass die Peers, die zu dem ausgefallenen Peer gesendet haben nun zu dem neuen Peer senden müssen. Mit dem Neuverbinden der Peers wird auch das Zwischenspeichern der Tupel beendet und die in der Zwischenzeit verarbeiteten Tupel werden weitergesendet.

Catch-Up

In der letzten Phase, dem Catch-Up, soll der Zustand, den der ausgefallene Peer hatte auf dem Peer, der die Teilanfrage übernommen hat wiederhergestellt werden. Außerdem werden eventuell speziell vorgehaltene Backup-Peers neu ausgewählt.

Je nach Recovery-Strategie können sich die Aktivitäten innerhalb der einzelnen Phasen unterscheiden. Der größte Unterschied besteht innerhalb der Catch-Up-Phase. Die einzelnen Strategien werden nun genauer beschrieben.

5.6.3 Strategien

Für die Vorgehensweise bei und vor dem Ausfall von Peers gibt es mehrere Ansätze, die jeweils eigene Vor- und Nachteile haben. Die Projektgruppe hat sich am ausführlichsten mit den Strategien „Amnesia“ und „Active-Standby“ beschäftigt.

5.6.3.1 Anmesia

Bei der Anmesia-Strategie werden die ausgefallenen Teilanfragen dynamisch an andere Peers verteilt, es ist also vorher kein Backup-Peer fest bestimmt worden. Bis auf die oben angesprochene Informationsverteilung findet also keine Vorbereitung statt. Diese Strategie ist eine reine Reaktion.

Die Amnesia-Strategie ist sowohl in fehlerfreier Ausführung, als auch im Fehlerfall am ressourcenschonendsten und somit aus Aspekten der Performance sehr günstig. Es muss kein extra Backup-Peer für das Recovery bereitgehalten werden. Stattdessen kann eine dynamische Zuweisung an einen Peer des Netzwerks (beispielsweise den mit der geringsten Auslastung) vorgenommen werden. Dieser ausgewählte Peer besitzt keine Informationen über den Zustand vor dem Ausfall des Knotens, denn ein Catch-Up findet nicht statt. Es wird lediglich die ausgefallene Teilanfrage installiert. Das bedeutet, dass alle Tupel zwischen dem Eintreten des Fehlers und der Fehlerentdeckung verloren sind. Zudem können auch bei zustandsbehafteten Operatoren wie Aggregationen oder Joins Probleme auftreten, da der neue Knoten die Zustände des ausgefallenen Knotens nicht reproduzieren kann und deshalb mit seinen Berechnungen bei Null anfängt.

5.6.3.2 Active-Standby

Diese Strategie setzt auf Replikation von Teilanfragen. Jede, oder eine bestimmte Auswahl von Teilanfragen wird auf zwei separaten Peers ausgeführt. D.h., beide Peers bearbeiten die eingehenden Tupel und senden sie weiter. Auf dem folgenden Peer gibt es einen Merger, der die Tupel beider Peers zusammenführt, dabei jedoch nur die Tupel von einem von beiden, nämlich die des Haupt-Peers, weiterverarbeitet. Die Verbindungen der Peers sind in Abbildung 5.16 skizziert. Die Tupel des Haupt-Peers werden weiterverarbeitet, die Tupel des zweites Peers bis zu dem Ausfall des Haupt-Peers verworfen.

Take-Over-Phase

Sollte der Haupt-Peer ausfallen, nutzt der folgende Peer automatisch die ankommenden Tupel des Backup-Peers. Da sowohl der Haupt-Peer, als auch der Backup-Peer die gleichen Tupel verarbeitet

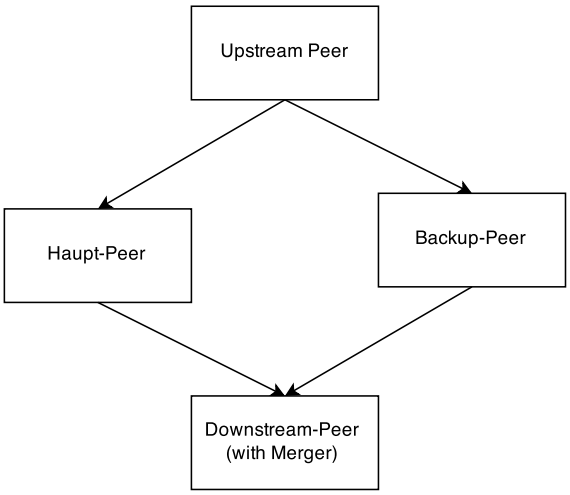


Abbildung 5.16: Active-Standby im Recovery mit einfacher Replikation

haben, ist der Ausgabestrom der gleiche. Es gehen also weder Tupel verloren, noch werden Ergebnisse verfälscht. Diese Strategie berücksichtigt also die Zustände von zustandsbehafteten Operatoren wie Joins.

Die zusätzliche Bearbeitungszeit ist durch die mindestens einfache, jedoch je nach Bedarf auch höhere Replikation recht hoch. Dafür kann der Backup-Peer im Fehlerfall direkt weiter die richtigen Tupel liefern, der Merger auf dem folgenden Peer muss lediglich auf den Backup-Peer umschalten, wenn er vom Haupt-Peer keine Daten mehr erhält.

Catch-Up-Phase

Für das Catch-Up muss ein neuer Backup-Peer bereitgestellt werden. Es wird also ein anderer Peer ausgewählt, der den ausgefallenen Peer ersetzt und zum neuen Backup-Peer wird, während der bisherige Backup-Peer nun der Haupt-Peer ist. Der neue Backup-Peer hat jedoch nicht den Zustand, den der Haupt-Peer hat. Dies kann man lösen, indem der Haupt-Peer seinen Zustand an den neuen Backup-Peer überträgt. Alternativ kann jedoch auf dieses Verfahren verzichtet werden, da der Backup-Peer ab nun die selben Tupel mitverarbeitet wie der Haupt-Peer und sich so dem richtigen Zustand annähert. Fällt der Haupt-Peer ein weiteres Mal aus, hat sich der Zustand entweder schon vollständig angeglichen, oder es muss mit (evtl. nur leicht) veränderten Ergebnissen gerechnet werden. In der Implementierung wurde die Zustandskopie aufgrund des signifikant höheren Implementierungsaufwands nicht genutzt.

5.6.4 Aufbau

In den obigen Abschnitten wurden einige austauschbare Strategien und Techniken vorgestellt, bei denen je nach Einsatzziel unterschiedliche Lösungen die besten sind. So kann z.B. nicht direkt entschieden werden, welche Strategie zur Auswahl eines neuen Peers die beste ist oder welche Recovery-

Strategie die geeignetste ist. Um diese einzelnen Komponenten austauschbar zu halten, ist ein modularer Aufbau von Vorteil.

Der Allokator ist dafür zuständig, einen Peer auszuwählen, der eine Teilanfrage übernehmen soll. Hier sind z.B. einfache Lösungen wie Round-Robin möglich. Es sind jedoch auch komplexere Allokatoren möglich, die die Last der einzelnen Peers mit einbeziehen. Wichtig ist, dass der Allokator einfach auszutauschen ist.

Das selbe gilt für die Strategie. Auch diese muss einfach austauschbar und erweiterbar sein, sodass sie an die Aufgabe angepasst werden kann. Benötigt man die maximale Performance und hat nur wenig Leistungsspielraum, so eignet sich die Amnesia-Strategie. Ist es vor allem wichtig, dass keine Informationen verloren gehen, ist Active-Standby die bessere Wahl. Eventuell möchte man jedoch auch für verteilte Anfragen mehrere Strategien verbinden. Zum Beispiel könnte man für zustandsbehaftete Operatoren Active Standby einsetzten, während für Anfragen, in denen keine zustandsbehafteten Operatoren vorkommen, nur Amnesia eingesetzt wird. Um dies zu ermöglichen, werden Strategien und Allokatoren von einem „Strategy-Manager“ aufgerufen. Dieser kann dann je nach Anfrage entscheiden, welche Strategie geeignet ist.

5.7 Mobile App

Im Folgenden wird das Konzept zur mobilen App vorgestellt. Hierbei wird zunächst der grundlegende Aufbau der App erläutert. Anschließend wird genauer auf die Kommunikation zwischen einzelnen Bestandteilen der App eingegangen. Des Weiteren wird die Paketstruktur vorgestellt und einige Mock-ups gezeigt.

5.7.1 Grundlegender Aufbau

Die Abbildung 5.17 zeigt das Konzept für den grundlegenden Aufbau der Mobile App. Kernelemente sind dabei *Activities* und *Fragments*. Dabei können einer Activity ein oder mehrere Fragments zugeordnet werden. Des Weiteren sollen Fragments auch in unterschiedliche Activities genutzt werden, um bereits bestehende Funktionalität nicht mehrmals implementieren zu müssen. Aus diesem Grund muss ein Fragment für sich atomar sein und darf weder von einer darunter liegenden Activity noch von anderen Fragments abhängig sein. Dadurch ergibt sich für Activities die Rolle als Container, um die verschiedene Fragments zu initialisieren und diese zu halten oder um grundlegende Initialisierungen von Attributen/ Einstellungen etc. vorzunehmen. Damit die Fragments unabhängig sind, dürfen keine Funktionen der Activity direkt verwendet werden. Allerdings sollen benötigte Funktionen nicht mehrfach in den Fragments implementiert werden, sondern stattdessen in Services ausgelagert werden. Dieser Aufbau hat folgende Vorteile:

- Funktionen können wiederverwendet werden und müssen nicht redundant implementiert werden
- Activities und Fragments enthalten keine grundlegende Logik, sondern sind ausschließlich für die Verwaltung und Steuerung von Nutzungsoberflächen (*Views*) zuständig.

Generell sollen alle Services in den Activities und Fragments über die Dependency Injection (DI) eingebunden und verwendet werden, um eine leichte Austauschbarkeit zu ermöglichen. Diese wird

in Activities und Fragments durch das Framework Roboguice[1] ermöglicht, indem zum einen die entsprechende Klasse mittels Vererbung von `RoboActivity` bzw. `RoboFragment` (nicht `RoboFragmentActivity`) erweitert wird und zum anderen die benötigten Services bei *Attribute Injection* angegeben werden. Bei Services gilt zunächst, dass diese möglichst zustandslos konstruiert werden sollten (keine Werte in globalen Variablen speichern), allerdings kann es hier durch die Verwendung von Frameworks auch Ausnahmen geben. Aus diesen Grund können Services auch als *Singleton* definiert werden.

Die von den Fragments und Activities verwendeten Services können auch weitere Services verwenden, auch wenn diese in der *LSOP Service*-Komponente liegen. Beispielsweise verwendet der *Service B* hier noch den *Service C* und den *LSOPService A*. Diese Abhängigkeiten sollen ebenfalls über die DI in die Services eingebunden werden. Allerdings soll hierbei die *Constructor Injection* verwendet werden, sodass alle Abhängigkeiten als Parameter im Konstruktor angegeben werden.

Die jeweiligen Services dürfen das Modell modifizieren, wohingegen Fragments oder Activities dies nicht direkt dürfen, sondern hierzu Services verwenden müssen. Allerdings ist es möglich, dass Fragments oder Activities über Änderungen am Modell informiert werden. Dies ist durch das *Observer Pattern* möglich.

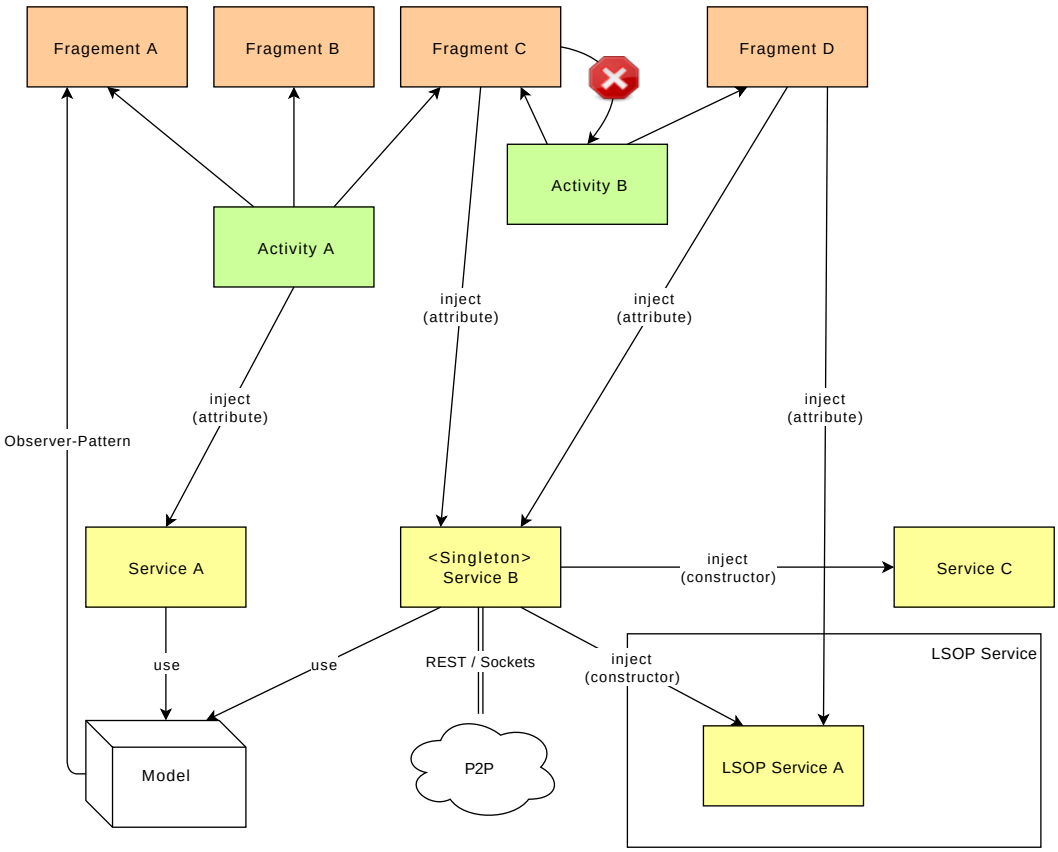


Abbildung 5.17: Konzept App-Architektur

[1] https://github.com/roboguice/roboguice/wiki

5.7.2 Kommunikation zwischen Fragments

Bei der Kommunikation der Fragments soll ebenfalls darauf geachtet werden, dass dies über Activities gelöst wird und nicht über eine direkte Referenz zwischen beiden Fragments. Hierdurch bleiben die Fragments weiterhin atomar. Die Abbildung 5.18 beschreibt die Grundlagen. Ein Fragment definiert ein entsprechendes Interface, das von der Activity implementiert wird. Daraufhin wird eine Methode des anderen Fragments aufgerufen und schließt die Kommunikation ab.

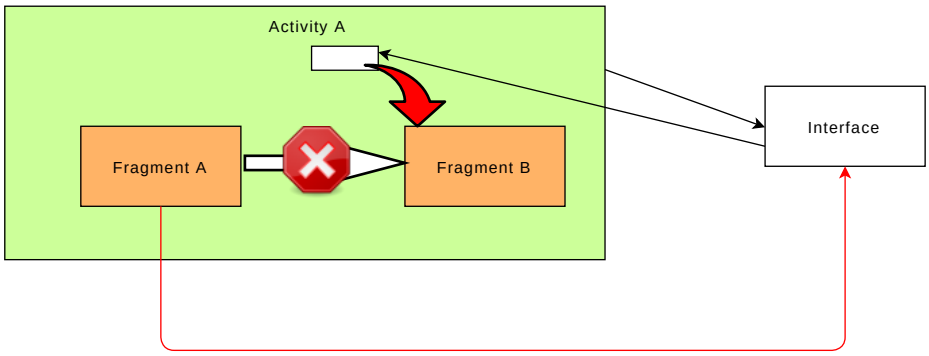


Abbildung 5.18: Konzept Fragment-Kommunikation

5.7.3 Paketstruktur

Bei der Erstellung von neuen Klassen soll folgende Paketstruktur berücksichtigt werden (siehe Abbildung 5.19). Dabei sind nur Pakete anzulegen, die auch derzeit benötigt werden. Beispielsweise ist das Paket *data* nur von Belang, wenn eine Datenbank-Nutzung integriert werden soll.

```
-data
        |-contract
        |-helper
-gui
        |-activity
        |-fragment
        |-adapter
        |-rows
-di
-model
        |-entity
        |-enums
-utils
-communication
        |- peer
                |-discovery
        |- rest
        |- robospice
                |-requests
                |-listener
```

Abbildung 5.19: Paketstruktur

5.7.4 Mockups

Die Abbildungen 5.20 und 5.21 zeigen die ersten Mockups der mobilen App. Dabei wurde bereits ein erster Auswahlbildschirm erstellt, mit dem unterschiedliche Sportarten ausgewählt werden können. In einem nächsten Schritt wird eine Startansicht erstellt, die das Spielfeld der jeweiligen Sportart sowie eine Übersicht über die Spieler der Teams anzeigen soll. Über einen Button werden dabei verschiedene Ansichten anwählbar sein. Die zweite Abbildung zeigt die mögliche Darstellung von Teamstatistiken und einem Seitenmenü, das animiert dargestellt werden kann.

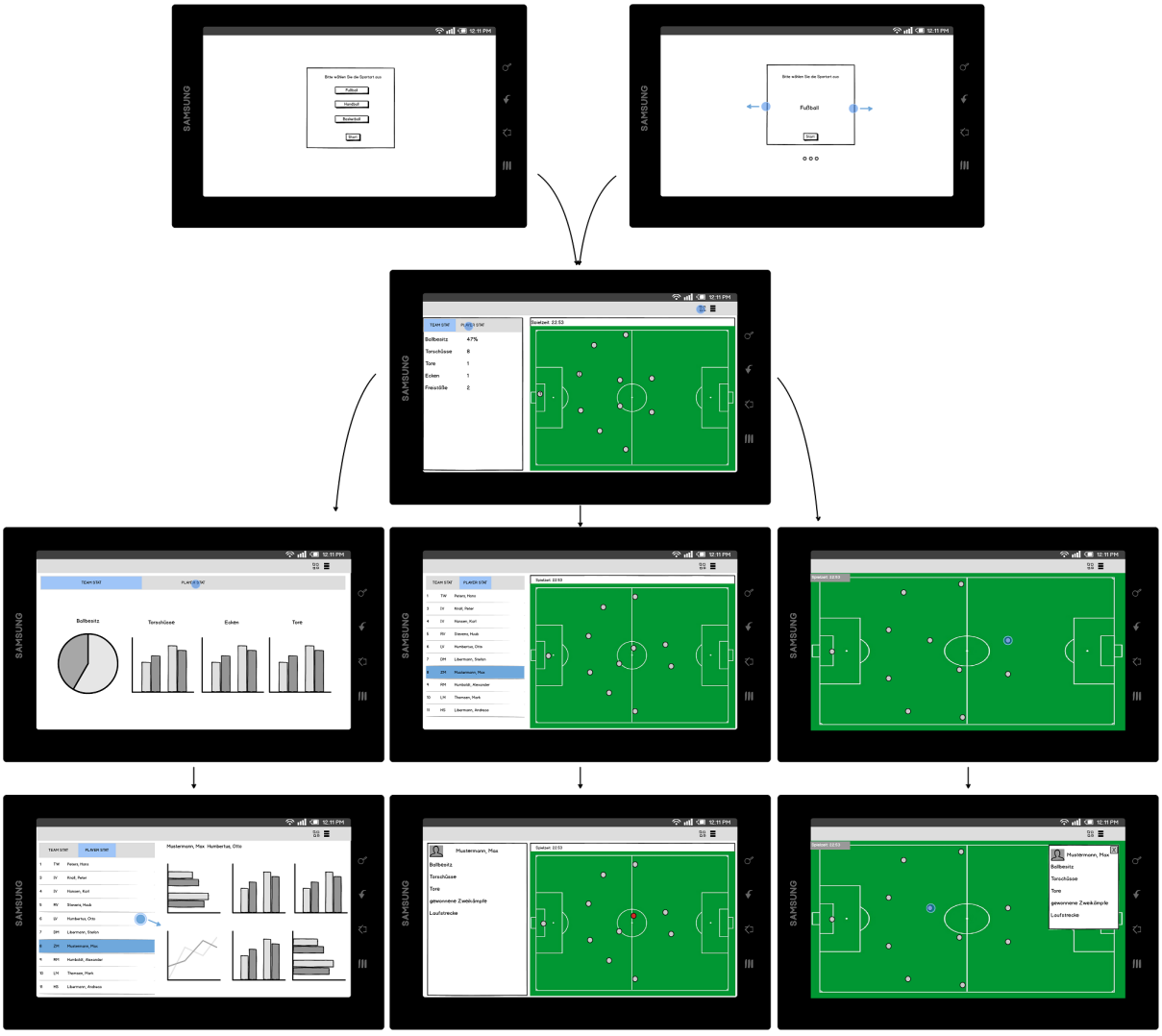


Abbildung 5.20: Allgemeine Mockups

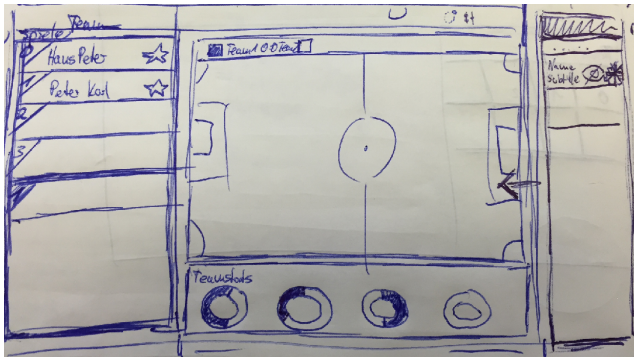


Abbildung 5.21: Mockup Splitscreen mit Teamstatistik

5.8 Monitoring Application

In diesem Abschnitt wird das Konzept zur Monitoring Application beschrieben. Mit der Monitoring Application soll eine Überwachung und Steuerung der Odysseus-Peers innerhalb eines Netzwerkes möglich sein. Um dies zu ermöglichen, wird die bereits verfügbare Webanwendung von Odysseus um einige Funktionen erweitert. Bislang ist es mit der Webanwendung möglich, die wichtigsten Funktionen vom Odysseus auszuführen, die auch über Odysseus-Studio angeboten werden. Dazu gehört z.B. das Erstellen, Starten, Stoppen und Entfernen von Anfragen oder auch das Betrachten eines Anfrageplans oder der Ergebnisse einer Anfrage.

Die Entscheidung für die Erweiterung der Webanwendung hat viele Vorteile, die im Folgenden kurz erläutert werden. Ein wesentlicher Vorteil für den Nutzer ist, dass die Webanwendung eine sehr große Flexibilität und Unabhängigkeit bietet. So benötigt der Nutzer lediglich einen Browser für die Ausführung der Anwendung. Es können die gängigen Browser (Chrome, Firefox, Safari etc.) von einem beliebigen Endgerät genutzt werden. Der Nutzer kann somit auch von den unterschiedlichsten Orten die Odysseus-Systeme überwachen. Für die Entwickler ist eine wesentlicher Vorteil, dass die Webanwendung bereits umfangreiche Schnittstellen zur Kommunikation mit einem Odysseus-System anbietet. So ist es es mit Webanwendung schon möglich, die Odysseus-Peers innerhalb eines Netzwerkes zu finden und mit diesen zu kommunizieren. Darüber hinaus kann die Webanwendung aufgrund seiner auf Maven[2]-basierten Architektur erweitert werden, ohne dass der bisherige Code verändert werden muss. Ein weiterer Vorteil ist, dass für die Monitoring Application hauptsächlich der Client der Webanwendung erweitert werden muss, um die Anforderung an die Monitoring Application zu erfüllen. Daher werden in Abbildung 5.22 einige Mockups gezeigt, um ein Eindruck zu geben, wie die Monitoring Application beim Client umgesetzt werden soll. Die Mockups zeigen eine Konsole zur Kommunikation mit einem Peer (1), den Log eines Peers (2), die Ping-Map eines Peer (3) und ein verteilen Anfrageplan (4).

[2] `https://maven.apache.org/`

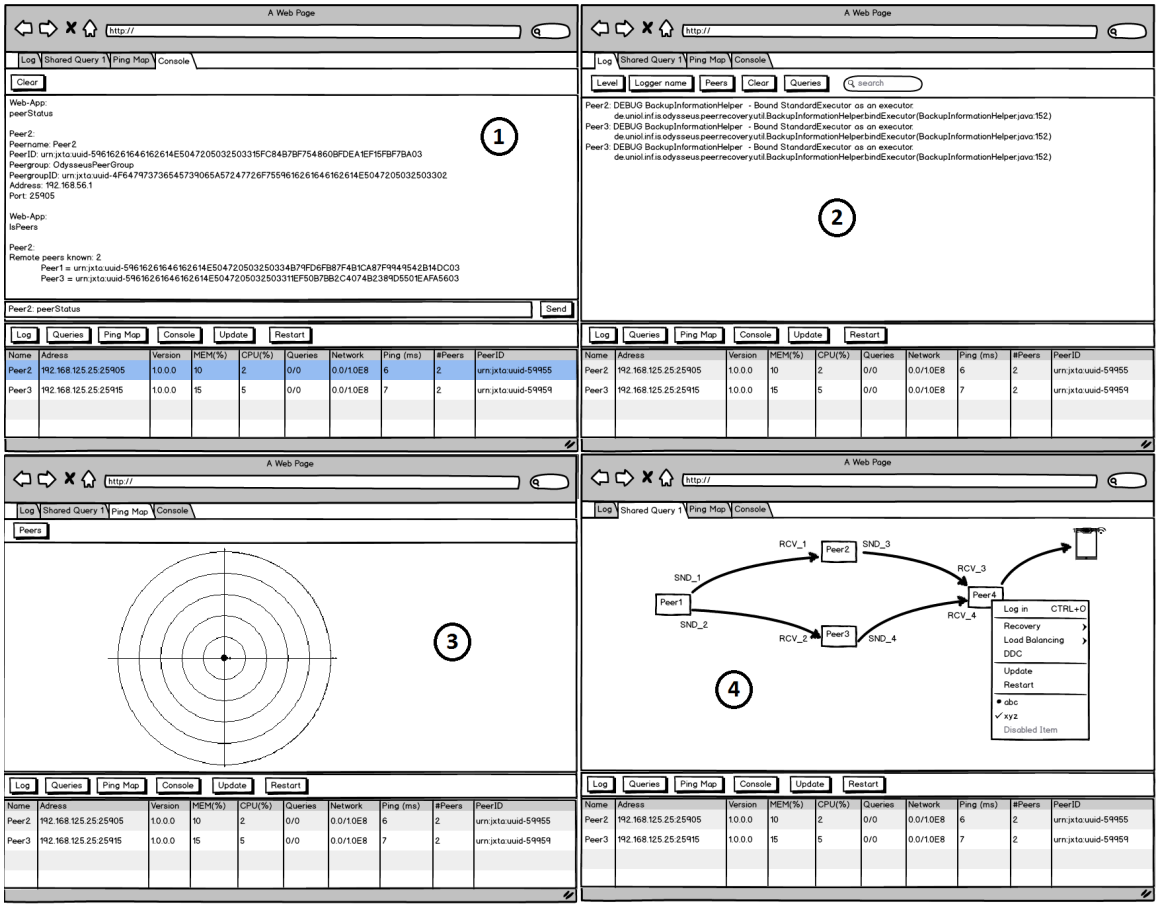


Abbildung 5.22: Mockups der Monitoring Application

5.9 Zusammenfassung

Es wurden Konzepte für die unterschiedlichen Teilschritte bis zur endgültigen Darstellung der Statistiken vorgestellt. Zur Lokalisierung eines einzelnen Spielers bieten sich dabei die Verfahren per GPS und WIFI an. Mittels *Lateration* werden die endgültigen Positionen der jeweiligen Sensoren auf einem Spielfeld bestimmt. Für die Unabhängigkeit gegenüber Sensoren dient ein *Zwischenschema*, in das die ankommenden Rohdaten transformiert werden sollen. Konstanten, u.a. Spielfeldabmessungen, Spielernamen und Sensor-Identifikationsnummern, werden dabei im *DDC* definiert. Zur Datenanalyse werden Anfragen ausgeführt, die die Daten aus dem Zwischenschema und den generischen Bewegungsdaten analysieren sollen und darauf aufbauend Sportart-spezifische Informationen generieren. Die Anfragen werden mittels der domänenspezifischen Sprache *SportsQL* umgesetzt und im Netzwerk verteilt ausgeführt. Um eine optimale Auslastung der einzelnen Peers zu gewährleisten, wird das Load-Balancing eingesetzt. Der Schwerpunkt liegt auf die Übertragungsmethoden *Parallel-Track-* und *Moving-State-Verfahren*. Damit beim Wegfall eines Peers nicht die gesamte Analyse abbricht, werden durch das *Recovery* Ausfälle kompensiert. Um eine Analyse der Daten zu initiieren, ist zu guter Letzt ein Client notwendig, der Anfragen an Herakles stellt. Eine mobile App für Android Betriebssysteme übernimmt diese Aufgabe und stellt Ergebnisse graphisch dar.

6 Implementierung

In diesem Kapitel wird auf die Implementierung von Herakles eingegangen. Dabei wird zunächst die Tracking Application erklärt, mit der es möglich ist, Positionsdaten zu messen. Anschließend wird innerhalb der Datenabstraktion erläutert, wie die gemessenen Rohdaten mit Hilfe des DDC in das Zwischenschema transformiert werden. In der Datenanalyse wird anhand zweier Anfragen beispielhaft gezeigt, mit welchen Operatoren die Datenströme verarbeitet werden. In einem weiteren Abschnitt wird auf die Umsetzung der Anfragesprache SportsQL eingegangen. Des Weiteren wird erklärt, wie die Kommunikation zwischen einzelnen Herakles-Komponenten umgesetzt wurde. In zwei weiteren Abschnitten wird die Implementierung von Load Balancing und Recovery in Odysseus erläutert. Nach der Erläuterung des LSOP Service, der Dienste für GUI-Anwendungen anbietet, wird das Kapitel mit der Umsetzung ebendieser GUI-Anwendungen abgeschlossen. Darunter fallen die Coach Application, die Developer Application und die Monitoring Application.

6.1 Tracking Application

Smartphones besitzen im Allgemeinen ein eigenes GPS-Modul. Dadurch ist es möglich, die Smartphones als Positionssensoren zu nutzen, die ihren aktuellen Standort an das System Herakles senden. Hierfür wurde die App „GPSender“ entwickelt, die auf die GPS-Funktion zugreift, die Positionsdaten aufbereitet und dann an einen entsprechenden Server weiterleitet. Die folgenden Abschnitte erläutern die zugrunde liegende Architektur der Applikation, die auf Basis der Android-Plattform entwickelt wurde.

6.1.1 Architektur - GUI

Allgemein beschränkt sich die Architektur der GUI auf eine Hauptansicht. Hier werden die GPS-Daten und der aktuelle Zustand des Trackings (aktiv oder inaktiv) dargestellt. Via „Start-“ und „Stop-Button“ wird das Tracking gestartet bzw. gestoppt.

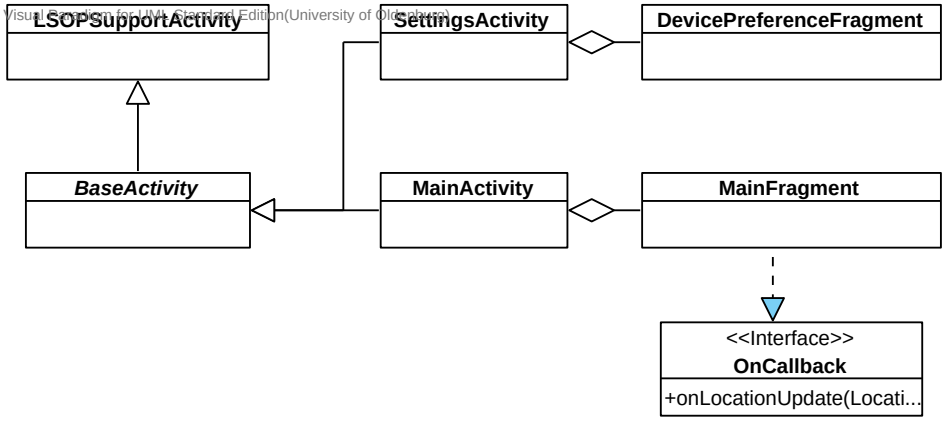


Abbildung 6.1: Grundarchitektur Tracking Applikation

Die Basis der Applikation bildet die `MainActivity`, die in der Abbildung 6.1 zu erkennen ist. Sie wird von der abstrakten Klasse `BaseActivity` abgeleitet. In der `BaseActivity` werden grundlegende Layouts festgelegt, die dann für alle weiteren Activities gelten. Da diese Applikation für Android-Geräte ab Android-Version 5.0 konzipiert wurde, mussten die „AppCompat"-Bibliothek[1] eingebunden werden, die es ermöglicht, dass auch ältere Geräte die Layouts richtig darstellen. Da zur Entwicklungszeit das Projekt „Roboguice"[2] in der Version 2.0 keine Unterstützung für die benötigte „AppCompat"-Bibliothek angeboten hat, wurde die Hilfsklasse `LSOPSupportActivity` erstellt, die das Problem löst. Als Basisfragment dient das `MainFragment`, das durch die `MainActivity` umgesetzt wird. Für Einstellungen wurde ebenfalls eine Activity mit zugehörigem Fragment entwickelt. Hierbei handelt es sich um die `SettingsActivity` und das `DevicePreferenceFragment`. Zur Darstellung der aktuellen Position auf dem Display implementiert das `MainFragment` die Methoden des Interfaces `OnCallback`. Das Tracking selbst soll im Folgenden näher erläutert werden.

6.1.2 Architektur - Tracking

Zum Tracking gehören in der App mehrere Schritte. Durch den „Start-Button" im `MainFragment` wird ein Service gestartet, der auch weiterläuft, wenn die Applikation pausiert wird. Sind neue Positionsdaten vorhanden, werden sie einerseits auf dem Display dargestellt und andererseits an einen Server geschickt. Die beim Tracking beteiligten Klassen sind in Abbildung 6.2 abgebildet.

Die abstrakte Klasse `AbstractTracker` wird von der Android-Klasse `Services` abgeleitet, sodass sie als Service gestartet werden kann. Von dieser Klasse werden die drei konkreten Klassen `GPSTracker`, `PlayTracker` und `WifiTracker` abgeleitet, die Implementierungen für die Methoden `startBackgroundTask()`, `startTracking()` und `stopTracking()` liefern. Wie die Namen bereits vermuten lassen, handelt es sich dabei um unterschiedliche Implementierungen für die Positionsbestimmung. Der `GPSTracker` und der `PlayTracker` arbeiten mit GPS Koordinaten. Sie verwenden unterschiedliche Möglichkeiten des Android-Systems, um die Position zu bestimmen. Der erstere nutzt die vom Android-System angebotenen Methoden, bei denen das GPS Signal direkt genutzt wird. Der zweite hingegen nutzt die PlayServices von Google, mit denen die Positionsbestimmung weiter abstrahiert und damit vereinfacht wird. Der `WifiTracker` arbeitet mit den Abständen zu mindestens drei Routern, sodass eine Triangulation durchgeführt werden kann (siehe hierzu auch Abschnitt 2.2.0.1).

Bei der Initialisierung des Services wird die Positionssuche und ein Hintergrundprozess gestartet. Dieser Hintergrundprozess sendet zum einen neue Positionsdaten mithilfe der Klasse `ConnectionManager` an einen vordefinierten Server und zum anderen zusätzlich eine anwendungsweite Nachricht, die dann von der abstrakten Klasse `AbstractTrackingReceiver` empfangen und verarbeitet wird. Der Server, der die aktuellen Positionsdaten der Applikation empfängt, wird als Quelle für das P2P Netzwerk genutzt.

[1] `https://developer.android.com/tools/support-library/features.html`
[2] `https://github.com/roboguice/roboguice`

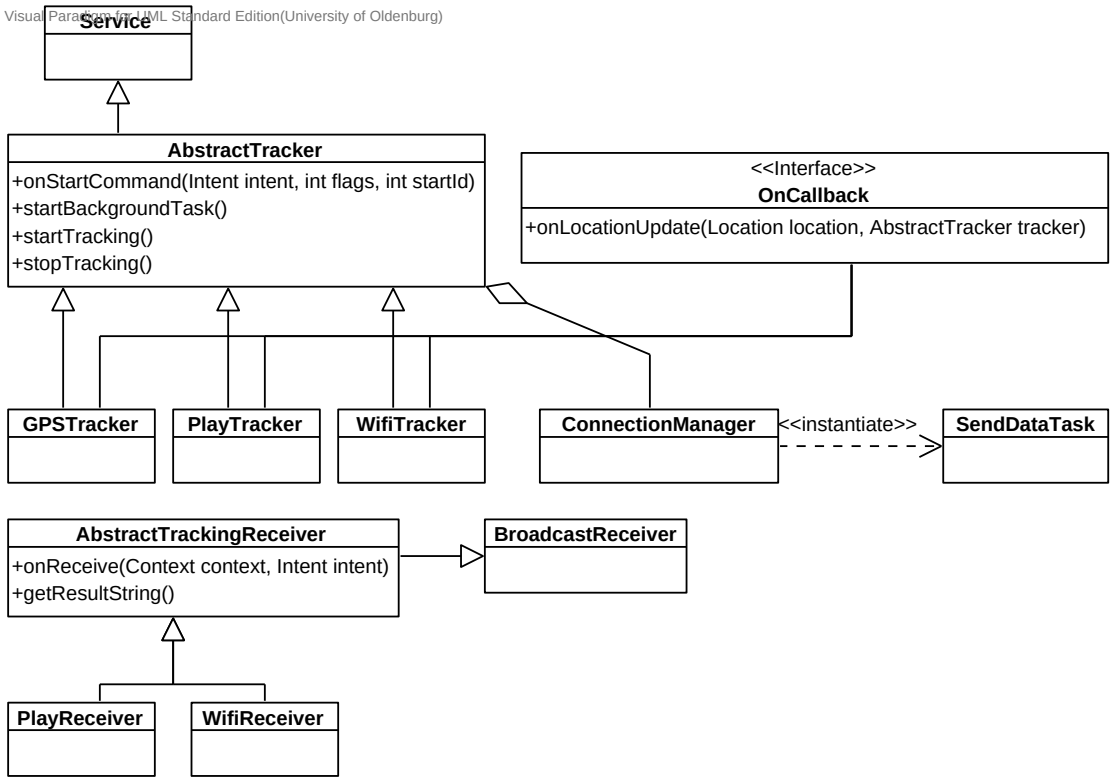


Abbildung 6.2: Grundarchitektur Tracking Applikation

6.1.3 GPServer

Um die Daten von den mobilen Geräten mit der GPSender App empfangen und weiterleiten zu können, wird die GPServer App genutzt. Diese empfängt die Daten der Smartphones über UDP und leitet diese gegebenenfalls an eine Odysseus-Instanz weiter. Zusätzlich kann die GPServer App die Daten auch als CSV-Datei abspeichern. Die Architektur der GPServer App ist simpel gehalten. Die beteiligten Klassen sind in Abbildung 6.3 zu sehen.

Das Modell beinhaltet die Klassen `GPSender` und `Odysseus`. Die erste repräsentiert ein Smartphone, welches die GPSender App ausführt und Daten an den Server sendet. Das Modell beinhaltet unter anderem den Namen des Spielers, dessen Position über das Smartphone aufgezeichnet wird, und den Zeitstempel der letzten Aktualisierung. Das Modell `Odysseus` repräsentiert eine Odysseus-Instanz, an die die Daten der Smartphones weitergeleitet werden. Dieses Modell beinhaltet einen Socket, über den die Daten gesendet werden. Die Listen `ConnectedSenders` und `ConnectedSockets` beinhalten jeweils alle verbundenen Smartphones bzw. Odysseus-Instanzen.

Neben den Modellen gibt es noch den Thread `UDPServiceWorker`. Dieser empfängt kontinuierlich die Daten von den verbundenen Smartphones und sendet diese an die verbundenen Odysseus-Instanzen weiter. Außerdem speichert diese Klasse gegebenenfalls die Daten in einer Datei.

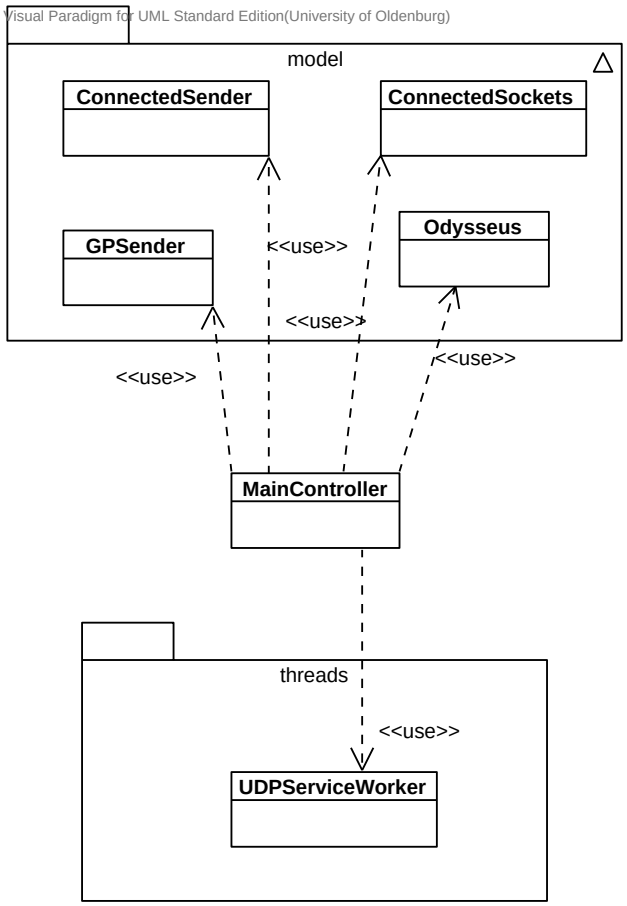


Abbildung 6.3: Klassen in der GPServer App.

Der `MainController` steuert die Oberfläche, die mit JavaFX realisiert ist. Hier werden zudem die entsprechenden Threads gestartet.

6.2 Datenabstraktion

Im folgenden Abschnitt wird beschrieben, wie der Datenstrom vor der Verarbeitung und Analyse zunächst in ein einheitliches Schema transformiert wird. Das ist notwendig um eine Sensorunabhängigkeit zu gewährleisten. Darüber hinaus wird die Funktionsweise des DDC beschrieben und insbesondere die Verteilung der einzelnen Einträge im Netzwerk erläutert. Nach Abschluss der Transformation liegen die Daten in einem einheitlichen Format vor, sodass von den jeweiligen Quellen abstrahiert werden kann.

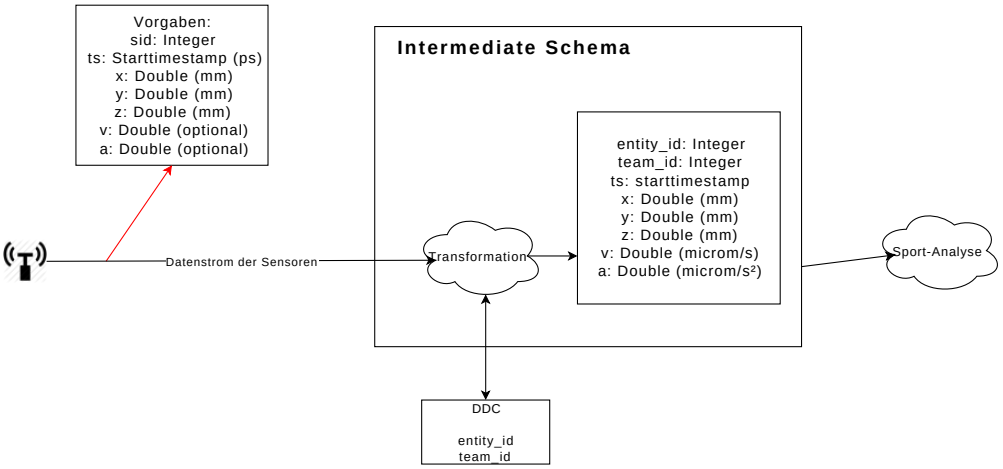


Abbildung 6.4: Transformation in das Zwischenschema

6.2.1 Transformation der Rohdaten

Die Transformation des ankommenden Datenstroms in das Zwischenschema muss individuell auf die verwendeten Sensoren angepasst werden. Zur Evaluation unseres Systems verwenden wir u.a. Daten des Fußball-Computerspiels Eat the Whistle (ETW). Im Folgenden wird beschrieben, wie eine Datentransformation der ETW-Rohdaten in das Zwischenschema aussehen kann. Die Abbildung 6.5 zeigt die Operatoren, die für die Transformation der Rohdaten verantwortlich sind.

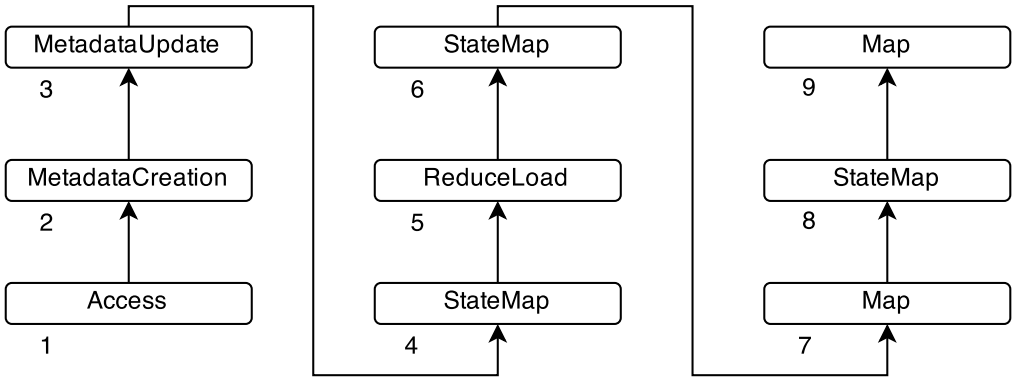


Abbildung 6.5: Transformation der ETW-Rohdaten in das Zwischenschema

1. Der Access-Operator empfängt die Rohdaten von einem ETW-Server. Die Rohdaten bestehen dabei aus einem Zeitstempel, der Sensor-Id und Positionskoordinaten (x, y, z). (Ausgabeschema: ts, sid, x, y, z)

2. Der MetadataCreation-Operator erzeugt Metadaten für ein Tupel. (Ausgabeschema: ts, sid, x, y, z)

3. Der MetadataUpdate-Operator legt den Startzeitstempel eines Tupels fest. Dieser kann aus den Rohdaten übernommen werden. (Ausgabeschema: ts, sid, x, y, z)

4. Der StateMap-Operator ordnet jedem Tupel eine Entitäts-Id und eine Team-Id zu. Diese werden anhand der `sid` aus dem DDC gelesen. (Ausgabeschema: ts, entity_id, team_id, x, y, z)

5. Der ReduceLoad-Operator garantiert dafür, dass die Datenrate eine vorher festgelegte Grenze nicht überschreitet. Dadurch wird erreicht, dass die Last, welche durch die Verarbeitung der Daten entsteht, ein bestimmtes Level nicht überschreitet.

6. Der StateMap-Operator berechnet zum einen die euklidische Distanz zweier aufeinanderfolgender Tupel einer Entität. Zum anderen wird die Differenz der Zeitstempel zweier aufeinander folgender Tupel einer Entität berechnet. (Ausgabeschema: ts, entity_id, team_id, x, y, z, euclideanDistance, deltaT)

7. Der Map-Operator berechnet die Geschwindigkeit einer Entität aus dem Quotienten von `euclideanDistance` und `deltaT` (Ausgabeschema: ts, entity_id, team_id, x, y, z, v)

8. Der StateMap-Operator berechnet zum einen die Differenz der Geschwindigkeiten zweier aufeinanderfolgender Tupel einer Entität. Zum anderen wird die Differenz der Zeitstempel zweier aufeinander folgender Tupel einer Entität berechnet. (Ausgabeschema: ts, entity_id, team_id, x, y, z, deltaV, deltaT)

9. Der Map-Operator berechnet die Beschleunigung aus dem Quotienten von `deltaV` und `deltaT` (Ausgabeschema: ts, entity_id, team_id, x, y, z, v, a)

6.2.2 Distributed Data Container

Das DDC wird für die Nutzung von Konstanten im P2P-Netzwerk benötigt. Wie bereits im Konzept beschrieben, ist eine einfache Speicherung der Konstanten auf einem Peer nicht sinnvoll, da dieser das Netzwerk verlassen kann und im Anschluss kein Zugriff mehr auf die gespeicherten Daten besteht. Darüber hinaus wäre ständiger Zugriff über das Netzwerk vergleichsweise langsam. Aus diesem Grund müssen die Konstanten auf alle Peers des Netzwerkes verteilt werden. Dies stellt jedoch eine weitere Herausforderung dar, da die Daten zu jeder Zeit konsistent gehalten werden müssen. In diesem Abschnitt wird die generelle Funktionsweise des DDC erläutert und speziell auf die Verteilung und Synchronisation der Daten eingegangen.

6.2.2.1 Speicherung von Konstanten

Das DDC verwaltet statische Schlüssel-Wert-Paare und deren Änderungszeitstempel (in Millisekunden). Es enthält Informationen, welche für jedes Sensorsystem und jeden Systemaufbau individuell sind und somit bei jeder Einrichtung des Systems neu erfasst und eingetragen werden müssen. Es muss dabei sicherstellen, dass Werte für bereits vorhandene Schlüssel nur dann überschrieben werden, wenn der Änderungszeitstempel neuer ist. Das DDC ist ein Service, der über OSGi gebunden wird. Als konkrete Schnittstellen für Sportanalyse-Parameter dienen die `AbstractSportsDDCAccess`-Klasse und ihre Subklassen, die für vordefinierte Schlüssel entsprechende Getter bereitstellt. Auf diese statischen Informationen kann bei der Datenanalyse zugegriffen werden. Aufgrund der verteilten Architektur unseres Systems muss zudem sichergestellt werden, dass Änderungen am DDC allen Teilnehmern mitgeteilt werden. Für Fußball müssen bspw. die folgenden statischen Informationen im DDC bereitgestellt werden:

- Spielrichtung der Teams
  - Es muss sowohl die ID für das Team festgelegt werden, welches von rechts nach links spielt, als auch für das Team, welches von links nach rechts spielt
- Spielfeldabmessungen
  - Die individuellen Abmessungen des Spielfeldes müssen erfasst werden
- Torpositionen
  - Die Position der Tore muss erfasst werden
- Sensorsystem-Informationen
  - sensoridlist: Eine Liste mit allen Sensoren des Systems
  - BTU: Die Basis-Zeiteinheit der Daten im Zwischenschema
  - conversionfactor: Faktor zur Umrechnung der Zeitstempel des Sensorsystems zur BTU
  - timebetween: Der (minimale) zeitliche Abstand zwischen zwei Tupel eines Sensors im Zwischenschema
- Sensor-Informationen
  - Entity-ID: Die ID der Entität, welche dem Sensor zugeteilt ist (z.B. die Spieler-ID)

- Entity: Der Name der Entität (z.B. der Spielername)
- Remark: Zusätzliche Informationen zum Sensor (z.B. linkes Bein)
- Team-ID: ID des Teams, welchem die Entität zugeteilt ist

Als Quelle für die Einträge im DDC können zum einen eine fest kodierte Eigenschafts-Datei oder zum anderen ein DDC-Advertisement dienen, die im folgenden Abschnitt genauer beschrieben werden. Die Datei kann von Hand vor dem Starten von Odysseus editiert werden. Sie muss *datacontainer.ddc* heißen und sich im Ordner <Windows-Benutzer>/odysseus befinden (Mac OS: ~/.odysseus/). Die andere Art, wie die Datei editiert wird, ist durch Odysseus: Wenn ein DDC durch ein Advertisement verändert wird oder es die Eigenschafts-Datei noch nicht gibt, wird die Datei geändert oder ggf. neu erstellt. Bei diesem programmatischen Editieren wird für jedes Schlüssel-Wert-Paar zusätzlich ein Änderungszeitstempel in der Datei hinterlegt. Beim manuellen Editieren der Datei (dadurch erkennbar, dass keine Zeitstempel in der Datei sind) dient das Änderungsdatum der Datei selbst als Zeitstempel.

Wird ein Peer heruntergefahren, werden alle derzeit im DDC hinterlegten Schlüssel-Wert-Paare in eine Datei persistiert, sofern ein entsprechendes Flag gesetzt ist. Dies hat den Vorteil, dass beim erneuten Hochfahren des Peers alle Einträge direkt wieder verfügbar sind. Wird das Flag nicht gesetzt, werden die betreffenden Einträge nicht gesichert. Dies ist vor allem dann sinnvoll, wenn es sich um Einträge handelt die für die aktuell laufende Instanz relevant, allerdings beim erneuten Starten des Peers nicht mehr gültig sind. Einen solchen Fall gibt es im Recovery, da hier die Backup-Informationen im DDC hinterlegt werden. Der Vorteil ist, dass die Informationen auf einfache Art und Weise zwischen den Peers ausgetauscht werden können und somit prinzipiell jeder Peer in der Lage ist ein Recovery durchzuführen.

Die Klasse `DDCConsole` fügt der OSGi-Konsole Kommandos für alle Methoden des DDCs zur Verfügung, wodurch sowohl das DDC als auch die Verteilung getestet werden kann. Auch hierüber können manuell Einträge erfasst und verteilt werden.

6.2.2.2 Verteilung der Konstanten

Aufgrund der Tatsache, dass wir Odysseus im P2P-Kontext verwenden, müssen die Informationen aus dem DDC allen Peers zur Verfügung gestellt werden. Dabei wird zwischen einem initialen Verteilen und der Verteilung von Änderungen unterschieden. Der Vorteil bei dieser Unterscheidung ist, dass nur zu Beginn ein Mal alle Einträge verteilt werden müssen und im Anschluss nur noch die Änderungen, die auf dem jeweiligen Peer durchgeführt wurden. Die Verteilung wird durch verschiedene JXTA-Advertisements durchgeführt, die im Folgenden genauer erläutert werden:

- DistributedDataContainerAdvertisement (`DDCAdvertisement`): Dieses DDC-Advertisement sagt aus, dass der sendene Peer ein DDC hat. Das Advertisement wird nach dem Laden des DDC aus der Datei verteilt. Andere Peers, die dieses Advertisement erhalten, können dem sendenen Peer mit einem `DDCRequest` (Message) antworten. Dieser Peer antwortet wiederum mit einer `DDCMessage`, die die `DDCEntries` enthält, auf den Request.

- DistributedDataContainerChangeAdvertisement (`DDCChangeAdvertisement`): Dieses Advertisement sagt aus, dass der sendene Peer Änderungen an seinem DDC hat. Das Advertisement wird

Änderungen am DDC verteilt. Andere Peers, die dieses Advertisement erhalten, können dem sendenen Peer mit einem `DDCRequest` (Message) antworten. Dieser Peer antwortet wiederum mit einer `DDCMessage`, die zum einen die hinzugefügten und zum anderen die entfernten `DDCEntries` enthält, auf den Request.

- DistributedDataContainerRequestAdvertisement (`DDCRequestAdvertisement`): Dieses Advertisement sagt aus, dass der sendene Peer neu im Netzwerk ist und ein DDC haben möchte. Andere Peers, die dieses Advertisement erhalten, antworten dem sendenen Peer mit einer `DDCMessage`, die die `DDCEntries` enthält, auf den Request.

Diese `DDCAdvertisements` können durch einen `DistributedDataContainerAdvertisementGenerator` erstellt werden. Das Auslesen eines DDCAdvertisements und das anschließende Schreiben ins DDC übernimmt der `DistributedDataContainerCommunicator`. Die Abbildung 6.6 visualisiert beispielhaft das initiale Verteilen der DDC Einträge.

Neben der beschriebenen Verteilung nach dem Hochfahren des Peers, gibt es auch noch eine Verteilung von geänderten Einträgen.

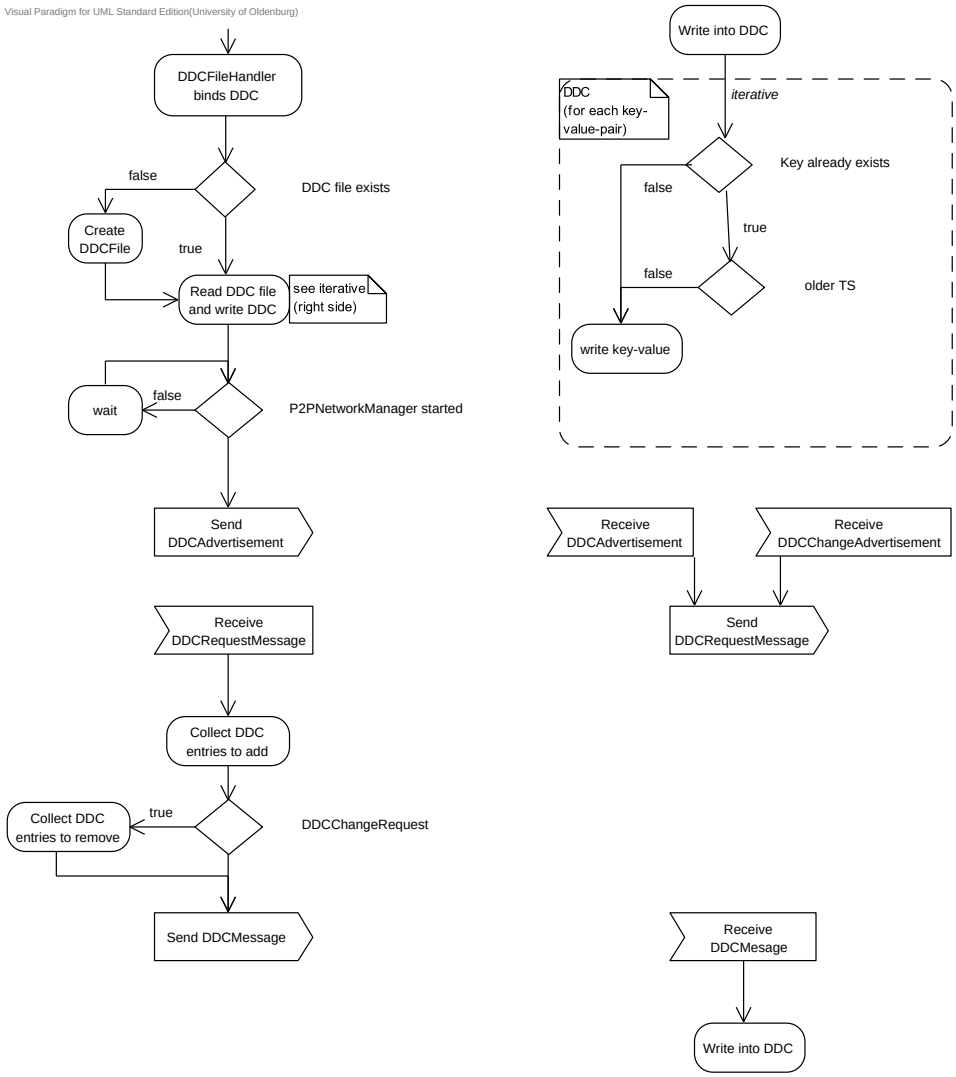


Abbildung 6.6: Anfrageplan zur Ermittlung der Sprints

```
#PARSER PQL
#ADDQUERY
#DEFINE gameStream soccergame
#DEFINE firstddcKey "sensorid."
#DEFINE secondddcKey ",entity_id"

ddcValueTest = MAP({
        expressions = [
                ['fromDDC(concat(concat(${firstddcKey},sid),${
                   ↪ secondddcKey}))','entity_id'],
                ['fromDDC(concat(concat(${firstddcKey},sid),",
                   ↪ entity"))','entity'],
                ['fromDDC(concat(concat(${firstddcKey},sid),",
                   ↪ team_id"))','team_id'],
                ['fromDDC(concat(concat(${firstddcKey},sid),",
                   ↪ remark"))','remark'],
                ['x','ball_x'],
                ['y','ball_y'],
                ['z','ball_z'],
                ['sid','sid']
        ]
},soccergame)
```

Listing 6.1: Verwendung von DDC-Einträgen in PQL

6.2.2.3 Datenanreicherung aus DDC

Über die im DDC hinterlegten Einträge kann der eingehende Datenstrom mit weiteren Inhalten angereichert werden. Hierfür kann im MAP-Operator eine entsprechende MEP-Funktion genutzt werden. Hier muss lediglich der hierarchische Schlüssel angegeben werden. Listing 6.1 zeigt die Verwendung der beschriebenen Funktion in PQL.

Neben der Verwendung in PQL ist auch eine Verwendung direkt aus dem Quellcode möglich. Dies erlaubt es beispielsweise, dass Einträge aus dem DDC während der Verarbeitung in einem Operator abgerufen werden können. Darüber hinaus können entsprechende Programmroutinen speziell auf die Werte im DDC abgestimmt werden.

6.3 Datenanalyse

Für die Datenanalyse werden in Herakles mehrere Anfragen erstellt, die unterschiedliche Ereignisse in den Daten erkennen sollen. Dabei gibt es mit Analysezeit, Spielerpositionen und Heatmap drei sportartenunabhänigen Anfragen. Speziell für Fußball gibt es die Anfragen Tore, Torschüsse, Ecken, Ballkontakte, Laufstrecke, Sprints und Pässe. Im Folgenden soll die Datenanalyse beispielhaft anhand der Anfragen Analysezeit und Sprints erklärt werden. Während die Abbildung 6.7 den Anfrageplan zu der Analysezeit darstellt, beinhaltet die Abbildung 6.8 den Anfrageplan zu den Sprints.

6.3.1 Analysezeit-Anfrage

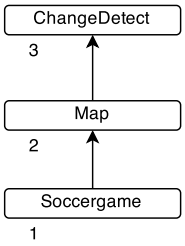


Abbildung 6.7: Anfrageplan zur Ermittlung der Analysezeit

1. Der Soccergame-Operator ist die Quelle, welche intern die Transformation der Rohdaten in das Zwischenschema vornimmt und darauf aufbauend Geschwindigkeit und Beschleunigung ausrechnet. (Ausgabeschema: entity_id, team_id, ts, x, y, z, v, a)

2. Der Map-Operator berechnet aus dem Timestamp die Minuten und Sekunden. Dies geschieht in Abhängigkeit der BaseTimeUnit, welche aus dem DDC gelesen werden kann. (Ausgabeschema: minute, second)

3. Der ChangeDetect-Operator achtet darauf, dass für jede Minute und Sekunde nur ein Tupel weitergeleitet wird. Dies ist notwendig, da bspw. bei Mikrosekunden als BaseTimeUnit mehrere Tupel mit der gleichen Zeit von dem Map-Operator ausgegeben werden. (Ausgabeschema: minute, second)

6.3.2 Sprints-Anfrage

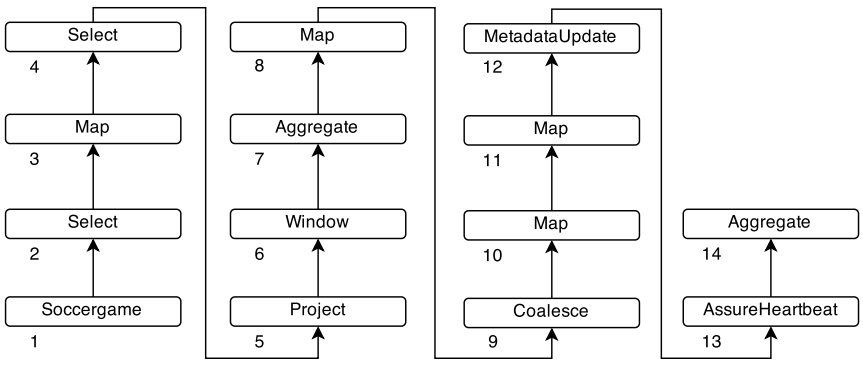


Abbildung 6.8: Anfrageplan zur Ermittlung der Sprints

1. Der Soccergame-Operator ist die Quelle, welche intern die Transformation der Rohdaten in das Zwischenschema vornimmt und darauf aufbauend Geschwindigkeit und Beschleunigung ausrechnet. (Ausgabeschema: entity_id, team_id, ts, x, y, z, v, a)

2. Der Select-Operator filtert alle Tupel heraus, die nicht zu den Spielern beider Teams gehören. Dies wird dadurch erreicht, indem überprüft wird, ob die Team-ID eines Tupels zu einem der beiden Teams gehört. Die beiden Team-Ids können dabei aus dem DDC gelesen werden. (Ausgabeschema: entity_id, team_id, ts, x, y, z, v, a)

3. Der Map-Operator berechnet aus dem Timestamp die Minuten und Sekunden. Dies geschieht in Abhängigkeit der BaseTimeUnit, welche aus dem DDC gelesen werden kann. Die Berechnung der Zeit ist notwendig, wenn die Sprints nur aus einer bestimmten Zeitspanne ermittelt werden sollen. (Ausgabeschema: entity_id, team_id, ts, minute, second, x, y, z, v, a)

4. Der Select-Operator filtert alle Tupel heraus, die nicht in der vom Nutzer bestimmten Zeitspanne liegen. (Ausgabeschema: entity_id, team_id, ts, minute, second, x, y, z, v, a)

5. Der Project-Operator entfernt Attribute, die nicht benötigt werden. (Ausgabeschema: entity_id, team_id, ts, v)

6. Der Window-Operator definiert ein Tumbling-Zeitfenster. Dabei wird der Datenstrom in gleich große disjunkte Abschnitte unterteilt, so dass jedes Datenstromelement nur in einem Fenster vorkommt. Die Fenster-Größe beträgt hierbei 500 Millisekunden. (Ausgabeschema: entity_id, team_id, ts, v)

7. Der Aggregate-Operator berechnet die durchschnittliche Geschwindigkeit eines Spielers innerhalb des definierten Zeitfensters.(Ausgabeschema: entity_id, team_id, min_ts, max_ts, avg_v)

8. Der Map-Operator rechnet die durchschnittliche Geschwindigkeit von Mikrometer/Sekunde in Kilometer/Stunde um. (Ausgabeschema: entity_id, team_id, min_ts, max_ts, avg_v, speed_kmh)

9. Der Coalesce-Operator aggregiert aufeinander folgende Tupel eines Spielers. Dabei startet die Aggregation, wenn ein Tupel eines Spielers geliefert wird, bei dem die Geschwindigkeit höher ist als 24 Km/h. Mit diesem Tupel startet der Sprint eines Spielers. Die Aggregation endet, sobald ein Tupel eines Spielers eine Geschwindigkeit von weniger als 19 Km/h hat. Mit diesem Tupel endet der Sprint eines Spielers. Aus diesen Daten kann nun aggregiert, bei welchem Timestamp der Sprint startet und endet. Des Weiteren kann die durchschnittliche und maximale Geschwindigkeit berechnet werden. (Ausgabeschema: entity_id, team_id, min_ts, max_ts, avg_speed, max_speed)

10. Der Map-Operator berechnet aus der Differenz der beiden Zeitstempel und der durchschnittlichen Geschwindigkeit die Distanz eines Sprints. (Ausgabeschema: entity_id, team_id, ts_difference, avg_speed, max_speed, sprint_distance)

11. Der nächste Map-Operator teilt einen Sprint anhand der Sprintdistanz in eine Kategorie ein. Es gibt fünf Kategorien, wobei jede Kategorie für ein Distanz-Intervall steht. Beispielsweise gehört ein Sprint zu Kategorie 1, wenn die Distanz zwischen 1 und 10 Metern ist. (Ausgabeschema: entity_id, team_id, ts_difference, avg_speed, max_speed, sprint_distance, dist_1_10, dist_11_20, dist_21_30, dist_31_40, dist_41_x)

12. Der MetadataUpdate-Operator entfernt den Endzeitstempel eines Tupels, so dass es für den letzten Aggregationsoperator kein Fenster gibt. Ein Fenster ist in diesem Fall nicht notwendig, da die Sprints über dem gesamten Messungszeitraum gezählt werden sollen. (Ausgabeschema: entity_id, team_id, ts_difference, avg_speed, max_speed, sprint_distance, dist_1_10, dist_11_20, dist_21_30, dist_31_40,dist_41_x)

13. Der AssureHeartbeat-Operator garantiert, dass alle 5 Sekunden ein Heartbeat an den Aggregationsoperator gesendet wird. Dies ist notwendig, damit die Anfrage in regelmäßigen Abständen auch dann aggregierte Ergebnisse liefert, wenn kein neuer Sprint gemessen wurde. (Ausgabeschema: entity_id, team_id, ts_difference, avg_speed, max_speed, sprint_distance, dist_1_10, dist_11_20, dist_21_30, dist_31_40, dist_41_x)

14. Der Aggregate-Operator aggregiert alle Sprints eines Spielers. Die Anfrage liefert somit letztlich die Anzahl an Sprints eines Spielers pro Kategorie, die durchschnittliche Geschwindigkeit, die maximale Geschwindigkeit und die durchschnittliche Sprintdistanz. (Ausgabeschema: entity_id, team_id, avg_speed, max_speed, avg_distance, dist_1_10, dist_11_20, dist_21_30, dist_31_40, dist_41_x)

6.4 SportsQL

In diesem Abschnitt wird auf die Implementierung des SportsQL-Konzeptes aus 5.4 eingegangen. Um eine SportsQL-Anfrage in Odysseus auszuführen, muss diese ähnlich wie bei PQL oder CQL mit Hilfe eines Parsers in einen logischen Anfrageplan übersetzt werden. Dieser logische Anfrageplan beinhaltet alle logischen Operatoren, die für die Ausführung der Anfrage notwendig ist. Da in SportsQL selber keine Operatoren definiert werden, muss es für jede mögliche SportsQL-Anfrage einen eigenen Parser geben, der die notwendigen logischen Operatoren definiert. Die Abbildung 6.9 gibt einen Überblick über die Klassen und Schnittstellen, mit denen die SportsQL-Anfragen erstellt werden.

Es gibt eine zentrale Klasse `SportsQLParser`, welche die von Odysseus bereitgestellte Schnittstelle `IQueryParser` implementiert. Die `parse`-Methode wird aufgerufen, sobald eine SportsQL-Anfrage erstellt werden soll. Nach Aufrufen der Methode wird zunächst aus der textuellen Anfrage ein `SportsQLQuery`-Objekt erstellt, welches u.a. den Namen, die Sportart, den Statistiktypen und die Parameter der Anfrage beinhaltet. Das ist simpel möglich, da die SportsQL-Anfrage ein JSON-Objekt ist. Anschließend wird mit Hilfe der Registry `SportsQLParserRegistry` der zu der Anfrage passende `ISportsQLParser` angefordert. Dies geschieht in Abhängigkeit der Sportart, des Statistiktyps und des Namens der Anfrage. Beispielsweise wird der Parser `SprintsPlayer-SportsQLParser` zurückgeliefert, wenn die Sportart *Soccer*, der Statistiktyp *Player* und der Name der Anfrage *Sprints* ist. Die einzelnen SportsQL-Parser verwenden zum Erstellen des logischen Anfrageplans meistens die Hilfsklasse `OperatorBuildHelper`, die zahlreiche Methoden anbietet, um Operatoren zu erstellen. Nachdem alle notwendigen logischen Operatoren erstellt worden sind, liefert der Parser eine Instanz von `ILogicalQuery` zurück, welche von Odysseus in einen physischen Anfrageplan transformiert und letztlich ausgeführt wird.

Zum Erstellen einer neuen SportsQL-Anfrage muss lediglich eine neue Klasse geschrieben werden, die die Schnittstelle `ISportsQLParser` implementiert und die über den OSGi-Service-Mechanismus bekannt gemacht wird. In der Klasse muss darauf geachtet werden, dass über Annotationen die Anfrage beschrieben wird. Listing 6.2 zeigt dies beispielhaft für die Sprints-Anfrage.

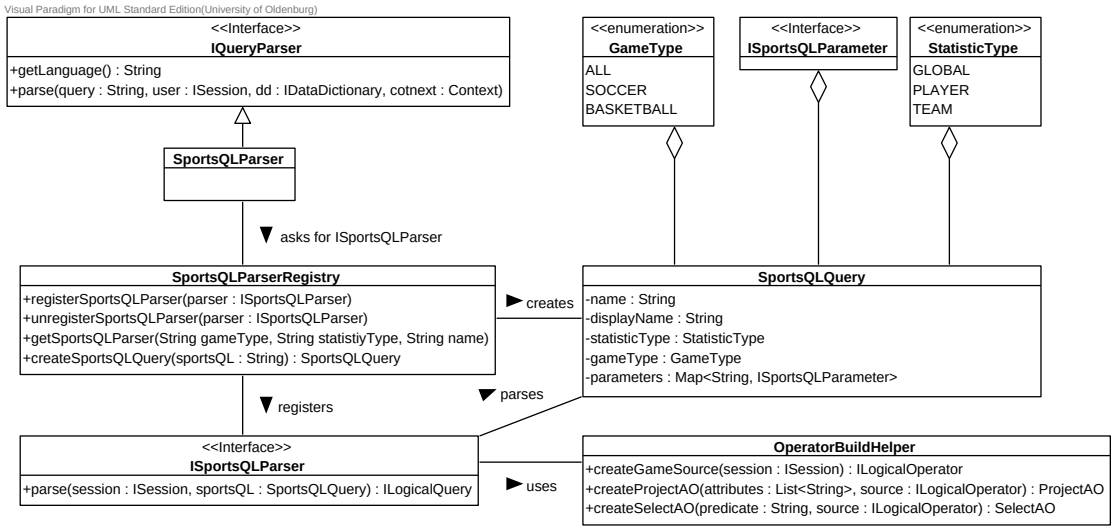


Abbildung 6.9: Parsen von SportsQL-Anfragen

Neben dem Namen der Anfrage muss festgelegt werden, welche Sportarten und Statistiktypen die Anfrage unterstützt. Des Weiteren können Parameter definiert werden, welche die Anfrage benötigt. Zu jedem Parameter muss der Name und die zuständige Klasse festgelegt werden. Außerdem kann bestimmt werden, ob ein Parameter zwingend (mandatory) gesetzt werden muss.

Zum Verteilen der Anfragen können die Verteilungsparameter analog zu PQL in OdysseusScript festgelegt werden. Bei Ausführung der Anfragen durch die Coach Application ist dies jedoch nicht möglich, da die Coach Application nur SportsQL an Odysseus senden soll und dennoch Einfluss auf die Verteilungsstrategie haben soll. Um dieses Problem zu lösen, gibt es die Möglichkeit, dass die Coach Application neben dem SportsQL auch den Namen einer Verteilungsstrategie an Odysseus senden kann. In Odysseus wird aus diesen Daten eine OdysseusScript-Anfrage erstellt, die zum einen das SportsQL und zum anderen die Verteilungsparameter in Abhängigkeit der zuvor festgelegten Verteilungsstrategie enthält. Das Klassendiagramm in Abbildung 6.10 zeigt dazu die verwendeten Schnittstellen und Klassen.

Um eine neue Verteilungsstrategie zu definieren, muss die Schnittstelle `ISportsQLDistributor` implementiert werden. Diese liefert zum einen den Namen der Strategie und zum anderen die Verteilungsparameter zurück. Die Implementierung kann dabei die Hilfsklasse `DistributionConfigHelper` benutzen, welche zahlreiche Methoden anbietet, um Verteilungsparameter festzulegen. Die Registry `SportsQLDistributorRegistry` registriert die verschiedenen Verteilungsstrategien. Die Methode `addSportsQLDistributorConfig` der Registry liefert zu einer SportsQL-Anfrage und einer Verteilungsstrategie eine generierte OdysseusScript-Anfrage zurück. Beispielsweise wird bei der Verteilungsstrategie *recovery_active_standby* das in Listing 6.3 beschriebene OdysseusScript generiert.

```
1  @SportsQL(
2                  gameTypes = { GameType.SOCCER },
3                  statisticTypes = { StatisticType.PLAYER },
4                  name = "sprints",
5                  parameters = {
6                  @SportsQLParameter(name = "time", parameterClass =
                       ↪ SportsQLTimeParameter.class, mandatory =
                       ↪ false),
7                  @SportsQLParameter(name = "space", parameterClass =
                       ↪  SportsQLSpaceParameter.class, mandatory =
                       ↪ false)
8                  })
9  public class SprintsPlayerSportsQLParser implements ISportsQLParser
       ↪ {
10 ...
11 }
```

Listing 6.2: SprintsPlayerQuery

```
1  #PARSER SportsQL
2  #CONFIG DISTRIBUTE true
3  #PEER_ALLOCATE ROUNDROBIN
4  #PEER_PARTITION OPERATORSETCLOUD
5  #PEER_MODIFICATION REPLICATION 2
6  #PEER_MODIFICATION RECOVERY_ACTIVESTANDBY
7  #PEER_POSTPROCESSOR FORCELOCALSOURCES
8  #PEER_POSTPROCESSOR MERGE
9  #ADDQUERY
10 ///SportsQL-Anfrage
```

Listing 6.3: Verteilunggsstrategie für Active-Standby Recovery

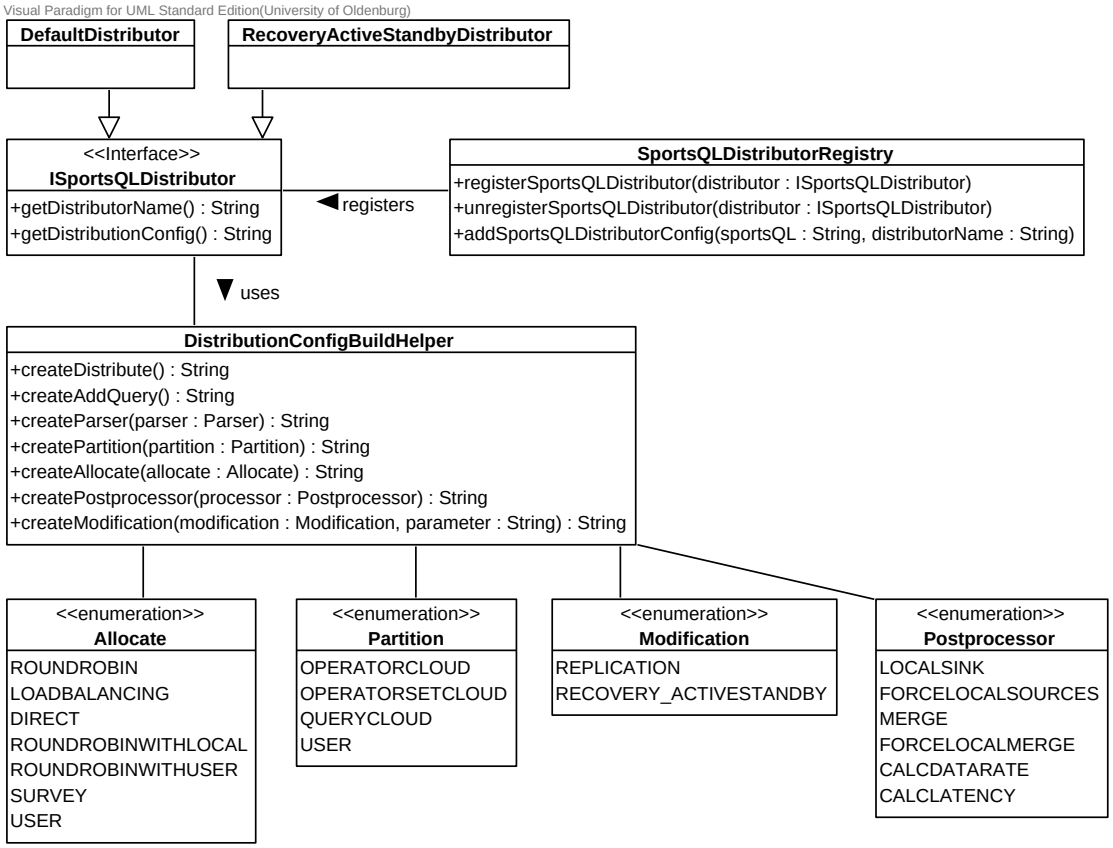


Abbildung 6.10: Verteilung von SportsQL-Anfragen

6.5 Kommunikation

Dieser Abschnitt befasst sich damit, wie die Kommunikation zwischen den einzelnen Komponenten innerhalb von Herakles umgesetzt wurde. Hierbei muss zum einen betrachtet werden, dass die einzelnen Odysseus-Peers untereinander kommunizieren müssen, um beispielsweise ein Recovery durchzuführen. Zum anderen muss die Kommunikation zwischen einem einzelnen Odysseus-System und den GUI-Anwendungen (Coach, Monitoring, Developer Application) beschrieben werden.

6.5.1 JXTA

Die Kommunikation zwischen den einzelnen Odysseus-Peers wird mit der bereits in Odysseus verwendeten JXTA-Bibliothek[3] umgesetzt. Für mehr Informationen zu JXTA sei auf 2.11 hingewiesen. Bei der Kommunikation mit JXTA muss unterschieden werden, ob ein Peer nur mit einem einzigen anderen Peer oder mit allen Peers kommunizieren möchte.

[3] `https://jxta.kenai.com/`

Für den ersten Fall sieht JXTA ein Konzept vor, bei dem die Kommunikation zwischen zwei Peers mit Hilfe einer Message umgesetzt wird. Dieses Konzept wird z. B. beim Recovery verwendet, um einem Peer mitzuteilen, dass er die Ergebnisse einer Anfrage nicht mehr senden soll. Die Abbildung 6.11 zeigt dieses Konzept beispielhaft, wie es im Recovery umgesetzt wird.

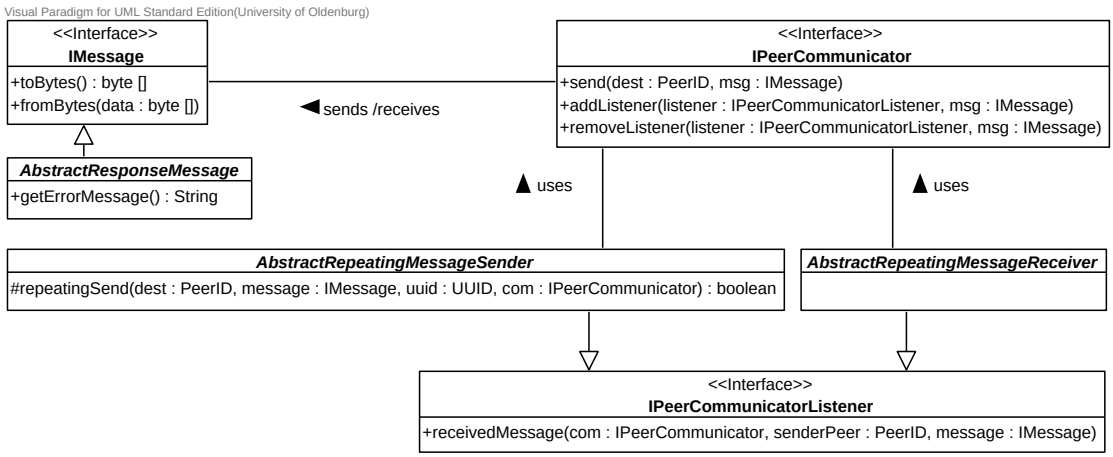


Abbildung 6.11: Senden einer Nachricht in Odysseus

Jede Nachricht muss von `IMessage` abgeleitet werden, wobei es für Nachrichten, die als Antwort dienen, eine abstrakte Klasse `AbstractResponseMessage` gibt. Die Antworten haben dabei in der Regel nur den Zweck, dem Sender mitzuteilen, ob die Nachricht angekommen ist. In einigen Fällen wird die Antwort auch dazu verwendet, um den Sender darüber in Kenntnis zu setzen, ob eine Operation erfolgreich durchgeführt werden konnte.

Für das Senden und Empfangen einer Nachricht ist die Schnittstelle `IPeerCommunicator` verantwortlich. Die abstrakte Klasse `AbstractRepeatingMessageSender` greift mit der Methode `repeatingSend` auf den `IPeerCommunicator` zu, um eine Nachricht mehrfach zu versenden. Dadurch soll die Wahrscheinlichkeit erhöht werden, dass die Nachricht beim Empfänger ankommt. Die abstrakte Klasse `AbstractRepeatingMessageReceiver` empfängt die wiederholt gesendeten Nachrichten und sorgt dafür, dass die selbe Nachricht nicht mehrfach weiterverarbeitet wird. Zum Empfangen einer Nachricht dient der Listener `IPeerCommunicatorListener`, welcher in der Schnittstelle `IPeerCommunicator` registriert werden kann. Nachdem eine Nachricht empfangen wurde, wird unmittelbar eine Antwort gesendet, um dem Sender mitzuteilen, dass die Nachricht angekommen ist. Daher muss die abstrakte Klasse `AbstractRepeatingMessageSender` auch den Listener `IPeerCommunicatorListener` implementieren. Wenn die Antwort innerhalb einer bestimmten Zeit empfangen wird, kann sichergestellt werden, dass die Nachricht angekommen ist und die Methode `repeatingSend` liefert `true` zurück.

Für den zweiten Fall bietet JXTA die Möglichkeit an, dass ein Peer ein Advertisement veröffentlichen kann, mit dem allen anderen Peers etwas mitgeteilt werden kann. Dieses Konzept bietet sich z. B. an, um den anderen Peers den Webservice-Port oder die Allokator-Strategie vom Recovery mitzuteilen. Die Gültigkeit von Advertisements kann dabei von den Peers gesteuert werden, so dass auch neue Peers bereits veröffentlichte Advertisements erhalten. Während beispielsweise das `Webservice-`

`Advertisment` nicht mehr gültig sein soll, wenn der dazu gehörende Peer nicht mehr erreichbar ist, soll das `RecoveryAllocatorAdvertisement` unbegrenzte Gültigkeit haben. Die Abbildung 6.12 zeigt die wesentlichen Schnittstellen beim Veröffentlichen eines Advertisements.

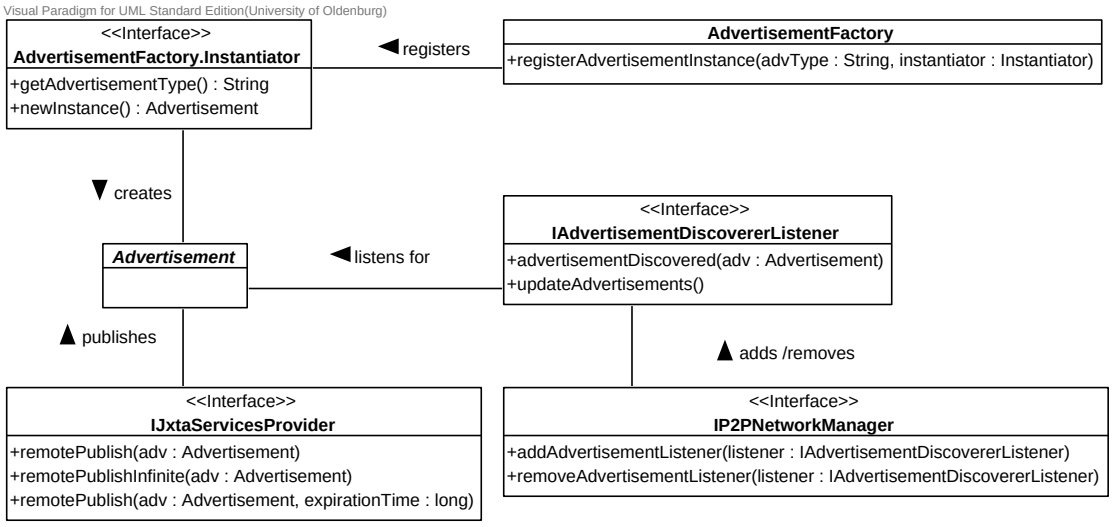


Abbildung 6.12: Veröffentlichung eines Advertisements in Odysseus

Für ein neues Advertisement muss es zum einen eine Implementierung der abstrakten Klasse `Advertisement` geben und zum anderen eine Implementierung von `AdvertisementFactory.Instantiator`, welche für das Erstellen des neuen Advertisements verantwortlich ist. Zum Veröffentlichen eines Advertisements kann die Schnittstelle `IJxtaServicesProvider` verwendet werden. Dabei muss auch die Gültigkeit des Advertisements festgelegt werden. Um Advertisements zu empfangen, kann der Listener `IAdvertisementDiscovererListener` verwendet werden, welcher mit Hilfe der Schnittstelle `IP2PNetworkManager` registriert werden kann. Des Weiteren muss zum Veröffentlichen und Empfangen eines Advertisements die Implementierung von `AdvertisementFactory.Instantiator` in der Klasse `AdvertisementFactory` registriert werden.

6.5.2 REST

Damit die GUI-Anwendungen mit Odysseus kommunizieren können, wurde eine Representational State Transfer (REST)-Schnittstelle implementiert. Sie bietet zahlreiche Ressourcen, die durch eindeutige URIs identifziert werden, über einen Webserver an. Somit ist es beispielsweise möglich, dass eine externe Anwendung eine Ressource ansprechen kann, mit der eine Anfrage erzeugt wird. Die Abbildung 6.13 zeigt, wie die REST-Schnittstelle in Odysseus implementiert wurde.

Es gibt eine zentrale Klasse `RestService`, mit der die REST-Schnittstelle gestartet oder gestoppt werden kann. Falls beim Starten der Port 9679 schon belegt ist, wird dieser solange hochgezählt, bis ein freier Port gefunden wird. Um Server-Ressourcen zu der REST-Schnittstelle hinzuzufügen, muss die Schnittstelle `IRestProvider` implementiert werden. Eine Server-Ressource bietet dabei i. d.

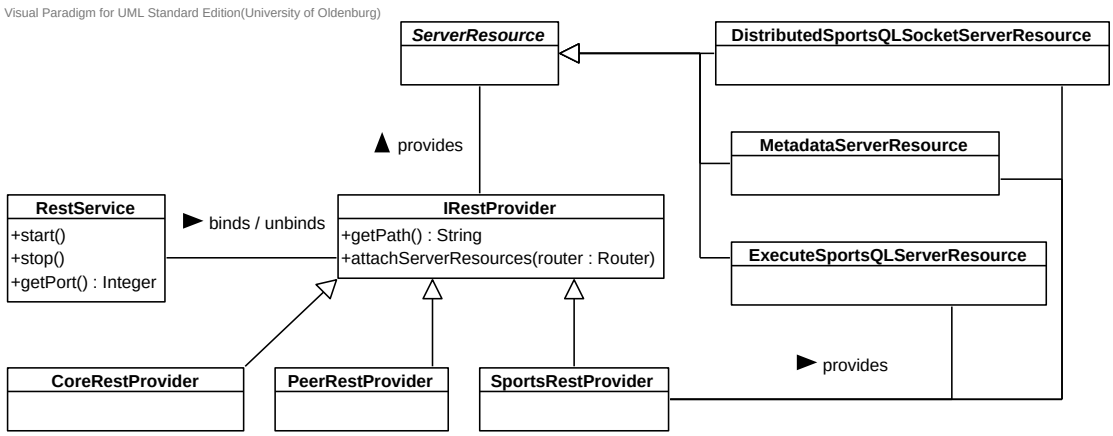


Abbildung 6.13: REST Schnittstelle in Odysseus

R. eine Methode an, mit der eine bestimmte Operation ausgeführt werden kann. Des Weiteren muss die Schnittstelle zurückliefern, unter welchem Pfad sich die Server-Ressourcen befinden.

Für Herakles wurden drei `IRestprovider` implementiert, die die folgenden Server-Ressourcen anbieten:

- CoreRestprovider
  - LoginServerResource: Einloggen in das Odysseus-System
  - AddQueryServerResourcen: Anfrage erstellen
  - StartQueryServerResource: Anfrage starten
  - StopQueryServerResource: Anfrage stoppen
  - RemoveQueryServerResource: Anfrage entfernen
  - CreateSocketServerResource: Socket öffnen, für die Ergebnisse einer Anfrage
- PeerRestProvider
  - PingMapServerResource: Socket für die Daten einer Ping-Map öffnen
  - GetSharedQueryIdsServerResource: Alle SharedQuery-Ids anfordern
  - GetSharedQueryServerResource: Daten zu einer verteilten Anfrage anfordern
  - GetLocalQueriesServerResource: Daten zu den lokalen Anfragen einer verteilten Anfrage anfordern
- SportsRestProvider
  - ExecuteSportsQLServerResource: SportsQL ausführen und verteilen
  - DistributedSportsQLSocketServerResource: Socket öffnen, für die Ergebnisse einer verteilten SportsQL-Anfrage

– MetadataServerResource: Daten vom DDC abfragen

Zur Implementierung der REST-Schnittstelle wurde die Bibliothek Restlet[4] verwendet. Für die Wahl der Bibliothek war ausschlaggebend, dass eine spezielle Version für *OSGi*-Umgebungen angeboten wird. Des Weiteren bietet die Bibliothek mit Jackson[5] eine Erweiterung an, der eine automatische Serialisierung von Java-Objekten zu JSON ermöglicht. Neben Restlet gibt es noch zahlreiche weitere Bibliotheken, wie z.B. Jersey[6] oder RESTEasy[7], die auch hätten verwendet werden können.

6.6 Load Balancing

In diesem Abschnitt wird die Umsetzung des in Kapitel 5.5 beschriebenen Load-Balancing Konzepts erläutert. Das Klassendiagramm in Abbildung 6.14 zeigt, wie die grundlegende Struktur umgesetzt wurde: Um die Austauschbarkeit der wesentlichen Komponenten zu gewährleisten, wurde überall das Strategie-Entwurfsmuster eingesetzt. Es gibt drei Schnittstellen, die jeweils eine Kernaufgabe des Load-Balancings repräsentieren und in einer konkreten Implementierung umgesetzt werden müssen.

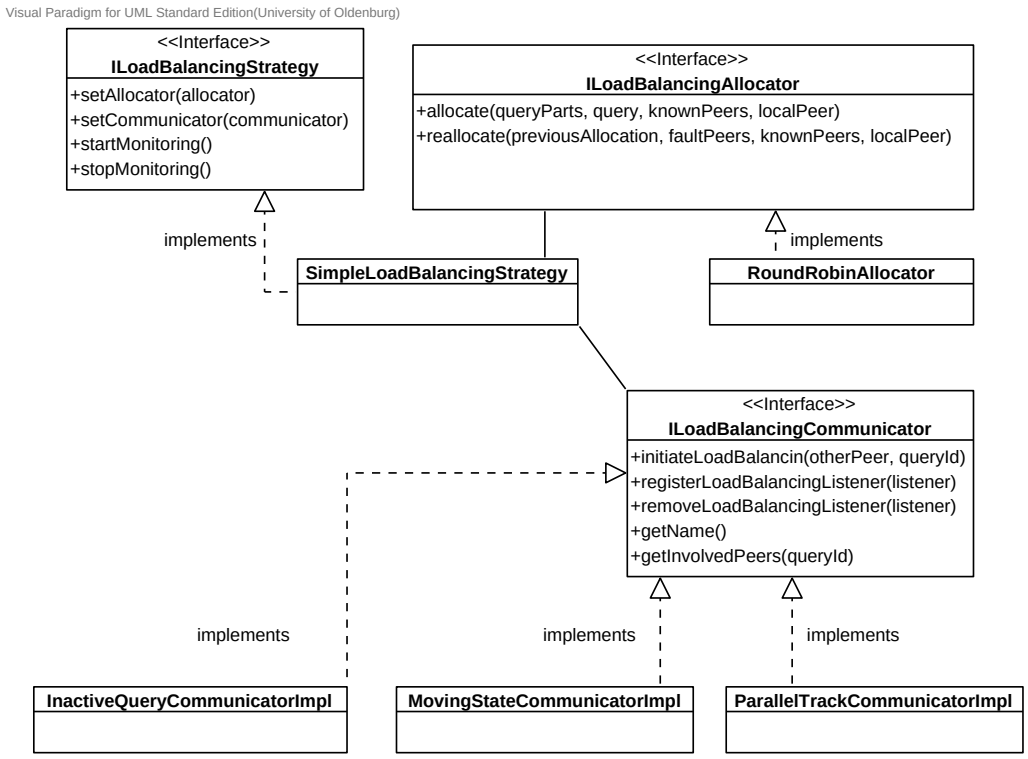


Abbildung 6.14: Überblick Struktur Load-Balancing

[4] http://restlet.com/
[5] http://jackson.codehaus.org/
[6] https://jersey.java.net/
[7] http://resteasy.jboss.org/

Zentrales Element ist dabei die Strategie, die Referenzen auf den verwendeten Allokator und Kommunikator hält. Durch eine entsprechende Erweiterung der Strategie erlaubt dies prinzipiell, dass die Strategie als übergeordnete Instanz je nach Anwendungsfall einen anderen Allokator und einen anderen Kommunikator auswählt. Über die Schnittstelle `ILoadBalancingStrategy` können eigene Load-Balancing Strategien entwickelt werden. Diese Schnittstelle gibt vier Methoden vor: Mit `setAllocator` wird der verwendete Allokator und mit `setCommunicator` der verwendete Kommuniaktor gesetzt. Die Methode `startMonitoring` startet die Überwachung eines Peers und `stopMonitoring` beendet die Überwachung eines Peers. Diese Schnittstelle wird von der Klasse `SimpleLoadBalancingStrategy` implementiert.

Die Schnittstelle `ILoadBalancingAllocator` beschreibt die Methoden eines Allokators und wird von der Klasse `RoundRobinAllocator` implementiert. Ein Allokator muss die Methoden `allocate` und `reallocate` implementieren. Mit `allocate` wird die Allokation einer Teilanfrage angestoßen, d.h. es wird ein Slave-Peer gesucht, der diese Teilanfrage aufnehmen kann. Die Methode `reallocate` kann genutzt werden falls eine erste Allokation fehlgeschlagen ist. Sie erlaubt den Ausschluss bestimmter Peers, zum Beispiel weil diese sich in der Vergangenheit als unzuverlässig erwiesen haben.

In der Schnittstelle `ILoadBalancingCommunicator` werden die Methoden eines Kommunikators definiert: Mit `initiateLoadBalancing` kann ein Load-Balancing-Vorgang angestoßen werden. Über `registerLoadBalancingListener` und `removeLoadBalancingListener` können sich andere Klassen gemäß Beobachter-Entwurfsmuster als Beobachter ein- oder austragen, die nach erfolreichem Load-Balancing benachrichtigt werden. Die Methode `getName` gibt den Namen des Kommunikators zurück, damit dieser von der Strategie zugeordnet werden kann. Mit `getInvolvedPeers` kann eine Liste von betroffenen Peers angefordert werden, die beim Verschieben einer Teilanfrage in den Prozess mit einbezogen werden müssen. Basierend auf dieser Schnittstelle wurden in den Klassen `MovingStateCommunicatorImpl`, `ParallelTrackCommunicatorImpl` und `InactiveQueryCommunicatorImpl` drei verschiedene Kommunikatoren umgesetzt.

Im Folgenden wird zunächst die Umsetzung der Strategie erläutert, danach die des Allokators. Zum Schluss wird die Umsetzung der Kommunikatoren beschrieben.

6.6.1 Umsetzung der Strategie

Wie in Abschnitt 5.5 beschrieben, überwacht eine Load-Balancing Strategie einen Peer und löst gegebenenfalls einen Load-Balancing-Vorgang aus. Außerdem wählt die Strategie die zu übertragende Teilanfrage aus. Jede Load-Balancing Strategie muss die Schnittstelle `ILoadBalancingStrategy` implementieren.

Die oben beschriebene Schnittstelle wird von der Klasse `SimpleLoadBalancingStrategy` umgesetzt: Wird `startMonitoring` aufgerufen, erzeugt die Klasse einen zusätzlichen Überwachungs-Prozess. Dadurch entsteht dem Gesamtsystem kaum zusätzliche Last solange kein Load-Balancing ausgelöst wird. Der Ablauf innerhalb des Prozesses ist im Aktivitätsdiagramm in Abbildung 6.15 dargestellt: Der Prozess überprüft in regelmäßigen, einstellbaren Abständen die aktuelle Prozessorlast des jeweiligen Peers. Liegt die Last über einem festgelegten Schwellwert, wählt die Strategie in der Methode `getQueryToRemove` die erste auf dem Peer installiere Anfrage zur Übertragung aus. Eine intelligentere Auswahl wäre zwar sinnvoll, wurde aber aus Zeitgründen im Projekt

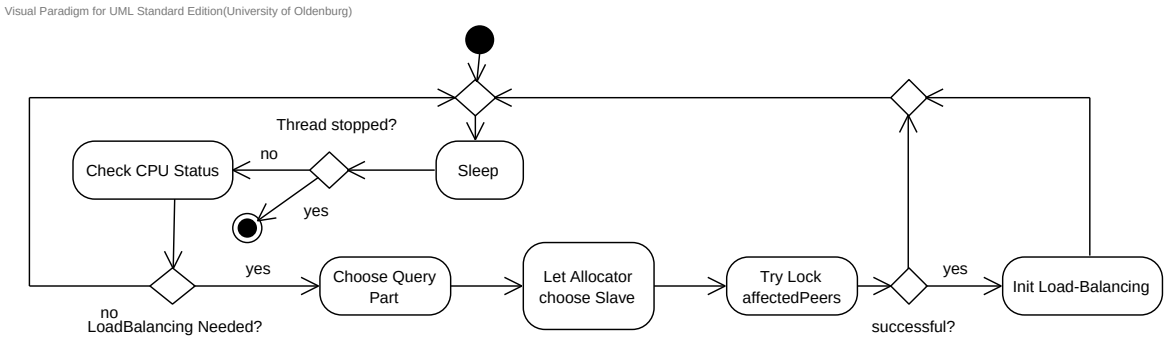


Abbildung 6.15: Aktivitätsdiagramm Überwachungs-Prozess in der Load-Balancing Strategy

nicht umgesetzt. Diese Art der Auswahl reicht allerdings aus, um die generelle Machbarkeit von Load-Balancing in Odysseus P2P zu zeigen. Außerdem wird in dieser Strategie immer eine komplette Teilanfrage ausgewählt. Im Gegensatz zu einzelnen Operatoren verfügen diese Teilanfragen bereits über (relative) Quellen und Senken, wie etwa Jxta-Sender oder Jxta-Receiver, was den notwendigen Aufwand zum Verschieben vermindert. Zudem wird so die vom Nutzer eingestellte Fragmentierungseinstellung beibehalten, die andernfalls übergangen werden würde.

Nach der Auswahl einer Teilanfrage wird diese an die Methode `allocate` des jeweiligen Allokators übergeben, der, wie im folgenden Abschnitt beschrieben, einen übernehmenden Slave für diese Anfrage findet. Im nächsten Schritt werden alle an der Teilanfrage beteiligten Peers ermittelt und für weitere Load-Balancing-Vorgänge gesperrt. Dadurch ist gewährleistet, dass Änderungen an laufenden Anfragen auf den Upstream- und Downstream-Peers nicht durch weitere Vorgänge beeinflusst werden. Im schlimmsten Fall könnten sonst zwei Vorgänge gleichzeitig auf die gleichen Operatoren zugreifen und diese unterschiedlich modifizieren.

Ist eine Teilanfrage ausgewählt, ein Slave gefunden und alle beteiligten Peers gesperrt, kann die eigentliche Übertragung beginnen. Dazu werden diese Informationen an die Methode `initiateLoadBalancing` des jeweiligen Kommunikators übergeben und damit der Load-Balancing Vorgang angestoßen.

Nach erfolgreichem oder fehlgeschlagenem Load-Balancing wird vom Kommunikator die Methode `notifyLoadBalancingFinished` der Strategie aufgerufen und mit einem Übergabeparameter angegeben, ob der Vorgang erfolgreich war. Die Strategie gibt nun alle vorher gesperrten Peers wieder frei und beginnt wieder mit der regelmäßigen Überwachung der Prozessorlast.

6.6.2 Umsetzung des Allokators

Gemäß dem Konzept in Abschnitt 5.5 ist es die Aufgabe eines Allokators, einen geeigneten Peer zu finden, der die zu verschiebende Anfrage übernehmen kann. Dazu wurde in der Projektgruppe ein Allokator in der Klasse `RoundRobinAllocator` implementiert, der diese Aufgabe nach dem Round-Robin-Verfahren durchführt. Round-Robin wurde gewählt, weil es sich dabei um ein einfach

zu implementierendes Verfahren eignet, welches dennoch in der Lage ist, die generelle Machbarkeit des Load-Balancings zu zeigen.

Streng genommen implementiert dieser Allokator die Methoden der Schnittstelle `ILoadBalancingAllocator` allerdings nicht direkt, sondern erbt von `AbstractRoundRobinAllocator`, welcher wiederum die Schnittstelle implementiert. Die abstrakte Klasse enthält die eigentliche Auswahllogik, während die abgeleitete Klasse noch die Methode `determineConsideredPeerIDs` enthält, mit der bestimmte Peers (z.B. der lokale Peer) von der Auswahl ausgeschlossen werden können. Dieser Zwischenschritt wurde gewählt, damit kleinere Anpassungen der Round-Robin-Allokation ohne komplette Neuimplementierung des Allokators durchgeführt werden können und die Wartbarkeit erhöht wird.

Wird von der jeweiligen Strategie die `allocate`-Methode des Allokators aufgerufen, wird zunächst in der Methode `determineConsideredPeerIDs` eine Liste gültiger Peers erzeugt, die als Slave in Frage kommen. Danach wird nach dem Round-Robin-Verfahren der nächste Slave aus der Liste der gültigen Peers ausgewählt und an die Strategie zurückgegeben. In der Methode `reallocate` wird analog verfahren.

6.6.3 Umsetzung der Kommunikatoren

Wie im Konzept in Abschnitt 5.5 erläutert sind Kommunikatoren für die eigentliche Durchführung eines Load-Balancing Vorgangs zuständig. Dazu wurden in dieser Projektgruppe drei verschiedene Kommunikatoren auf Basis der Schnittstelle `ILoadBalancingCommunicator` implementiert: Die Klasse `ParallelTrackCommunicatorImpl` setzt das im Konzept beschriebene Parallel-Track-Verfahren und die Klasse `MovingStateCommunicatorImpl` das beschriebene Moving-State-Verfahren um. Zusätzlich wurde in der Klasse `InactiveQueryCommunicatorImpl` ein Kommunikator für die Übertragung inaktiver, also nicht laufender, Anfragen entwickelt.

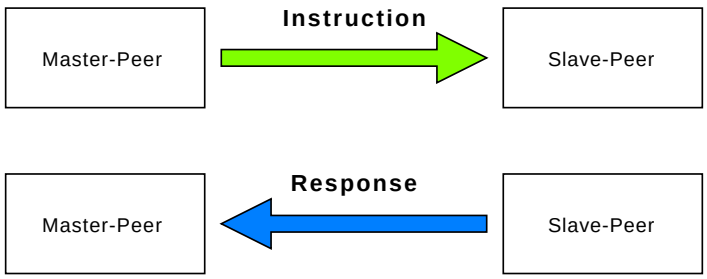


Abbildung 6.16: Instruction- und Response-Nachrichten im Load-Balancing

Der jeweilige Kommunikator implementiert neben der Kommunikator-Schnittstelle auch die von Odysseus P2P vorgegebene Schnittstelle `IPeerCommunicatorListener`. Diese Schnittstelle ermöglicht den Empfang von Nachrichten innerhalb eines Odysseus P2P Netzwerks. Mit Hilfe solcher Nachrichten implementiert jeder Kommunikator sein eigenes Protokoll, welches die einzelnen Phasen des Vorgangs abbildet. Bei den Nachrichten wird dabei in Instruction- und Response-Nachrichten unterschieden. Wie in Abbildung 6.16 veranschaulicht, werden Nachrichten von einem Master zu einem

Slave (aber auch zu einem Upstream- oder Downstream-Peer) als Instruction-Nachricht bezeichnet, während alle Nachrichten an den Master als Response-Nachricht bezeichnet werden.

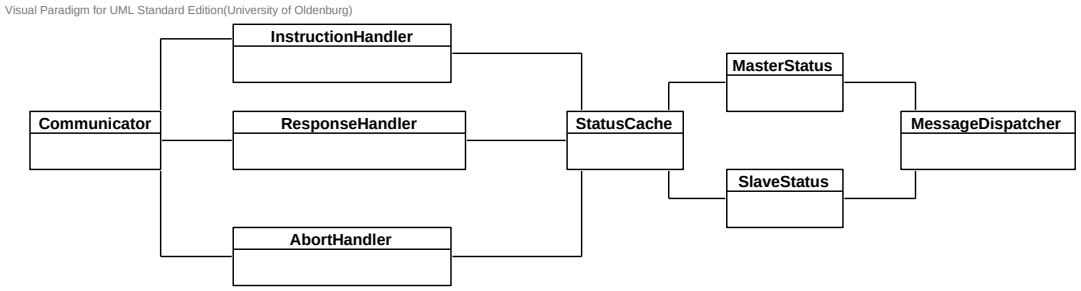


Abbildung 6.17: Struktureller Aufbau eines Kommunikators im Load-Balancing

Diese Aufteilung der Nachrichten ermöglicht eine Strukturierung der Kommunikator-Protokolle in eine Master- und eine Slave-Seite und erhöht damit die Übersichtlichkeit. Alle drei Kommunikatoren und die damit verknüpften Protokolle sind strukturell analog aufgebaut, wie in Abbildung 6.17 gezeigt. Die Master-Seite empfängt die Response-Nachrichten in einem `ResponseHandler` und sendet je nach Protokollphase Instruction-Nachrichten an Slave-Peer, Upstream- und Downstream-Peers. Die Slave-Seite empfängt diese Anweisungen in einem `InstructionHandler`, führt diese aus und sendet eine Response-Nachricht (in der Regel Erfolg oder Misserfolg) an den Master. Im Fehlerfall werden außerdem Abort-Nachrichten verwendet, die in einem `AbortHandler` ausgewertet werden und auf den betroffenen Peers ein Rollback des Load-Balancings einleiten.

Um im Fehlerfall ein Rollback zu ermöglichen und die einzelnen Phasen eines Kommunikator-Protokolls zu koordinieren, hat jeder Kommunikator sowohl für den Master, als auch für den Slave, Upstream- und Downstream-Peers eine eigene Zustandsklasse (`MasterState` und `SlaveState`), die wesentliche Informationen zum aktuellen Vorgang speichert. Die Verwaltung dieser Status übernimmt ein zentraler `StatusCache`. Außerdem existiert zu jedem Kommunikator ein passender `MessageDispatcher`, mit dem die protokollspezifischen Nachrichten versendet werden können.

Dieser analoge Aufbau aller Kommunikatoren hilft dabei, bestehende Kommunikatoren zu verstehen und schnell neue Protokolle zu implementieren, da grundlegende Strukturen wiederverwendet werden können und nur an das jeweilige Protokoll angepasst werden müssen. Im Folgenden werden nun die Eigenheiten der einzelnen Kommunikatoren erläutert.

6.6.3.1 Parallel-Track-Kommunikator

Im Wesentlichen entspricht der Aufbau des Parallel-Track-Kommunikators dem im voringen Abschnitt vorgestellten generischen Kommunikator-Aufbau. Wie im Konzept erläutert, basiert der Parallel-Track-Kommunikator darauf, dass sowohl der alte, als auch der neue Datenstrom parallel verarbeitet werden und ein Synchronisationsoperator jeweils nur einen der Datenströme weiterleitet. Dieser Synchronisationsoperator ist im Projekt in der Klasse `LoadBalancingSynchronizerPO` realisiert.

Zunächst leitet dieser Operator nur den auf Port 0 eingehenden Datenstrom (alter Datenstrom) weiter. Der neue Datenstrom wird an Port 1 empfangen und die Zeitstempel der Tupel werden mit denen des anderen Datenstroms verglichen. Um einen Datenverlust möglichst zu vermeiden, stellt der Operator genau dann vom alten auf den neuen Datenstrom um, wenn beide Zeitstempel übereinstimmen. Für den Fall, dass diese Situation nie eintritt kann im Operator eine Zeitspanne eingestellt werden nach der auf jeden Fall auf den neuen Datenstrom umgeschaltet wird. Dieser zeitstempelbasierte Ansatz hat den Nachteil, dass unter Umständen mehrere Tupel mit gleichem Zeitstempel vorkommen könnnen. In einem solchen Fall würde der Operator ggf. einige Tupel überspringen, so dass ein kleiner Datenverlust auftritt. Alternativ wäre ein Ansatz mit eigenen Markierungs-Punctuations denkbar, die in beide Datenströme eingefügt werden und dann vom Synchronisationsoperator interpretiert werden. Auf diesen Ansatz hat die Projektgruppe allerdings verzichtet, da zum Zeitpunkt der Entscheidung noch nicht alle Odysseus-Operatoren Punctuations unterstützt haben.

6.6.3.2 Moving-State-Kommunikator

Der Moving-State-Kommunikator implementiert die im Konzept beschriebene Moving-State-Methode, bei der der Datenstrom zunächst gepuffert wird und dann die Zustände der Operatoren vom Master zum Slave übertragen werden.

Die Pufferung der Datenströme erfolgt dabei nicht in einem eigenen Operator, sondern in den Subscriptions vor den Sender-Operatoren der Upstream-Peers. Dies hat den Vorteil, dass kein zusätzlicher Operator eingebaut werden muss, wodurch unter Umständen ein Datenverlust auftreten könnte. Um die puffernde Subscription mit dem neuen Datenstrom zu verbinden, kann diese Subscription ohne Neuverknüpfung auf den neuen Sender-Operator umgeleitet werden. Zum Puffern müssen die Subscriptions allerdings vom Typ `ControllablePhysicalSubscription` sein.

Die größte Herausforderung des Moving-State-Ansatzes stellt die Übertragung der Operatorzustände dar. Zunächst ist es notwendig, die zustandsbehafteten Operatoren überhaupt erkennen zu können. Da Odysseus dazu keine eigene Möglichkeit bietet, wurde im Projekt die Schnittstelle `IStatefulPO` entwickelt, welche von zustandsbehafteten Operatoren implementiert werden muss. Diese Schnittstelle hat eine Doppelfunktion: Zum einen markiert sie zustandsbehaftete Operatoren, so dass diese vom Algorithmus erkannt werden können. Zum anderen enthält die Schnittstelle die Methoden `getState` und `setState`, die zum Auslesen und Setzen von Status genutzt werden. Da die Operatoren heterogen sind und jeder Status operatorindividuell unterschiedlich sein kann, ist lediglich festgelegt, dass ein Operatorstatus ein serialisierbares Objekt ist. Dadurch kann jeder Operator einen individuellen Status implementieren und gleichzeitig kann die Übertragung der Status über das Netzwerk einheitlich erfolgen. In der Projektgruppe wurde die `IStatefulPO` Schnittstelle für Fenster-, Aggregations- und teilweise für Join-Operatoren, sowie für den AssureHeartbeat-Operator und die StateMap umgesetzt.

Um Operatoren richtig zuordnen zu können, muss die Reihenfolge der gefundenen Operatoren auf dem Master mit der Reihenfolge der Operatoren auf dem Slave übereinstimmen. Dazu wird auf beiden Peers die gleiche Methode zum Finden zustandsbehafteter Operatoren verwendet. Diese durchsucht die Operatorbäume der jeweiligen Anfragen iterativ nach Operatoren, die `IStatefulPO` implementieren. Außerdem findet eine einfache Überprüfung statt: Bei der Zustandsübertragung wird der Operatortyp mitgesendet. Stimmen diese auf Master und Slave nicht überein, ist die Reihenfolge unterschiedlich und es liegt ein Fehler vor.

Die Zustandsübertragung selbst erfolgt über eigene Sender/Receiver-Paare. Sender und Receiver haben nur eine begrenzte Puffergröße, Operatorzustände können aber zum Teil sehr groß werden. Daher werden die Zustände bei Bedarf zunächst in Pakete unterteilt und auf der Empfängerseite wieder zusammengesetzt.

6.6.3.3 Inactive-Query-Kommunikator

Zusätzlich zum Parallel-Track- und Moving-State-Kommunikator, die auf Basis der jeweiligen Konzepte die Verschiebung von laufenden Anfragen ermöglichen, wurde mit dem Inactive-Query-Kommunikator ein Kommunikator für inaktive Anfragen entwickelt.

Der Aufbau des Inactive-Query-Kommunikators entspricht dem der beiden anderen Kommunikatoren, allerdings ist das Protokoll im Vergleich stark vereinfacht. Im Gegensatz zu laufenden Anfragen muss bei inaktiven Anfrage kein Tupelverlust verhindert werden. Entsprechend ist keine Synchronisierung oder Pufferung notwendig. In diesem Kommunikator wird lediglich die zu verschiebende Anfrage als PQL-Code an den Slave geschickt und von diesem installiert. Danach werden die logischen Anfragepläne auf den Upstream- und Downstream-Peers angepasst. Nach diesem Schritt ist das Load-Balancing bereits beendet und der Kommunikator hat seine Aufgabe erfüllt.

6.7 Recovery

In diesem Abschnitt wird die Umsetzung des in Abschnitt 5.6 beschriebenen Konzeptes für ein Recovery in verteilten DSMS erläutert. Die Struktur der Recovery-Implementierung ist dabei in Abbildung 6.18 dargestellt.

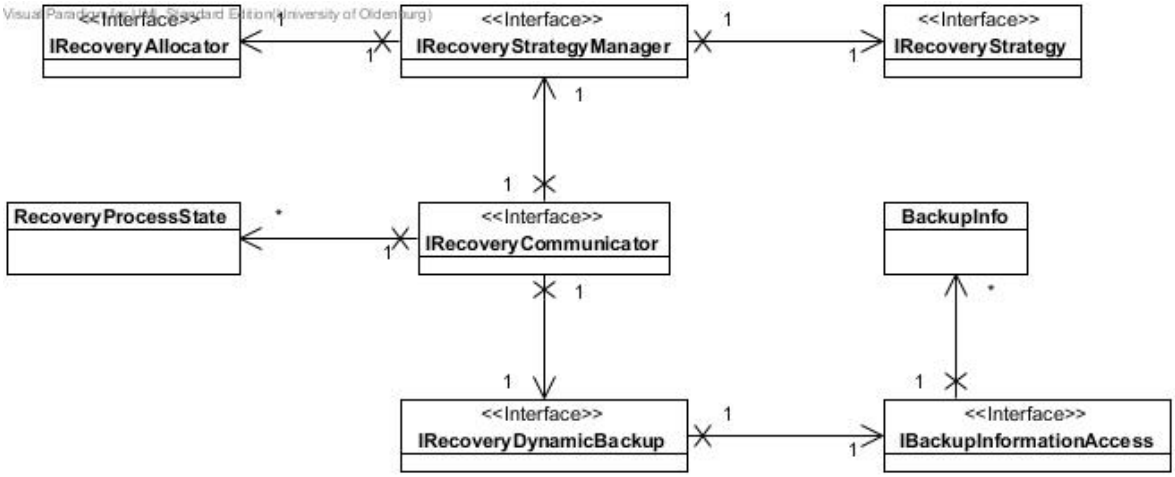


Abbildung 6.18: Struktur der Recovery-Implementierung

Informationen über verteilte Anfragen werden in `BackupInfo`-Objekte gespeichert. Auf diese kann über einen `IBackupInformationAccess` zugegriffen werden. Näheres zu den Backup-Informationen folgt in Abschnitt 6.7.1.

Verwendung finden sie in einem `IRecoveryDynamicBackup`, der fester Bestandteil der Kommunikationsschnittstelle, dem `IRecoveryCommunicator`, ist. Sie kümmert sich darum, den Pro-

zess zu starten, dynamisch einen Backup-Peer zu finden. Dabei initiiert sie das Suchen nach dem zuständigen Peer für das Recovery und greift auf die Backup-Information zu, um die ausgefallenen Query-Parts auf dem neuen Peer installieren zu können.

Die Kommunikationsschnittstelle wird in Abschnitt 6.7.4 beschrieben. Ein weiterer Bestandteil der Kommunikationsschnittstelle besteht aus `RecoveryProcessState`-Objekten, mit denen Informationen über einzelne Recovery-Prozesse zusammengefasst werden (Abschnitt 6.7.1.2).

Die `IRecoveryStrategy` bestimmt darüber auf welche Weise das Recovery durchgeführt wird. Die Strategie ist austauschbar und der `IRecoveryStrategyManager` ist dafür zuständig, die Recovery-Strategie (`IRecoveryStrategy`) zu wählen. Die beiden umgesetzten Strategien für das Recovery *„Simple“* (auch bekannt als *Amnesia*) und *Active-Standby* werden in den Abschnitten 6.7.5 und 6.7.6 beschrieben.

Ein `IRecoveryAllocator` ist dafür zuständig bei beispielsweise einem Peer-Ausfall einen neuen Peer zu bestimmen, welcher die Aufgaben des ausgefallenen Peers übernehmen kann. Genauer beschrieben wird die Umsetzung in Abschnitt 6.7.3. Abgeschlossen wird der Abschnitt mit einer Betrachtung eines speziellen Fehlerfalles für unser Projekt: der Ausfall des Peers, der mit dem Tablet verbunden ist (Abschnitt 6.7.7).

6.7.1 Speichern von Backup-Informationen

Fällt ein Peer aus, sind die Informationen über Anfragen, die auf diesem installiert waren, nicht mehr zugänglich. Somit müssen diese Informationen dezentral gespeichert werden, sodass Anfragen bei einem Ausfall auf einem anderen Peer neu installiert werden können. Dabei ist vor allem der PQL-Ausdruck und der aktuelle Zustand (z.B. „installiert“ oder „laufend“) entscheidend.

Die Informationen über eine Anfrage werden in `BackupInfo`-Objekten zusammengefasst und enthalten Folgendes:

- Den PQL-Ausdruck der Anfrage
- Den Zustand der verteilten Anfrage (z.B. „installiert“)
- Die ID der verteilten Anfrage (für lokale Anfragen irrelevant)
- Die Information, ob der Peer beim Verteilungsprozess als Master gesetzt wurde (für lokale Anfragen irrelevant)
- Die Verbindungsinformationen für die Sockets zum Handheld:
  - Die IP des Handhelds
  - Die IP des Host-Peers
  - Die IP des Sockets beim Host-Peer (irrelevant für Teilanfragen ohne Senken,die eine Socket-Verbindung aufbauen)

Alle `BackupInfo`-Objekte für einen Peer werden in einer Map verwaltet, wobei die lokalen IDs der Anfragen (vom `Executor` vergeben) die Schlüssel darstellen. Ein Eintrag im DDC hat die ID eines Peers als Schlüssel und seine (serialisierte) Map als Wert. Somit beinhaltet ein Eintrag im DDC alle verfügbaren Informationen über einen Peer.

6.7.1.1 Erstellen/Ändern von Backup-Informationen

Grundsätzlich ist jeder Peer für das Erstellen der Backup-Informationen über sich verantwortlich. Die Klasse `BackupInformationHelper (BIHelper)` horcht dabei auf den lokalen `Executor`:

- Wird eine Anfrage hinzugefügt, werden neue Backup-Informationen erstellt (bis auf die Attribute für verteilte Anfragen), im DDC gespeichert und das Verteilen des DDC wird angestoßen.
- Wird eine Anfrage entfernt, werden die entsprechenden Backup-Informationen aus dem DDC entfernt und das Verteilen des DDC wird angestoßen.
- Wird der Zustand einer Anfrage verändert (z.B. von „installiert“ auf „laufend“), werden die entsprechenden Backup-Informationen im DDC angepasst und das Verteilen des DDC wird angestoßen.

Für verteilte Anfragen horcht der `BIHelper` zusätzlich auf den Service `IQueryPartController`. Von ihm bekommt er Informationen darüber, ob der lokale Peer einer verteilten Anfrage als Master oder Slave zugeteilt wurde und, im Falle eines Slaves, welcher Peer der Master zu der verteilten Anfrage ist. Auf Grundlage dieser Informationen können die Backup-Informationen um die entsprechenden Attribute erweitert werden. Analog zu den Szenarien, die durch den `Executor` ausgelöst werden, werden die Backup-Informationen im DDC angepasst und das Verteilen des DDC wird angestoßen.

6.7.1.2 Entfernen von Backup-Informationen

Nachdem ein Peer-Ausfall bemerkt wurde, werden alle Backup-Informationen über den entsprechenden Peer ausgelesen. Sie werden dann zum einen für die Recovery verwendet und zum anderen vom dem Peer, der die Recovery durchführt, aus dem DDC entfernt.

6.7.2 Recovery-Statusobjekte

Es ist möglich, dass ein Recovery-Vorgang nicht funktioniert, wenn z.B. der ausgewählte Peer den Query-Part nicht installieren kann. Dies kommt z.B. dann vor, wenn der Peer die Quellen nicht hat und den ersten Query-Part einer Anfrage übernehmen soll. Für diesen Zweck wird der Status jedes Recovery-Vorgangs bei dem durchführenden Recovery-Peer gespeichert. Dazu gibt es die Klasse `RecoveryProcessState`. Jeder Status hat eine eindeutige UUID und speichert die PeerID des ausgefallenen Peers. Dazu werden diese in einem `RecoverySubProcessState` gespeichert, welche den Behandlungsvorgang für jeden ausgefallenen Query-Part abbilden. Dort sind alle Informationen gespeichert, die zum Recovery benötigt werden, also z.B. die sharedQueryID, evtl. die Verbindungsinformationen zum Tablet, etc. Außerdem wird hier eine Liste von Peers gespeichert, auf denen das Recovery nicht funktioniert hat. Kann also z.B. der zweite Peer den Query-Part nicht installieren, wird es bei einem anderen Peer nochmal versucht und dieser Peer wird auf die Liste der nicht möglichen Peers für diese Query-Part gesetzt.

6.7.3 Allokation und Reallokation

Das Recovery beinhaltet eine Schnittstelle für Allokatoren, so dass verschiedene Strategien implementiert werden können. Die Allokatoren basieren dabei auf Query-Parts und zum jetzigen Zeitpunkt sind die beiden folgenden Strategien umgesetzt:

- `RoundRobin` - Für jeden zu wiederherstellenden Query-Part wird reihum ein Peer (außer dem, der das Recovery anstößt) ausgewählt, um ihn zu installieren.
- `RoundRobinWithLocal` - Ist ein `RoundRobin`-Algorithmus, der den Peer mit einbezieht, der das Recovery anstößt.

Während eines Recovery-Prozesses werden Zustände für diesen gespeichert, sodass die Nachrichtengröße verringert werden kann. Der Zustand eines Recovery-Prozesses besteht aus den folgenden Komponenten:

- `RecoveryProcessState` - Der `RecoveryProcessState` ist der Hauptzustand. Dort sind zum einen die IDs der verschiedenen verteilten Anfragen und zum anderen die Zuordnung von Query-Part zu Peer hinterlegt. Letztere wird benötigt, wenn eine Reallokation durchgeführt werden soll.
- `RecoverySubProcessState` - Der Unterzustand wird pro Query-Part, der wiederhergestellt werden soll, erstellt. Ein solcher Unterzustand enthält u.a. die ID der verteilten Anfrage, den eigentlichen Query-Part und alle Peers, bei denen eine Installation des Query-Parts bereits fehlgeschlagen ist.

Über den Service `IRecoveryCommunicator` können diese Zustände an jeder Stelle innerhalb des Prozesses über eine eindeutige ID abgerufen werden. Somit müssen nur diese IDs übertragen werden.

Für den Fall, dass die Allokation bzw. die Installation des Query-Parts auf dem ausgewählten Peer fehlschlägt, wird eine Reallokation durchgeführt. Die Reallokation wird allerdings nur für den entsprechenden Query-Part durchgeführt. Die Grundlage für die Reallokation wird aus den Zuständen für diesen Query-Part ermittelt.

6.7.4 Kommunikationsprotokoll

Der Service `IRecoveryCommunicator` ist der zentrale Kommunikationspunkt für das Recovery. Dabei werden verschiedene Nachrichten und Empfangsbestätigungen von entsprechenden Sendern und Empfängern verarbeitet. Das Senden einer Nachricht erfolgt mehrfach bis eine Empfangsbestätigung eingegangen oder eine Zeitschranke überschritten ist.

6.7.4.1 Nachrichten

Es folgt eine kurze Beschreibung der verschiedenen Nachrichten innerhalb des Kommunikationsprotokolls. Für jeden Nachrichtentyp in der Form `Recovery<Type>Message` dient eine entsprechende `Recovery<Type>ResponseMessage` als Empfangsbestätigung, die im Fehlerfall eine entsprechende Fehlermeldung enthält. Eine Empfangsbestätigung ist notwendig, da die Nachrichten

grundsätzlich wiederholend gesendet werden. Es werden also so lange Nachrichten geschickt, bis die Bestätigung beim Sender angekommen ist. Kommt keine solche Antwort, wird das Senden erst nach einer bestimmten Zeitspanne, standardmäßig 60 Sekunden, eingestellt. Außerdem kann die Antwort zusätzliche Informationen enthalten, z.B. ob eine Anfrage erfolgreich installiert werden konnte.

- Die `RecoveryTupleSendMessage` steuert den Datenstrom zu dem ausgefallenen Peer und kann entweder als `HoldOnMessage` oder als `GoOnMessage` interpretiert werden.
  - Eine `HoldOnMessage` sorgt für ein temporäres Zwischenspeichern aller Datenstromelemente, die normalerweise an den Peer gesendet werden würden, der ausgefallen ist.
  - Eine `GoOnMessage` sorgt für eine Wiederaufnahme des Sendevorgangs zu dem ausgefallenen Peer (siehe `HoldOnMessage`).
- In einem dezentralen P2P-Netzwerk ist es möglich, dass mehrere Peers einen Peer-Ausfall feststellen. Um ein mehrfaches Recovery zu vermeiden, wird durch `RecoveryAgreementMessage`-Objekte bestimmt, wer das Recovery durchführen kann. Für den Fall, dass mehrere Peers das Recovery ausführen können, wird der Peer mit der höheren ID ausgewählt.
- In einer `RecoveryAddQueryMessage` ist der zu installierenden Query-Part in Form eines PQL-Ausdrucks enthalten.
- Die `RecoveryUpdatePipeMessage` aktualisiert Sender-Operatoren, die einen Datenstrom zu einem Peer mit einem wiederhergestellten Query-Part senden, oder Empfänger-Operatoren, die einen Datenstrom vom einem Peer mit einem wiederhergestellten Query-Part empfangen.
  - Eine `RecoveryUpdateReceiverMessage` sorgt für das Aktualisieren eines Empfängers. Da in der JXTA-Architektur die Empfänger für den Aufbau der Verbindung zuständig sind, ist diese Nachricht essentiell für Wiederaufnahme der Datenstromverarbeitung.
  - Eine `RecoveryUpdateSenderMessage` sorgt für das Aktualisieren eines Senders.

6.7.4.2 Fehlerbehandlung

Da es auch beim wiederholten Verschicken von Nachrichten dazu kommen kann, dass der entsprechende Peer nicht erreicht werden oder dieser die Nachricht nicht korrekt verarbeiten kann, wird im Folgenden beschrieben, was in den einzelnen Fällen (also je nach Schritt, bei dem eine Nachricht versendet wird) als Konsequenz umgesetzt wurde.

`sendHoldOnMessage`

Sollte eine `HoldOnMessage` nicht beim Empfänger ankommen, kommt es dazu, dass Tupel an einen ausgefallenen Peer weitergesendet werden und die Tupel somit verloren gehen, da sie nicht rechtzeitig zurückgehalten werden. In diesem Fall ist im Idealfall das Active-Standby aktiv und nur ein einzelner Ausfall zu verzeichnen, so dass die Tupel noch an einen weiteren Peer gesendet werden und kein Tupel unverarbeitet bleibt.

Eine umfangreiche Fehlerbehandlung erscheint nicht zweckmäßig, da man sonst Upstream ansetzen und dort bereits die Tupel vorhalten müsste und auch dies bietet keine Garantie dafür, dass nicht

einige Tupel verloren gehen. Der Aufwand steht also in keinem Verhältnis zum Ertrag. Im Normalfall sollte das Active-Standby (falls verwendet) als Absicherung genügen. Es wird eine Error-Log-Nachricht ausgegeben, um den Nutzer über das Auftreten des Fehlers zu informieren.

`sendGoOnMessage`

Die `GoOnMessage` hat den Zweck, dass ein Peer die Tupel wieder frei gibt, welche nach einer `HoldOnMessage` in einem Puffer zwischengespeichert wurden. Falls die `GoOnMessage` nicht beim Empfänger ankommt, werden die Tupel immer weiter in den Puffer geschrieben, obwohl diese eigentlich schon von dem anderen Peer (der die `GoOnMessage` sendet) weiterverarbeitet werden können. Um dies zu verhindern, wird ein Timeout-Protokoll eingesetzt, welches vorsieht, dass nach einer bestimmten Zeit, z.B. 1 Minute, automatisch die Tupel aus dem Puffer wieder freigegeben werden. In dieser Zeit sollte der Recovery-Prozess in der Regel abgeschlossen sein, so dass mit hoher Wahrscheinlichkeit keine Tupel verloren gehen.

`installQueriesOnNewPeer`

Schlägt eine `installQueriesOnNewPeer`-Nachricht fehl, kann das Recovery nicht ordnungsgemäß beendet werden. Es können zwei Fälle eintreten:

1. Das (wiederholte) Übermitteln der Nachricht ist fehlgeschlagen, da der Peer, der das Recovery durchführt, den ausgewählten Peer nicht (mehr) erreicht. Die Anfrage wurde nicht installiert.

2. Die Nachricht ist angekommen, jedoch ist die Antwort verloren gegangen. In diesem Fall wurde die Anfrage auf dem ausgewählten Peer installiert, der Peer, der das Recovery durchführt, bekommt es jedoch nicht mit.

Der erste Fall wird darüber gelöst, dass ein Neustart des Recoveries für den Query-Part, bei dem die Installation fehlgeschlagen ist, ausgeführt wird. Dafür wird die `restartRecovery`-Methode im `IRecoveryStrategyManager` genutzt. Die Behandlung des Neustarts ist im `Agreement-Handler` hinterlegt. Der zweite Fall bleibt unbehandelt, da dieser als unwahrscheinlich eingestuft und eine umfangreiche Fehlerbehandlung als zu viel Aufwand erachtet wird.

`sendRecoveryAgreementMessage`

Die `RecoveryAgreementMessage` wird für die Vereinbarung genutzt, welcher Peer das Recovery durchführen soll. Wenn diese Nachricht nicht versendet bzw. zugestellt werden konnte, gibt es mehrere mögliche Szenarien:

1. Die Nachricht konnte nicht beim ersten Mal zugestellt werden. Da die Nachricht so lange versendet wird, bis eine Antwort eintrifft, stellt dieser Fall keine Problematik dar.

2. Ein anderer Peer möchte ebenfalls das Recovery ausführen. Das führt dazu, dass `decideToRecover` wie im Normalfall aufgerufen wird.

3. Kein anderer Peer möchte wiederherstellen. Das führt dazu, dass dieser Peer dringend recovern sollte.

4. Mehrere `RecoveryAgreementMessage` Versuche (von anderen Peers) versagten. Das führt dazu, dass kein Peer weiß, ob andere wiederherstellen wollen.

    a) Nichts tun führt dann dazu, dass die Anfragen von dem ausgefallenen Peer verloren gehen.

    b) Das Recovery führt dann dazu, dass die Anfragen von dem ausgefallenen Peer mehrfach neu installiert werden.

Da das vierte Szenario weniger wahrscheinlich ist als die vorherigen, sollte hier keine weiterführende Fehlerbehandlung durchgeführt werden.

`sendUpdateReceiverMessage`

Die `sendUpdateReceiverMessage` wird dafür benötigt, dass während eines Recovery-Prozesses die Receiver der nachgelagerten Peers aktualisiert werden. Wenn ein Peer ausfällt und die Teilanfrage durch einen anderen Peer übernommen wird, müssen die nachfolgenden Peers aktualisiert werden, sodass der Datenstrom korrekt weiterverarbeitet wird. Den Receivern muss hierzu eine neue Pipe-Id mitgeteilt werden bzw. ein neuer Receiver mit der neuen PipeId installiert werden. Hierbei kann es möglicherweise zu einem Fehler kommen. Durch das wiederholte Senden dieser Nachricht wird gewährleistet, dass es zu mehreren Versuchen beim Erstellen des Receivers kommt. Sollte dies jedoch mehrfach fehlschlagen, gibt es folgende mögliche Szenarien:

1. Das Fehlschlagen bei der Installation des Receivers wird ignoriert. Allerdings führt dies dazu, dass die komplette Anfrage, sofern sie nicht repliziert wurde, keine Ergebnisse mehr liefert.

2. Es wird eine vorgegebene Zeit gewartet und anschließend erneut versucht, den Receiver mit der neuen PipeId zu installieren. Dieses Szenario würde allerdings dafür sorgen, dass während des Zeitraums alle Tupel verloren gehen und auch danach nicht gewährleistet ist, dass die Installation des Receivers bzw. die Nachrichtenzustellung möglich ist.

3. Der Recovery Prozess wird abgebrochen und auf einem anderen Peer durchgeführt. Hierzu ist es notwendig, dass auf dem zuvor ausgewählten Peer die Anfrage entfernt wird.

Da bei dem Szenario 1 keine weiteren Ergebnisse der Anfrage mehr möglich sind, wird diese Möglichkeit ausgeschlossen. Auch das Szenario 2 sorgt dafür, dass es zu einem Tupelverlust kommt, der im Kontext von Datenströmen vertretbar wäre. Allerdings ist nicht gewährleistet, dass der Peer in festgelegter Zeit erreicht werden kann. Sodass auch diese Möglichkeit nicht genutzt werden kann.

`sendUpdateSenderMessage`

Die `sendUpdateSenderMessage` wird verschickt, um den Sender des vorgelagerten Peers bei einem Recovery-Prozess zu aktualisieren. Dies ist notwendig, damit die Daten nicht mehr zu dem ausgefallenen Peer, sondern zu dem Peer geschickt werden, welcher diesen nach dem Recovery ersetzen soll. Zu einem Fehler kann es dabei kommen, wenn diese Nachricht den Sender nicht erreicht. Dies kann zum Beispiel passieren, wenn der vorgelagerte Peer ebenfalls ausgefallen ist, oder die beiden Peers keine Verbindung zueinander haben. Im Fehlerfall haben wir, äquivalent zum `sendUpdate-ReceiverMessage` Fehlerfall, folgende mögliche Szenarien:

1. Das Fehlschlagen bei der Installation des Senders wird ignoriert. Dies führt allerdings dazu, dass die komplette Anfrage, sofern es nicht repliziert wurde, keine Ergebnisse mehr liefert.

2. Es wird eine vorgegebene Zeit gewartet und anschließend erneut versucht den Peer zu erreichen, um den modifizierten Sender zu installieren. In diesem Zeitraum würden allerdings alle Tupel verloren gehen und es ist nicht gesichert, dass die Nachricht beim nächsten Versuch ankommt.

3. Der Recovery Prozess wird abgebrochen und auf einem anderen Peer erneut durchgeführt. Hierzu ist es notwendig, dass die Änderungen, welche im Zuge des fehlgeschlagenen Recovery-Prozesses vorgenommen wurden, zurückgesetzt werden (UNDO). Hierzu zählt auch das Update des Receivers des nachgelagerten Peers.

Szenario 1 würde zum Ausfall der Anfrage kommen, was nicht akzeptabel ist. Szenario 2 ist ebenfalls nicht zielführend, weil die Chance, dass die Nachricht beim zweiten Versuch ankommt, relativ gering ist. Dies wäre nur der Fall, wenn ein kurzfristiger Ausfall bei der Verbindung zwischen den beiden Peers die Fehlerursache ist. Szenario 3 würde den Fehlerfall am besten abfangen, allerdings ist die Umsetzung relativ komplex. Da der Fehlerfall als unwahrscheinlich eingestuft wird, wird eine Fehlerbehandlung im Rahmen dieser Projektgruppe nicht mehr umgesetzt. Des Weiteren ist eine Behandlung des Fehlerfalls momentan nicht notwendig, da die Implementierung vorsieht, dass der Receiver eine Verbindung zum Sender aufbaut und der Sender dementsprechend nicht den Receiver kennen muss. Wenn zwei aufeinander folgende Peers ausfallen entsteht ein Deadlock.

6.7.5 Simple Standardstrategie

Die simple Standard-Strategie ist diejenige, die verwendet wird, wenn keine andere Strategie angegeben wurde. Bemerkt ein Peer den Ausfall eines anderen, beginnt er damit, das, was auf dem ausgefallenen Peer lief, auf anderen Peers zu installieren. Währenddessen wird versucht, den Datenstrom zu puffern, um den Datenverlust gering zu halten.

Die simple Standardstrategie bietet einen umfangreichen und in den allermeisten Fällen funktionstüchtigen Ablauf für das Recovery. Es kann alleine genutzt, durch andere Strategien (z.B. Active-Standby) erweitert oder auch ganz durch eine andere Strategie ersetzt werden. Die Standardstrategie versucht den Datenverlust möglichst gering zu halten, kann diesen jedoch nicht vollständig verhindern.

Es ist möglich, dass jeder Peer das Recovery für einen ausgefallenen Peer durchführt. Dadurch ist es möglich, das Netzwerk auch bei vielen Ausfällen und nacheinander immer wieder auftretenden Ausfällen funktionstüchtig zu halten. Die Möglichkeiten haben jedoch auch Grenzen. So kann es sein, dass das Recovery fehlschlägt, wenn zwei Peers gleichzeitig ausfallen, die aufeinander folgende Query-Parts ausgeführt haben.

6.7.5.1 Ablauf

Im Folgenden wird der fehlerfreie Ablauf für die Recovery einer einzelnen Teilanfrage beschrieben. Um die Bezeichnungen kurz halten zu können zeigt Tabelle 6.2 verwendete Abkürzungen.

Die Empfangsbestätigungen für Nachrichten (siehe oben) werden hier der Übersicht halber nicht beschrieben. Abbildung 6.19 zeigt die Ausgangssituation für das Szenario.

Bezeichnung	Beschreibung
P1	Der Peer, der den Ausfall eines Peer bemerkt
Pf	Der ausgefallene Peer (Peer fail)
Pc	Der Peer, der übernimmt (Peer chosen)
QP	Die zu wiederherstellende Teilanfrage (QueryPart)

Tabelle 6.2: Abkürzungen für die einfache Recovery-Strategie

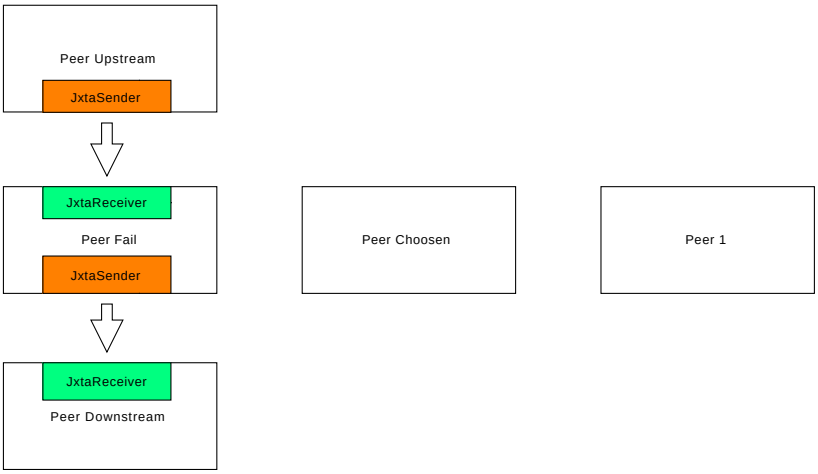


Abbildung 6.19: Ausgangssituation beim Recovery

Schritt 1

Schritt: Ausfall erkennen

Ausführender Peer: P1

Beschreibung: P1 bemerkt den Ausfall von Pf.

hauptsächlich beteiligte Klassen: `RecoveryCommunicator` (Die Methode `peerRemoved` wird aufgerufen.)

Gesendete Nachrichten: -

Schritt 2

Schritt: Aufruf der Strategie

Ausführender Peer: P1

Beschreibung: Die eingebundene Strategie wird aufgerufen, in diesem Fall die einfache Strategie.

hauptsächlich beteiligte Klassen: `RecoveryStrategyManagerSimple`

Gesendete Nachrichten: -

Schritt 3

Schritt: Informationslage prüfen

Ausführender Peer: P1

Beschreibung: P1 prüft, ob er Informationen über Pf hat (vgl. Kapitel 6.7.1) und fährt fort, falls dem so ist.

hauptsächlich beteiligte Klassen:
`SimpleRecoveryStrategy` und `BackupInformationAccess`

Gesendete Nachrichten: -

Schritt 4

Schritt: Allokation

Ausführender Peer: P1

Beschreibung: Ab jetzt werden die Schritte für alle bekannten Query-Parts einzeln durchgeführt. In unserem Beispiel nur einer (QP). Für diesen wird eine Liste aller Peers erstellt, die ihn übernehmen könnten. Diese Liste kann je nach Allokationsstrategie (vgl. Kapitel 6.7.1) variieren.

hauptsächlich beteiligte Klassen: `SimpleRecoveryStrategy, RecoveryDynamicBackup, AbstractRoundRobinAllocator`

Gesendete Nachrichten: -

Schritt 5

Schritt: Vorgänger Anhalten

Ausführender Peer: P1

Beschreibung: Damit möglichst wenig Datenstromelemente verloren gehen, wird von P1 ermittelt, welche Peers für QP direkte Vorgänger in der Verarbeitungskette waren. Hatte man also eine Verarbeitungsreihenfolge Peer A $\rightarrow$ Pf, ist Peer A der Vorgänger von Pf. Fällt nun Pf aus, wird Peer A mitgeteilt, dass die Datenstromelemente, die dieser Peer senden möchte, zwischengespeichert werden sollen. Die Vorgänger werden ermittelt, indem der PQL-Ausdruck von QP auf `JxtaReceiver`-Operatoren überprüft wird und die dort eingetragenen IDs genutzt werden. Da potenziell viele Peers den Ausfall erkennen können (durchaus auch viel später), akzeptieren vorhergehende Peers nur eine solche Nachricht, damit sie nicht angehalten werden, wenn die Recovery schon fertig ist.

hauptsächlich beteiligte Klassen: `RecoveryDynamicBackup (determineAndSendHoldOnMessages), TupleSendSender, TupleSendReceiver`

Gesendete Nachrichten: `RecoveryTupleSendMessage`

Der aktuelle Stand nach dem 5. Schritt ist in Abbildung 6.20 abgebildet. `Peer Fail` ist ausgefallen, `Peer 1` hat den Ausfall bemerkt.

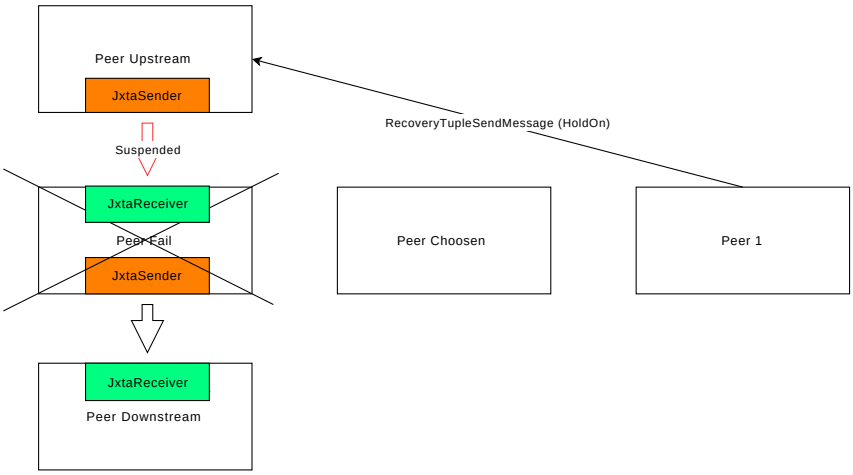


Abbildung 6.20: Situation nach Schritt 5 beim Recovery

Schritt 6.0

Schritt: Einigung (Agreement) über Recovery erreichen

Ausführender Peer: P1

Beschreibung: Da es möglich ist, dass mehrere Peers den Ausfall feststellen, muss eine Einigung darüber erzielt werden, wer das Recovery durchführt, damit der ausgefallene Query-Part nicht doppelt wiederhergestellt wird.

hauptsächlich beteiligte Klassen: `AgreementHelper (waitForAndDoRecovery)`

Gesendete Nachrichten: –

Schritt 6.1

Schritt: Auf die Erlaubnis zum Recovery warten

Ausführender Peer: P1

Beschreibung: Es wird geschaut, ob vielleicht ein anderer Peer schon (deutlich) früher festgestellt hat, dass Pf ausgefallen ist. In diesem Fall sind wir der Erste.

hauptsächlich beteiligte Klassen: `AgreementHelper (waitForAndDoRecovery)`

Gesendete Nachrichten: –

Schritt 6.2

Schritt: Einigungs-Rundschreiben

Ausführender Peer: P1

Beschreibung: Nun wird allen anderen bekannten Peers mitgeteilt, dass P1 das Recovery für QP auf Pf machen möchte. Dann wird eine festgelegte Zeit lang gewartet, ob noch ein anderer Peer zur gleichen Zeit so eine Nachricht versendet hat.

hauptsächlich beteiligte Klassen: `AgreementHelper`, `AgreementSender`, `AgreementReceiver`

Gesendete Nachrichten: `RecoveryAgreementMessage`, `RecoveryAgreementResponseMessage`

Abbildung 6.21 zeigt beispielhaft, wie `Peer 1` solche Nachrichten an alle ihm bekannten Peers versendet. Sollte ein anderer Peer in der Zeit, bis er diese Nachricht empfängt, ebenfalls eine solche Nachricht versenden, müssen sich die Peers in Schritt 6.3 einigen.

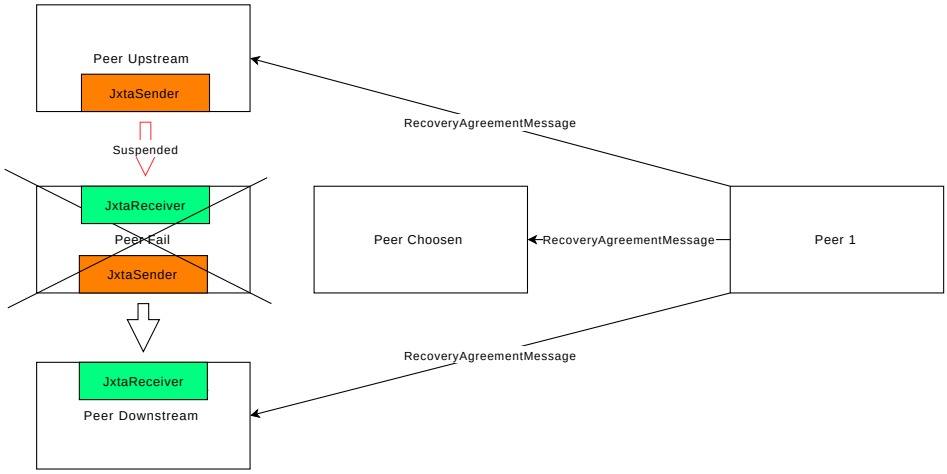


Abbildung 6.21: Situation bei Schritt 6.2 des Recoveries

Schritt 6.3

Schritt: Einigungs-Vergleich

Ausführender Peer: P1

Beschreibung: Angenommen, ein anderer Peer hat auch eine solche Nachricht versendet und P1 hat sie innerhalb der festgelegten Wartezeit erhalten. Dann vergleicht P1 die IDs aller Peers, die das Recovery für QP auf Pf durchführen möchten. Es wird sich auf den Peer mit der höchsten ID verständigt. Hier ist dies P1.

hauptsächlich beteiligte Klassen: `AgreementHelper (waitForAndDoRecovery)`

Gesendete Nachrichten: –

Schritt 7.0

Schritt: Installieren von QP auf anderem Peer

Ausführender Peer: P1, Pc

Beschreibung: Nun muss QP auf Pc installiert werden.

hauptsächlich beteiligte Klassen: Siehe Unterschritte

Gesendete Nachrichten: Siehe Unterschritte

Schritt 7.1

Schritt: Senden der Nachricht

Ausführender Peer: P1

Beschreibung: Die Nachricht wird an den Pc geschickt und beinhaltet u.a. den PQL-Ausdruck von QP.

hauptsächlich beteiligte Klassen: `RecoveryCommunicator, AddQuerySender`

Gesendete Nachrichten: `RecoveryAddQueryMessage`

Abbildung 6.22 veranschaulicht das Senden der Nachricht.

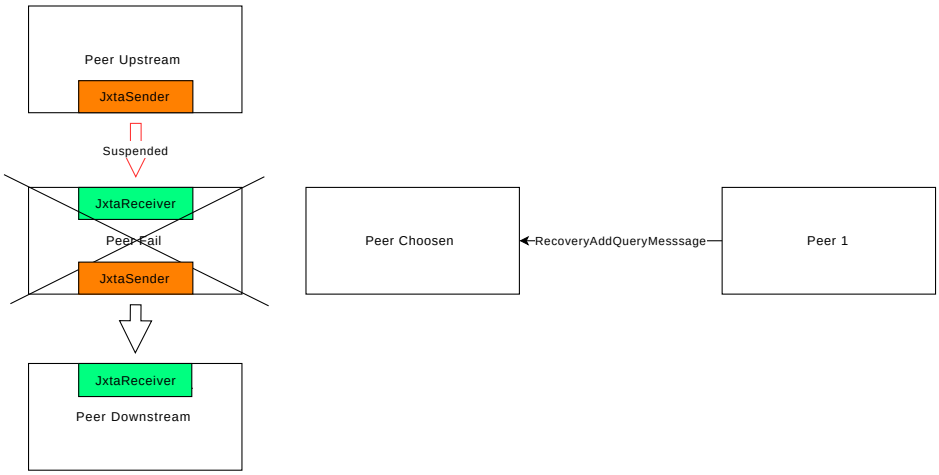


Abbildung 6.22: `Peer Choosen` bekommt Instruktionen

Schritt 7.2

Schritt: Installieren der Query

Ausführender Peer: Pc

Beschreibung: Empfängt Pc die Nachricht wird er versuchen, QP zu installieren. Dabei werden ein paar Besonderheiten berücksichtigt:

1. Enthält der PQL-Ausdruck die ID von Pf ist davon auszugehen, dass QP ein Query-Part ist, dessen Vorgänger oder Nachfolger ebenfalls auf Pf installiert war. Dann funktioniert die Installation nicht ohne weiteres, da die JXTA-Operatoren keine Verbindung aufbauen können. Hier wird auf einen Trick zurückgegriffen: Es wird die ID von Pc anstelle von Pf eingetragen.

2. Falls die Nachricht die ID einer verteilten Anfrage enthält, muss die wiederhergestellte Teilanfrage beim `QueryPartController` angemeldet werden.

3. Falls die Nachricht Verbindungsinformationen zum Handheld enthält, muss die Verbindung wieder aufgebaut werden (siehe Kapitel 6.7.7).

hauptsächlich beteiligte Klassen: `AddQueryReceiver`, `QueryPartController`, `SocketService`, `BackupInformationAccess`

Gesendete Nachrichten: –

Schritt 7.3

Schritt: Aktualisieren der Verbindungen

Ausführender Peer: Pc

Beschreibung: Die Verbindungen zu den anderen Peers (Vorgänger und Nachfolger) müssen aktualisiert werden. Es muss den anderen mitgeteilt werden, dass sie ihre JXTA-Sender-Operatoren beziehungsweise JXTA-Empfänger-Operatoren so umstellen müssen, dass sie nun zu Pc senden oder von Pc empfangen. Außerdem wird den Vorgängern mitgeteilt, dass sie nun weitersenden können (sie wurden in Schritt 5 angehalten). Das Senden dieser Nachrichten geschieht asynchron, um den Vorgang nicht unnötig zu verlangsamen. Es wird also fortgefahren, bevor die Antworten der Vorgänger und Nachfolger angekommen sind.

hauptsächlich beteiligte Klassen: `AddQueryReceiver, RecoveryCommunicator`

Gesendete Nachrichten: `RecoveryUpdatePipeMessage, RecoveryTupleSendMessage`

Die Situation nach Schritt 7.3 ist in Abbildung 6.23 dargestellt. Die Anfrage von `Peer Fail` wurde installiert, der `Peer Choosen` passt die Verbindungen der anderen Peer an. Anschließend wir dem `Peer Upstream` mitgeteilt, dass er mit dem Senden fortfahren kann.

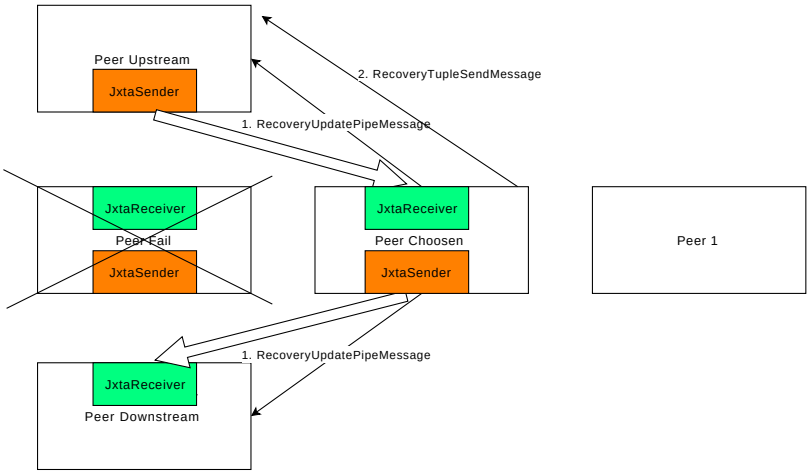


Abbildung 6.23: Situation nach Schritt 7.3

Schritt 8

Schritt: Backup-Informationen entfernen

Ausführender Peer: P1

Beschreibung: Sind die Schritte erfolgreich durchgelaufen, kann die Information über QP auf Pf aus den Backup-Informationen entfernt werden. Dies wird gemacht, wenn Pc meldet, dass die Anfrage erfolgreich installiert werden konnte. Das Löschen der Information wird auf P1 übernommen, da nicht davon ausgegangen werden kann, dass Pc die gleichen Informationen wie P1 hat.

hauptsächlich beteiligte Klassen: `AddQueryResponseHandler, BackupInformationAccess`

Gesendete Nachrichten: `RecoveryAddQueryResponseMessage`

Abschließend zeigt 6.24 die Situation nach einem erfolgreichen Recovery. Im Hintergrund wurden die Backup-Informationen auf allen Peers angepasst.

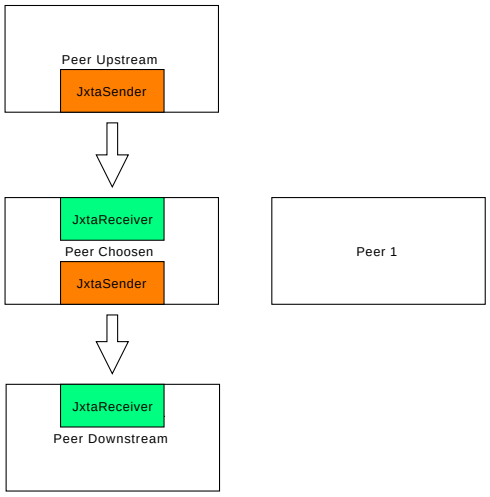


Abbildung 6.24: Situation nach erfolgreichem Recovery

6.7.6 Active-Standby

Abbildung 6.25 gibt einen Überblick über die Active-Standby-Implementierung. In den folgenden Abschnitten werden die einzelnen Klassen und Schnittstellen in die Kategorien „Operatoren“, „Modifikator“„Strategie“ und „Nachrichten“ unterteilt und erklärt.

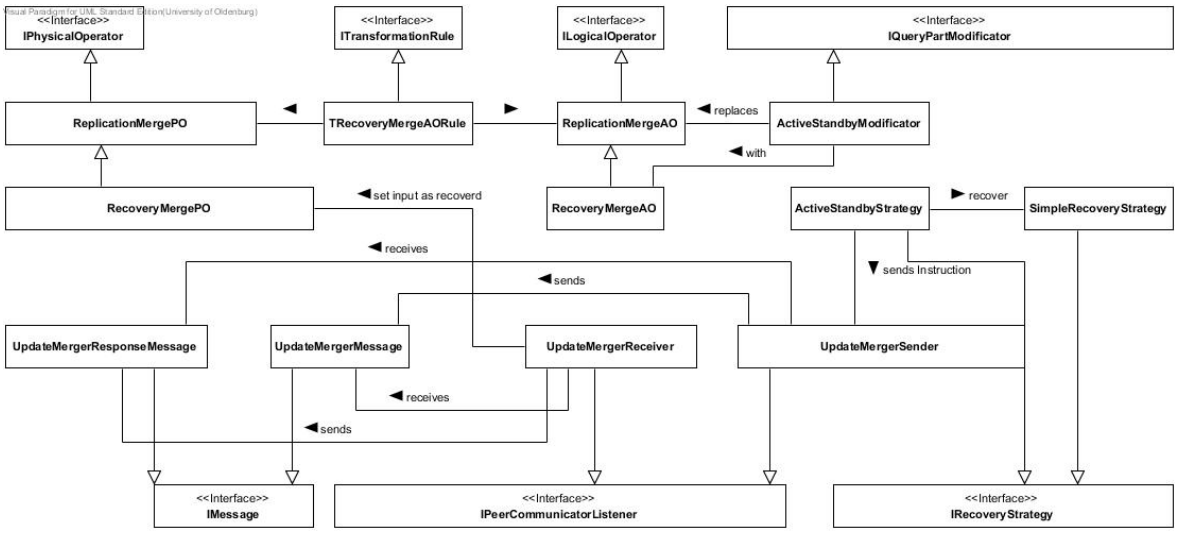


Abbildung 6.25: Klassenhierarchie für das Active-Standby

6.7.6.1 Operatoren

Für das Active-Standby wird der Operator `RecoveryMergeAO` bzw. `RecoveryMergePO` benötigt, welcher von `ReplicationAO` bzw. `ReplicationPO` abgeleitet wird. So lange noch kein Recovery statt gefunden hat, arbeitet der Operator zunächst wie ein normaler Replikationsoperator (`ReplicationPO`). Das heißt, der Operator vereinigt die replizierten Eingangsströme zu einem einzelnen Ausgangsstrom, so dass von den replizierten Tupeln immer nur einer in den Ausgangsstrom gelangt. Hierbei ist es möglich, dass Tupel von unterschiedlichen Eingangsströmen in den Ausgangsstrom gelangen. Findet nun ein Recovery statt und einer der Eingangsströme wird wiederhergestellt, sodass der Eingangsstrom von einem neuen Peer erzeugt wird, leitet der `RecoveryMerge`-Operator nur noch die Tupel von einem einzigen Eingangsstrom weiter, der noch nicht wiederhergestellt wurde. Wenn bereits alle Eingangsströme wiederhergestellt wurden, wird der Eingangsstrom ausgewählt, bei dem das Recovery am längsten her ist. Dies hat den Grund, dass der ausgewählte Eingangsstrom die geringsten Verluste bezüglich der Operatorenzustände hat und dementsprechend die besten Ergebnisse liefert. Bei wiederhergestellten Eingangsströmen werden die Operatoren neu erzeugt, so dass auch die Operatorenzustände neu sind und die Ergebnisse i.d.R. nicht die gleichen sind wie von einem nicht-wiederhergestellten Eingangsstrom.

6.7.6.2 Modifikator

Der Modifikator `ActiveStandbyModificator` erbt von `IQueryPartModificator` und hat den Zweck, aus einer logischen Anfrage die `ReplicationMergeAO`-Operatoren mit `RecoveryMergeAO`-Operatoren zu ersetzen. Dies ist notwendig, da Odysseus bei Replikation standardmäßig `ReplicationMergeAO` als Merge-Operator verwendet.

6.7.6.3 Strategie

Die Klasse `ActiveStandbyStrategy` erbt von `IRecoveryStrategy` und wird verwendet, sobald ein Peer nicht mehr erreichbar ist und das Recovery durchgeführt werden soll. Bei Active-Standby wird hierbei als erstes den `RecoveryMerge`-Operatoren mitgeteilt, welche Eingangsströme wiederhergestellt werden müssen. Dazu wird die Klasse `UpdateMergeSender` benutzt. Für die weiteren Schritte des Recovery-Prozesses wird die Klasse `SimpleRecoveryStrategy` verwendet, da sich das Active-Standby im weiteren Verlauf nicht von der einfachen Recovery-Strategie unterscheidet.

6.7.6.4 Nachrichten

Die Klasse `UpdateMergerSender` hat den Zweck, die Nachricht `UpdateMergerMessage` an diejenigen Peers zu senden, welche einen `RecoveryMerge`-Operator ausführen, bei dem ein Eingangsstrom wiederhergestellt werden soll. Damit wird den `RecoveryMerge`-Operatoren mitgeteilt, dass der wiederhergestellte Eingangsstrom zunächst nicht mehr verwendet werden soll. Die Klasse `UpdateMergerReceiver` empfängt die Nachricht `UpdateMergerMessage` und führt die Änderungen bei den entsprechenden `RecoveryMerge`-Operatoren durch. Die Nachricht `UpdateMergeResponseMessage` wird dabei sofort als Antwort gesendet und teilt dem Sender `UpdateMergerSender` mit, dass die Nachricht angekommen ist.

6.7.7 Ausfall eines Socket-Sender-Peers

Generell ist es möglich, dass jeder Peer des Netzwerks ausfällt und durch das Recovery auf einen anderen Peer übertragen werden kann. Handelt es sich dabei allerdings um den Peer, der die Socket-Verbindung zum Tablet aufgebaut hat, muss die Verbindung nach dem Ausfall wieder aufgebaut werden, sodass das Tablet weiter Daten empfängt. Dieser Spezialfall wird im folgenden Abschnitt genauer beschrieben.

6.7.7.1 Speicherung der nötigen Informationen

Zum Wiederaufbau der Verbindung sind folgende Informationen nötig:

1. IP des Tablets (damit diesem mitgeteilt werden kann, wo es nun für welche ausgefallene Verbindung die Daten wieder herbekommt)

2. IP des Host-Peers (dies ist die IP des Peers, auf dem die Socket-Verbindung gerade läuft. Fällt dieser aus, wird diese Information benötigt, damit das Tablet weiß, welche Verbindung ersetzt werden muss)

3. Port des Sockets beim Host-Peer (Der Port der Verbindung, wird aus den selben Gründen wie die IP gespeichert)

Die Speicherung findet an zwei Stellen statt, je nachdem, ob die Anfrage verteilt wird oder nicht. Wird er nicht verteilt, werden die Informationen in der Klasse `ExecuteSportsQLServerResource` gespeichert, da dann der Socket auf dem Peer geöffnet wird, auf dem die Nachricht zum

Installieren ankommt. Wird die Anfrage hingegen verteilt, geschieht dies in der Klasse `DistributedSportsQLSocketServerResource`. Dazu wird die IP des Tablets, die nur der erste Peer kennt, mit der Nachricht zum Installieren der Anfrage mitgeschickt.

Das Speichern geschieht über die Klasse `BackupInformationAccess`. Da keine direkte Abhängigkeit zum Recovery bestehen soll, ist im `SocketService` das Observer-Pattern eingebaut worden. Dies informiert den `BackupInformationHelper` über Änderungen, der das dann in den Backup-Informationen einträgt. Außerdem werden die Informationen über die neue Socket-Verbindung gespeichert, wenn im `AddQueryReceiver` eine Anfrage installiert wird, bei dem auch ein ausgefallener Socket ersetzt werden muss. Dann werden die Informationen des neuen Sockets gespeichert.

6.7.7.2 Vorgehen beim Ausfall

Fällt ein Peer aus, wird regulär nach den Strategien vorgegangen. Hat eine `RecoveryAddQueryMessage` Informationen zu einer Socket-Verbindung, weiß der Peer, der die Anfrage installieren soll, dass hier eine neue Socket-Verbindung aufgebaut werden muss. Der Socket-Server wird ganz normal, wie auch sonst, über den `SocketService` erstellt. Dann werden die Informationen gespeichert und anschließend der Client (für gewöhnlich das Tablet) informiert. Dieser Ablauf geschieht im `AddQueryReceiver`.

Informieren des Clients

Der Client wird über eine Nachricht der folgenden Form informiert:

$$IP_{AusgefallenerPeer} : Port_{AusgefallenerSocket} | IP_{NeuerPeer} : Port_{NeuerSocket},$$

also z.B. 192.168.1.1:12345|192.168.1.42:54321 . Dies geschieht derzeit über Port 53000. Das bedeutet, dass auf dem Client ein ServerSocket auf diesem Port lauscht und im Falle einer solchen Nachricht reagieren muss. In der Coach App geschieht dies im `PeerInfoServerSocket`.

Kommt eine solche Nachricht an, wird geschaut, ob Informationen (in diesem Fall Attribut-Informationen) über den ausgefallenen Socket vorhanden sind. Wenn ja, hatte der Client eine Verbindung und wird diese neu aufbauen. Die alte Verbindung wird gelöscht und eine neue mit den selben Listenern (Anfragen) wieder aufgebaut. So erhalten die Listener, die vorher von dem alten Socket Informationen bekommen haben, nun von dem neuen Socket die Informationen.

Auf dem Peer wird diese Nachricht im `RecoveryCommunicator` in der Methode `informClientAboutNewSocket` abgeschickt. Dazu wird die Klasse `SimpleSocketClient` benutzt.

6.8 LSOP Service

Dieser Abschnitt geht auf die Umsetzung des LSOP Services in Herakles ein. Der LSOP Service ist eine Bibliothek, die eine Reihe an Schnittstellen für die Sportanalyse mit Odysseus anbietet und die von unterschiedlichen GUI-Anwendungen genutzt werden können. Dadurch kann der Aufwand reduziert werden, der für die Implementierung einer neuen GUI-Anwendungen (z.B. einer Traineranwendung für den Desktop) nötig wäre. In Herakles wird der LSOP Service derzeit von der Coach und Developer Application genutzt.

6.8.1 Schnittstelle für den Distributed Data Container

Die DDC-Schnittstelle bietet eine Reihe an Methoden an, die Metadaten zur Sportanalyse liefern. Dazu gehören u.a. die Abmessungen des Spielfelds oder die Informationen zu den Teams und Spielern. Die Abbildung 6.26 gibt einen Überblick über die Implementierung der Schnittstelle.

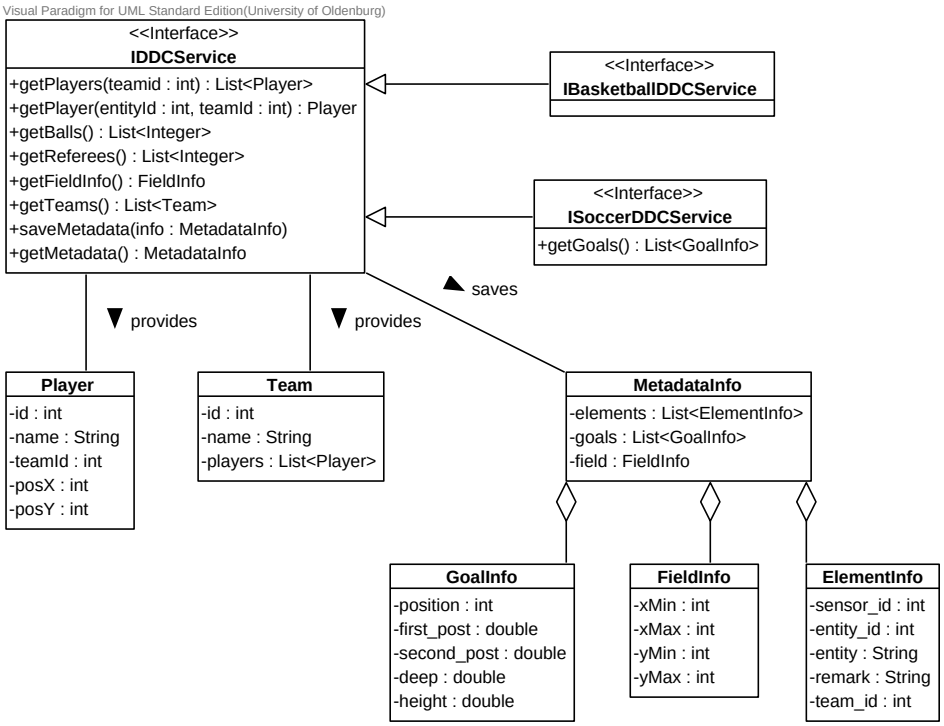


Abbildung 6.26: Verwendung des DDCs im LSOP Service

Damit Metadaten über die Schnittstelle angeboten werden können, müssen sie zunächst über die REST-Schnittstelle von Odysseus angefordert werden. Anschließend können die Metadaten über die `saveMetadata`-Methode von `IDDCService` gespeichert werden. Die DDC-Schnittelle verarbeitet nun die Metadaten und bietet einzelne Daten über verschiedene sportartenunabhängige Methoden an (z.B. `getFieldInfo`). Für sportartenabhängige Methoden muss eine neue Schnittstelle geschrieben werden (z.B. `ISoccerDDCService`), die von `IDDCService` erben muss.

6.8.2 Schnittstelle für Anfrageergebnisse

Mithilfe dieser Schnittstelle ist es möglich, die Ergebnisse einer Anfrage von Odysseus anzufordern. Dazu muss die Schnittstelle zum einen die IP-Adresse und den Port wissen, mit denen eine Socket-Verbindung zu einem Odysseus-System aufgebaut werden kann. Über die Socket-Verbindung werden die Ergebnisse einer Anfrage in einem Byte-Strom gesendet. Zum anderen benötigt die Schnittstelle das Ausgabeschema der Anfrage, um den Bytestrom in die einzelnen Datentypen der Attribute des Ausgabeschemas zu transformieren. Die Abbildung 6.27 zeigt die Struktur der Schnittstelle.

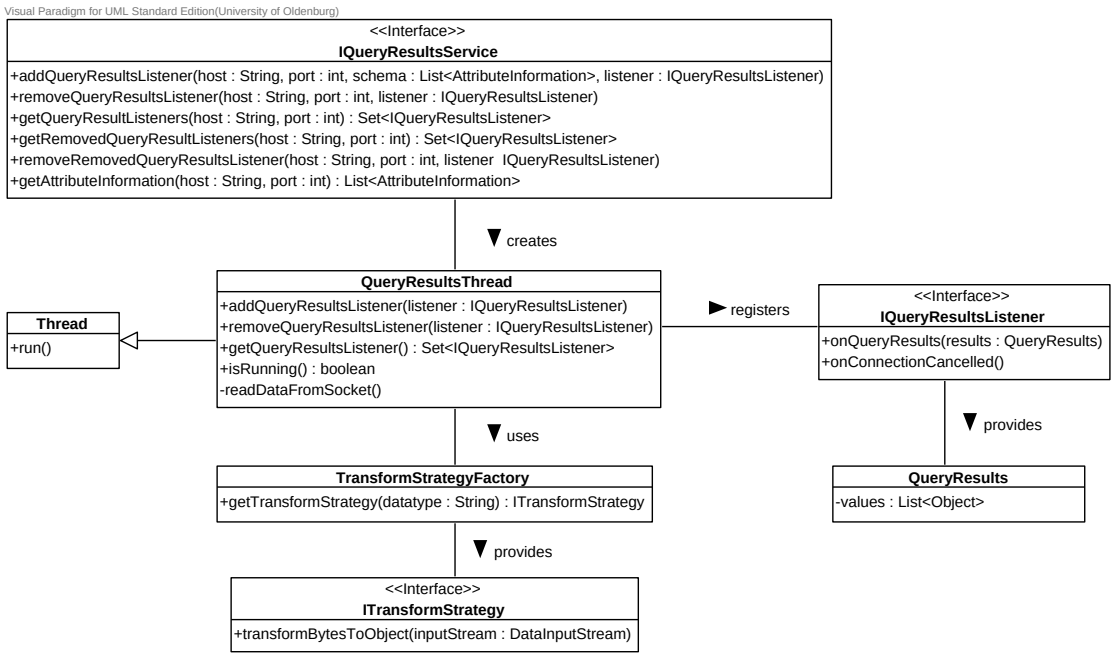


Abbildung 6.27: Anfordern der Ergebnisse einer Anfrage im LSOP Service

Über `IQueryResultsService` kann eine Anwendung einen `IQueryResultsListener` für die Ergebnisse einer Anfrage hinzufügen oder entfernen. In `QueryResultsThread` wird die Socket-Verbindung zu einem Odysseus-System aufgebaut. Die Ergebnisse werden über die Methode `readDataFromSocket` gelesen. Mit Hilfe der Factory `TransformStrategyFactory` können verschiedene Transformationsstrategien angefordert werden, mit denen Bytes in Datentypen umgewandelt werden. Aktuell gibt es Transformationsstrategien für die Standarddatentypen (Integer, String etc.) und für Timestamps. Der `IQueryResultsListener` erhält letztlich die Ergebnisse einer Anfrage als Liste von Objekten über die Methode `onQueryResults`. Jedes Objekt stellt dabei den Wert eines Attributes des Ausgabeschemas dar.

6.8.3 Schnittstelle für SportsQL

Diese Schnittstelle ermöglicht das Generieren von SportsQL-Anfragen. Diese können anschließend an Odysseus gesendet und ausgeführt werden. Bei der Konzeption der Schnittstelle wurde besonders auf die Austauschbarkeit und Erweiterbarkeit geachtet. Darüber hinaus wurde ein modularer Ansatz verfolgt, der es erlaubt, die Anfragen beliebig zu gestalten. Die Erzeugung der Anfragen ist durch diesen Ansatz maximal an die jeweiligen Bedürfnisse angepasst. Die Abbildung 6.28 zeigt die derzeitige Klassenstruktur beispielhaft für die Sprints-Anfrage.

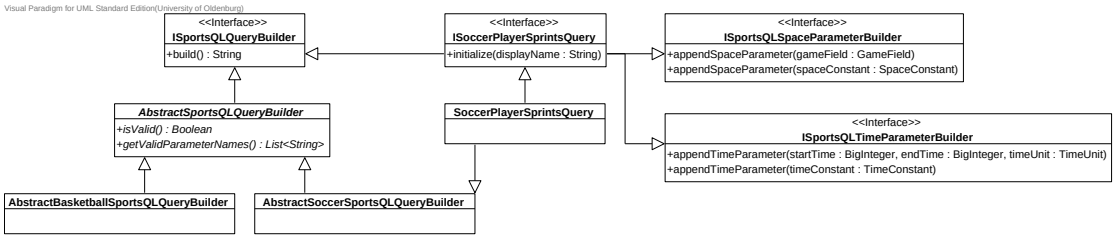


Abbildung 6.28: Erstellen von SportsQL-Anfragen im LSOP Service

Zu jeder Anfrage-Implementierung gibt es eine entsprechende Schnittstelle, wie etwa `ISoccerSprintsPlayerQuery`, die erstellt werden muss und in der individuelle Initialisierungsmethoden hinterlegt sind. Diese Schnittstelle muss zum einen die Basis-Schnittstelle `ISportsQLBuilder` und zum anderen die jeweiligen Schnittstellen für die Parameter der Anfrage erweitern. Da die Sprints-Anfrage zwei Parameter hat, wird die Schnittstelle dementsprechend um `ISportsQLSpaceParameterBuilder` und `ISportsQLTimeParameterBuilder` erweitert. Darüber hinaus gibt es noch zahlreiche weitere Schnittstelle für Parameter (Integer, String, etc.), die die Sprints-Anfrage jedoch nicht verwendet. Die konkrete Implementierung `SoccerPlayerSprintsQuery` muss die Schnittstelle `ISoccerSprintsPlayerQuery` implementieren und zudem von `AbstractSocerSportsQLQueryBuilder` erben. Um nun eine Sprints-Anfrage generieren zu lassen, können zunächst über die Methoden `appendSpaceParameter` und `appendTimeParameter` von `ISoccerSprintsPlayerQuery` zwei Parameter festgelegt werden. Die `build`-Methode generiert letztlich die SportsQL-Anfrage und liefert diese als String zurück.

6.9 Coach Application

Dieser Abschnitt befasst sich mit der Implementierung der Coach App. Die App wurde für Android Geräte entwickelt und stellt die Ergebnisse der Anfragen übersichtlich dar. Die Anfragen wurden mittels LSOP Service an Odysseus gestellt. Die folgenden Teilabschnitte sollen die Implementierung der App, mit dem Schwerpunkt auf die Architektur der GUI und der Anfragen, vorstellen.

6.9.1 Architektur - GUI

Das Diagramm in Abbildung 6.29 beschreibt die grundlegende Architektur der App, mit der `ChooseGameActivity` als Einstiegspunkt. Wie bereits in Abschnitt 2.12 beschrieben, besitzt jede Activity mindestens ein Fragment. Im Fall der `ChooseGameActivity` wird das `ChooseGameFragment` realisiert. Von hier aus kann der Nutzer entweder die Einstellungen aufrufen oder die Analyse starten. Die Einstellungen werden mithilfe der `SettingsActivity` sowie den zugehörigen Fragments `GeneralPreferenceFragment`, `PeerPreferenceFragment` sowie `QueryPreferenceFragment` umgesetzt.

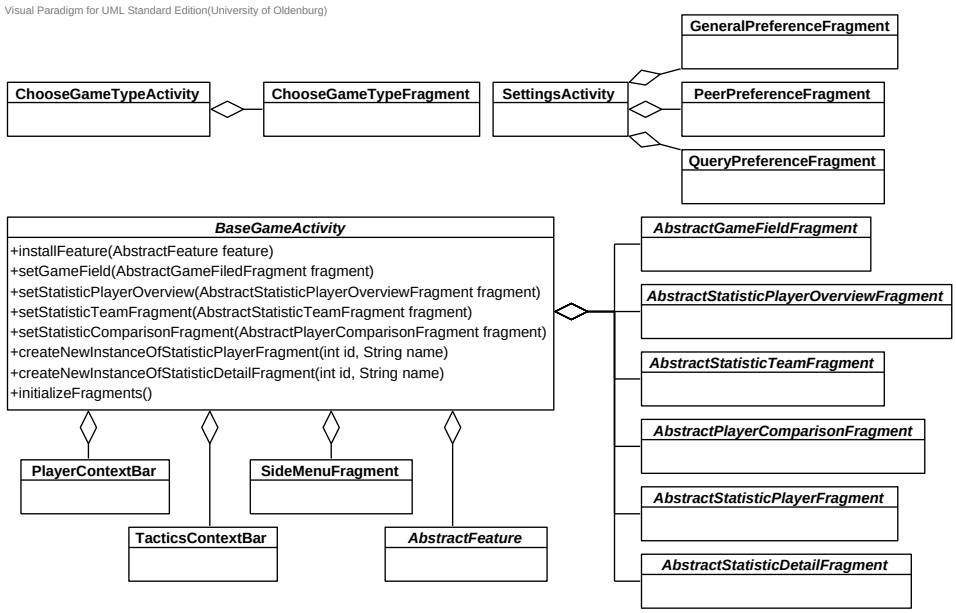


Abbildung 6.29: Grundarchitektur Coach Application

Die Analyseansicht besteht aus mehreren konkreten Implementierungen der abstrakten Fragments und der abstrakten Activity `BaseGameActivity`. Der Übersicht halber soll an dieser Stelle nur die Grundlage erläutert werden.

Basis ist die `BaseGameActivity`. Hier werden grundlegende Layouts und Funktionen festgelegt, die in allen Sportarten umgesetzt werden sollen. Hierzu gehören die abstrakten Fragments, wie z.B. das Spielfeld (`AbstractGameFieldFragment`) und die Übersicht der Spieler (`AbstractStatisticPlayerOverview`). Mithilfe der entsprechenden Methoden aus der `BaseGameActivity` können diese in die Ansicht eingebunden werden. Sollte z.B. für das Spielfeld kein Fragment angegeben werden, so wird das Layout der Analyse so gestaltet, dass der freie Platz durch die anderen Fragments besetzt wird. Nach dem Hinzufügen von Fragments, muss die Methode `initializeFragments()` aufgerufen werden, um entsprechende vordefinierte Bereiche mit den Fragments zu füllen. Die Basisfunktionen umfassen das Starten, Pausieren, Stoppen und Löschen von Anfragen. Des Weiteren sind auch spezielle Implementierungen der „Contextual Action Bar (CAB)“ bereits im Basispaket vorhanden. Hierbei handelt es sich um die `PlayerContextBar` und `TacticsContextBar`.

Um weitere, sportartspezifischere Funktionen zu ergänzen, wurde die Klasse `AbstractFeature` bereitgestellt. Die `BaseGameActivity` beinhaltet eine Liste, in die Features mittels der Methode `addFeature(AbstractFeature feature)` ergänzt werden können. Dem Entwickler bleibt es selbst überlassen, welchen Umfang die einzelnen Features haben.

Insgesamt wurde die Architektur sehr allgemein gehalten, sodass Entwickler eigene Fragments und Features für unterschiedliche Sportarten entwickeln können.

6.9.2 Architektur - Anfragen

Das Erstellen von Anfragen innerhalb der Coach Application ist die Voraussetzung für das Anzeigen von Statistiken. Das Klassendiagramm aus Abbildung 6.30 zeigt die wesentlichen Strukturen, die beim Erstellen von Anfragen in der Coach Application verwendet werden. Die Schnittstelle `IQueryExecutor` wird verwendet, um alle benötigten Anfragen zu erstellen, starten, pausieren, stoppen und entfernen. Es gibt verschiedene Implementierungen der Schnittstelle, die sich z.B. darin unterscheiden, dass Anfragen direkt beim Erstellen oder manuell gestartet werden. Für die Kommunikation mit Odysseus benutzt die Schnittstelle die Klasse `QueryCommunicator`, die wiederum auf die Schnittstelle `IRestService` zugreift. `IRestService` führt die REST-Anfragen an ein Odysseus-System letztlich aus und verwendet dazu die Bibliothek RoboGuice[8], die zahlreiche Hilfsklassen für die REST-Kommunikation anbietet. Um das Erstellen einer Anfrage innerhalb der Coach Application zu ermöglichen, muss die Schnittstelle `IQuery` implementiert werden. Über die Schnittstelle wird bspw. das SportsQL einer Anfrage angefordert. Die Implementierung muss in der `QueryFactory` bekannt gemacht werden. Diese Factory legt letztlich fest, welche Anfragen mit der Coach Application ausgeführt werden können. Zu jeder Implementierung von `IQuery` gibt es eine dazugehörige Implementierung von `IQueryListener`. Mit Hilfe des Listeners wird die GUI über neue Ergebnisse einer Anfrage informiert. Die Schnittstelle `IQuery` erhält die Ergebnisse einer Anfrage mit Hilfe eines `IQueryResultsListener`, der in der Schnittstelle `IQueryResultsService` vom LSOP Service registriert werden muss. Die Ergebnisse werden hierbei jedoch nur als Array von `java.lang.Object` geliefert, so dass `IQuery` als Schnittstelle zwischen `IQueryResultsService` und GUI diese noch zu konkreten Datentypen transformieren muss.

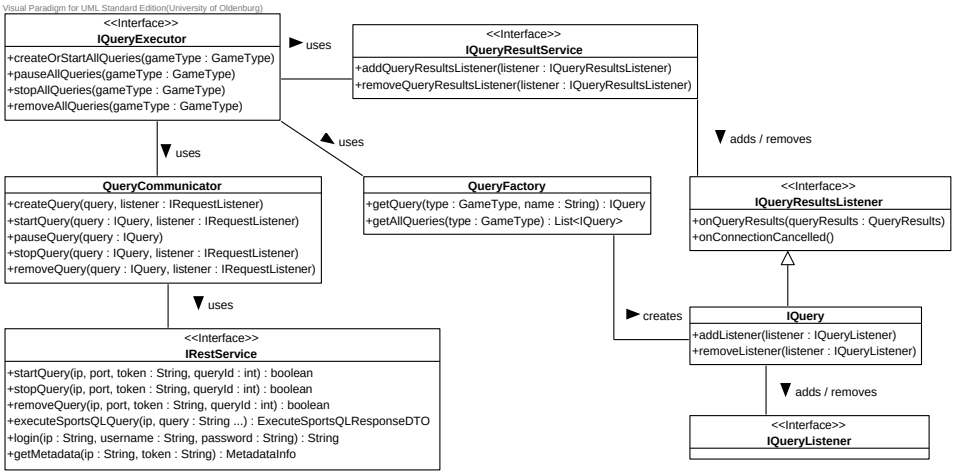


Abbildung 6.30: Anfragen in der Coach Application

[8] `https://github.com/roboguice/roboguice`

6.10 Developer Application

Die Developer Application ist eine GUI-Anwendung, die mit dem Framework JavaFX[9] implementiert wurde. Die Anwendung dient hauptsächlich zum Testen des LSOP Services und der REST-Schnittstelle von Odysseus, da dies mittels der Coach App auf einem Android-Gerät sehr aufwendig ist. So lassen sich z.B. SportsQL-Anfragen ausführen und die Ergebnisse der Anfrage können betrachtet werden. Die Entwicklung der Anwendung wurde nach den ersten drei Monaten eingestellt, da nur noch wenig Änderungen an dem LSOP Service und der REST-Schnittstelle vollzogen wurden. Außerdem konnten zu dem Zeitpunkt die SportsQL-Anfragen auch über Odysseus-Studio ausgeführt werden und nicht, wie ursprünglich gedacht, nur über die REST-Schnittstelle.

6.11 Monitoring Application

In diesem Abschnitt wird die Umsetzung der Monitoring Application beschrieben. Für die Monitoring Application wurde die bereits verfügbare Webanwendung von Odysseus um einige Funktionen zur Steuerung und Überwachung des P2P-Netwerkes erweitert. Die Webanwendung wird mit dem Google Web Toolkit[10] entwickelt und mit ihr ist es möglich, die wesentlichen Funktionen von Odysseus auszuführen, indem mit der SOAP-Webservice-Schnittstelle von Odysseus kommuniziert wird. Dazu gehört z.B. das Erstellen, Starten, Stoppen und Entfernen von Anfragen. Die Bedingung hierfür ist jedoch, dass der Nutzer die IP und den Port der Webservice-Schnittstelle des Odysseus-Systems kennt. Ein wesentliches Ziel der Monitoring Application ist es, dass die Odysseus-Peers automatisch entdeckt werden und mit ihnen kommuniziert werden kann. Um dies umzusetzen, wurde der Server der Webanwendung um die Funktionalitäten von JXTA erweitert. Der Server ist dadurch selber ein Peer im Netzwerk und kann die anderen Odysseus-Peers identifizieren. Dadurch lassen sich die gewünschten Funktionen aus dem Konzept zur Monitoring Application mit geringem Aufwand umsetzen.

Informationen zu Odysseus-Peers

Um Informationen zu den einzelnen Odysseus-Peers zu erhalten, sendet der Webserver in regelmäßigen Abständen eine `AskUsageMessage` an alle Peers. Die Peers antworten mit einer `AnswerUsageMessage`, welche Informationen u.a. zu Speicher- und CPU-Auslastung beinhaltet. Die Informationen werden anschließend in Form einer Tabelle in der Webanwendung dargestellt.

Aktualisieren und Neustarten von Odysseus-Peers

Odysseus-Peers können über einen Button in der Webanwendung aktualisiert und neu gestartet werden. Dazu sendet der Server eine `DoUpdateMessage` oder `DoRestartMessage` an ein Odysseus-System.

[9] http://www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html
[10] http://www.gwtproject.org/

Senden von Konsolen-Befehlen an die Odysseus-Peers

Über eine Konsole in der Webanwendung können Befehle an Odysseus-Peers gesendet werden. Dazu wird beim Server eine `CommandMessage` an ein Odysseus-System gesendet. Das Odysseus-System antwortet darauf i.d.R. mit einer `CommandOutputMessage`. Die Antwort wird anschließend in der Konsole angezeigt. Zuvor muss sich der Server jedoch mit einer `LoginMessage` bei dem Odysseus-System anmelden.

Darstellung des Logs eines Odysseus-Peers

Um die Logs der Odysseus-Peers zu empfangen, muss der Server zunächst ein `LoggingAdvertisement` veröffentlichen. Anschließend kann der Server Log-Nachrichten in Form einer `LogMessage` empfangen. Die Log-Nachrichten beinhalten ein Log-Level, einen Logger-Namen und ein Log-Text. Die Logs werden in der Webanwendung in Form einer Liste dargestellt, wobei nach verschiedenen Einstellungen gefiltert werden kann (z.B. nach dem Level).

Darstellung einer Ping-Map

Die Ping-Map eines Odysseus-Peers wird in der Webanwendung als SVG-Grafik dargestellt. Die Daten zu einer Ping-Map sendet ein Odysseus-System über eine Socket-Verbindung an den Webserver. Zuvor muss der Webserver das Odysseus-System über die REST-Schnittstelle auffordern, ein Socket für die Ping-Map zu öffnen. Die Daten der Ping-Map erhält Odysseus mit Hilfe eines `IPingMapListener` über die Schnittstelle `IPingMap`.

Darstellung eines verteilten Anfrageplans

Die Darstellung eines verteilten Anfrageplans wird in der Webanwendung ebenfalls mit SVG realisiert. Um einen verteilten Anfrageplan zu erstellen, fordert der Webserver zunächst über die REST-Schnittstelle alle Shared-Query-Ids eines Odysseus-Systems an. Dieses Odysseus-System wird im Folgenden als Master bezeichnet. Odysseus kann die Shared-Query-Ids über die Schnittstelle `IQueryPartController` erhalten. Nach Auswahl einer Shared-Query-Id durch den Nutzer benötigt der Webserver die lokalen Anfragen, die zu der verteilten Anfrage gehören. Dazu kommuniziert der Webserver wiederum über die REST-Schnittstelle mit dem Master. Der Master kann über `IQueryPartController` herausfinden, welche Peers an der verteilten Anfrage beteiligt sind. Von allen beteiligten Peers fordert der Master nun über die REST-Schnittstelle die lokalen Anfragen an, die der jeweilige Peer zu der verteilten Anfragen besitzt. Die lokalen Anfragen zu einer Shared-Query-ID kann der beteiligte Peer wiederum über `IQueryPartController` erhalten. Nachdem der Master die Antworten von allen Peers erhalten hat, kann er dem Webserver antworten. Der Webserver fügt nun die einzelnen lokalen Anfragen zu einem Shared-Query-Graphen zusammen, in dem jeder `SenderAO`-Operator mit dem passenden `ReceiverAO`-Operator verknüpft wird. Der passende `ReceiverAO`-Operator zu einem `SenderAO`-Operator liegt dann vor, wenn beide über die selbe `PipeID` verfügen.

6.12 Zusammenfassung

In diesem Kapitel wurde die Implementierung der verschiedenen Komponenten von Herakles vorgestellt. Zunächst wurde gezeigt, wie die GPS- und WLAN-Module des Smartphones als Positionssensor für die Tracking Application verwendet werden können. Im Anschluss wurde der GPServer besprochen, der die gesendeten Daten der Smartphones bündelt, speichert und zu Odysseus schickt.

Der folgende Abschnitt behandelt die Datenabstraktion. Hier wurde beschrieben, wie die Daten in eine einheitliche Form gebracht werden, um somit die Sensorunabhängigkeit zu gewährleisten. Daran beteiligt ist auch das DDC, welches im hier ebenfalls beschrieben wurde. Es wurde erläutert, wozu es genutzt wird und wie es verwendet wird, um die Konstanten auf allen Peers verfügbar zu machen.

Im Abschnitt zur Datenanalyse werden beispielhaft zwei Anfragen, die Analysezeit und die Sprints, genauer vorgestellt. Eng mit den Anfragen verwand ist die selbst entwickelte domänenspezifische Anfragesprache SportsQL. Die Umsetzung dieser wurde ebenfalls erklärt.

Der darauffolgende Abschnitt befasst sich mit der Kommunikation der verschiedenen Komponenten in Herakles untereinander. Dazu werden JXTA zur Verständigung der Peers untereinander und REST zur Kommunikation der verschiedenen GUI-Anwendungen mit Odysseus verwendet. Weiterhin wurde die Implementierung des Load-Balancings behandelt. Im Einzelnen wird auf die Realisierung der `SimpleLoadBalancing`-Strategie zur Überwachung eines Peers, des `RoundRobin`-Allokators zur Ermittlung eines neuen geeigneten Peers und der Parallel-Track-, Moving-State-, Inactive-Query-Kommunikatoren zur Durchführung des eigentlichen Load-Balancing Vorgangs eingegangen.

Im Zuge der Beschreibung des Recoverys wurden zunächst die Backup-Informationen, die Details über die Peers enthalten, erklärt. Sie sind notwendig, um im Falle eines Ausfalls die nötigen Informationen zur Wiederherstellung bereit zu haben. Im Anschluss wurden auch hier die Allokation und die Kommunikation inklusive Fehlerbehandlung vorgestellt. Darauffolgend wurde genauer auf die Simple Standardstrategie eingegangen und der Ablauf Schritt für Schritt erklärt. Im folgenden Abschnitt wurde die Active-Standby-Implementierung vorgestellt, die einen Zusatz darstellt, der höhere Ausfallsicherheit bietet.

Danach wurde eine Übersicht über die verschiedenen Schnittstellen, die der LSOP-Service anbietet, gegeben. Genauer sind dies die Schnittstellen zum DDC, zu den Anfrageergebnissen und zu SportsQL. Als letztes wurden die GUI-Anwendungen vorgestellt. Im Abschnitt zur Coach App wurde die Erstellung von Anfragen sowie die Architektur der GUI erklärt. Bevor schließlich eine Übersicht über die verschiedenen Funktionen der Monitoring Application gegeben wurde, gab es eine kurze Ausführung zur Developer Application.

7 Evaluation

In den folgenden Teilkapiteln werden die einzelnen Evaluationen und ihre Ergebnisse vorgestellt. Zu Beginn wird die Evaluation der Anfragen vorgestellt. Im Anschluss daran folgen das Load-Balancing und das Recovery. Anschließend werden die einzelnen Evaluationen der Coach App und der GPSender App präsentiert. Den Abschluss bildet die Monitoring App.

7.1 Verwendete Hardware

Die Tabelle 7.1 gibt eine Übersicht über die bei der Evaluation verwendete Hardware. Im weiteren Verlauf wird sich auf die Namen der Rechner bezogen.

Name	Betriebssystem	CPU	Speicher
Peer1	Linux 3.18.5-1-ARCH	Intel i7: 3,5 GHz	8 GB
Peer2	OS X Yosemite	Intel i7: 3,4 GHz	16 GB
Peer3	Microsoft Windows 7 Professional	Intel i5: 1,6 GHz	4 GB
Peer4	Microsoft Windows 8.1 Pro	Intel i5: 1,6 GHz	8 GB
Peer5	Microsoft Windows 8.1	Intel i3: 1,8 GHz	4 GB
Peer6	Microsoft Windows 8.1	Intel i5: 1.6 GHz	8 GB
Peer7	Microsoft Windows 8.1	Intel i7: 2,1 GHz	6 GB
Peer8	Microsoft Windows 7 Professional	Intel Core2Duo: 2,53 GHz	4 GB
Peer9	Microsoft Windows 10 Pro Technical Preview	Intel i7: 1,7 GHz	8 GB
Peer10	OS X Yosemite	Intel i5: 1,3 GHz	8 GB
Peer11	OS X Mountain Lion	Intel Core2Duo: 2.4GHz	8 GB
Peer12	Microsoft Windows 7 Professional	Intel i5: 2.27 GHz	8 GB

Tabelle 7.1: Verwendete Hardware

7.2 Anfragen

In diesem Teilkapitel werden die Evaluation der einzelnen Anfragen beschrieben. Dabei wird sowohl der Aufbau, die Durchführung als auch das Ergebnis und die Korrektheit vorgestellt.

7.2.1 Aufbau

Im Folgenden wird genauer beschrieben, wie die erstellten Anfragen evaluiert wurden. Hierzu wird zunächst der Aufbau der Evaluation erläutert. Dies ist für die Interpretation der Ergebnisse von Bedeutung und ermöglicht eine Reproduktion der Tests und Ergebnisse.

7.2.1.1 Quelle

Als Datengrundlage wird die DEBS-Quelle *soccergame_ 10_ 20.csv* verwendet. Dazu wird das passende DDC genutzt, welches auch in Listing A.3 im Anhang zu finden ist. Im Vergleich zu den Ursprungsdaten der DEBS Grand Challenge 2013 ist die genutzte Quelle kleiner, da die Datenrate verringert wurde. Durch die geringere Datenrate kann das gesamte Spiel ohne nennenswerten Qualitätsverlust bei den Anfragen deutlich schneller verarbeitet werden.

7.2.1.2 Verwendetes Produkt

Als Produkt wird das LSOP-Produkt genutzt. Alle genutzten Peers hatten dabei die exakt gleiche Versionsnummer. Das LSOP-Produkt enthält alle Features aus dem P2P-Bereich, die Anfragen für die Sportanalyse und auch das Load-Balancing und das Recovery.

7.2.1.3 Größe des Netzwerks

Das Netzwerk besteht aus einer Anzahl von heterogenen Rechnern. Die Größe des Netzwerks ist von der jeweiligen Anfrage abhängig und ist dort gesondert aufgeführt. Alle Rechner sind über einen isolierten Access Point miteinander verbunden und befinden sich in der gleichen Peer-Gruppe.

7.2.1.4 Messungen

Für die jeweiligen Evaluationsszenarien müssen unterschiedliche Werte gemessen werden, um eine Aussage aus der Evaluation ziehen zu können. Dabei wird zunächst geprüft, ob die Anfrage überhaupt Ergebnisse liefert. Kommen Ergebnisse an, werden diese auf ihre Korrektheit geprüft. Die Ergebnisse dieser Überprüfung sind im Abschnitt 7.2.3 zu finden. Um die Geschwindigkeit der Anfrage zu messen, werden die *Latenz* und der *Durchsatz* erfasst. Diese werden automatisch in eine csv-Datei geschrieben, welche in dieser Evaluation auch als Senke für die Ergebnisse der Anfrage fungiert. Der Durchsatz und die Latenz sind zwei zentrale Metriken der Datenstromverarbeitung. Der Durchsatz gibt an, wie viele Elemente pro Sekunde durch das System verarbeitet werden können. Damit dieser Wert nicht zu stark von Schwankungen des Datenstroms beeinflusst wird, werden hierbei eine festgelegte Anzahl an Elementen gezählt und dann berechnet, wie viele Elemente im Durchschnitt pro Sekunde verarbeitet wurden. Neben dem Durchsatz wird auch die Latenz ermittelt, die angibt, wie lange ein Datenstromelement vom Beginn der Verarbeitung bis zum Verlassen des Systems benötigt. Die Ergebnisse des Performanz-Tests sind in Anschnitt 7.2.2 dargestellt.

7.2.1.5 Szenarien

Für die Ermittlung der Performanz wurden drei repräsentative Anfragen ausgewählt, bei denen der Durchsatz und die Latenz ermittelt wurden. Um dabei das Spektrum alle Anfragen möglichst gut abzudecken, wurde die Anfrage mit der geringsten (Analysezeit-Anfrage), eine mit mittlerer (Ballkontakt-Anfrage) und die mit der größten Komplexität (Torschuss-Anfrage) ausgewählt. Dies ist auch den Abbildungen 7.1 und 7.2 zu entnehmen.

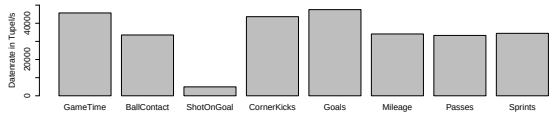


Abbildung 7.1: Vergleich des Durchsatzes

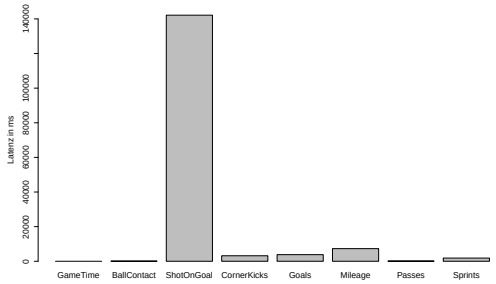


Abbildung 7.2: Vergleich der Latenz

Generell werden für alle Anfragen Latenz und Durchsatz zwischen verschiedenen Szenarien verglichen. Folgende drei Szenarien werden unterschieden:

Lokal Die Anfrage wird auf einem Rechner und auf nur einer Odysseus Instanz (Peer) ausgeführt. Dabei müssen keine Daten über das Netzwerk an andere Rechner oder Peers übertragen werden.

Lokal verteilt Die Anfrage wird auf mehreren Peers durchgeführt. Diese befinden sich allerdings alle auf dem gleichen Rechner. Für die Übertragung der Daten an einen anderen Peer werden bereits netzwerkspezifische Bereiche in Odysseus und im Betriebssystem genutzt. Die Daten werden jedoch nicht direkt über das Netzwerk übertragen, sondern ausschließlich an die Loopback-Adresse gesendet.

Verteilt Die Anfrage wird auf mehreren Peers ausgeführt, welche alle auf unterschiedlichen Rechnern gestartet sind. Für die Ausführung müssen die Daten über das Netzwerk übertragen werden. Hier ist vor allem die Stabilität und Übertragungsgeschwindigkeit des WLAN von entscheidender Bedeutung.

Alle Szenarien wurden für jede Anfrage fünf Mal ausgeführt. Dies gewährleistet, dass die gemessenen Latenz- und Durchsatzwerte vergleichbar sind. Andernfalls könnte es bei einer einmaligen Durchführung dazu kommen, dass Ausreißer zu stark in die Gewichtung mit einfließen.

7.2.2 Performanz

In diesem Abschnitt werden nun die Ergebnisse der Geschwindigkeitsmessung der Anfragen näher erläutert. Dabei werden die Analysezeit-Anfrage, Ballkontakt-Anfrage und die Torschuss-Anfrage getrennt voneinander beschrieben. Im Anschluss daran werden die Ergebnisse des Lasttestes diskutiert, bei dem alle bestehenden Anfragen gleichzeitig gestartet wurden. Eine echte Verteilung der Anfragen auf mehrere Peers war leider nicht möglich, da hierbei keine Ergebnisse ankamen.

7.2.2.1 Analysezeit-Anfrage

Die Analysezeit-Anfrage ist die einfachste Anfrage, die im Rahmen dieser Projektgruppe erstellt wurde. Sie gibt ausschließlich die Zeit der Analyse seit ihrem Beginn aus. Allerdings basiert die Anfrage dennoch auf den verarbeiteten Tupeln und muss somit auch alle verarbeiten. Die Aufteilung

der Anfrage auf verschiedene Peers wurde mit der Partitionierungsstrategie *OperatorCloud* verteilt. Dies ist die einzige Strategie, bei der eine Verteilung auf mehrere Peers möglich ist, da die Anfrage sehr kurz ist. Bei der Auswahl der Peers wird der Allokator *RoundRobin* genutzt, der alle bekannten Peers der Reihe nach auswählt.

Die Abbildung 7.3 zeigt den Anfrageplan der Analysezeit-Anfrage bei einer verteilten Ausführung. Dabei sind die verschiedenen Peers farblich hervorgehoben. Es gibt folgende Kodierung: *Peer6* und *Peer9*. Für eine verbesserte Übersicht ist die Darstellung aufgeteilt. Der erste Teil der Anfrage ist links dargestellt, gefolgt vom zweiten Teil auf der rechten Seite. Die Farbe der Pfeile gibt an, welche Teile der Anfrage verbunden sind.

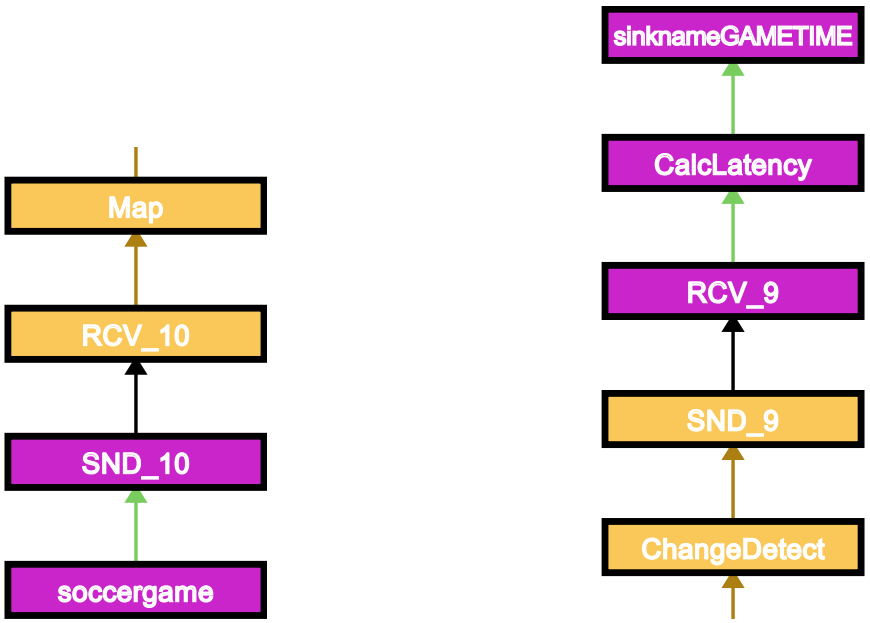


Abbildung 7.3: Verteilter Anfrageplan der Analysezeit-Anfrage mit zwei Peers

Die Abbildungen 7.4 (Durchsatz) und 7.5 (Latenz) zeigen die Ergebnisse der Auswertung der Analysezeit-Anfrage. Sowohl Durchsatz als auch Latenz werden als Boxplots visualisiert, da auf diese Weise alle wichtigen Informationen abgelesen werden können und Ausreißer sich nur minimal auf die Ergebnisse auswirken.

Beim Vergleich des Durchsatzes ist deutlich zu erkennen, dass dieser bei einer rein lokalen Ausführung deutlich höher ist, als wenn die Anfrage auf zwei Peers verteilt wird. Wird die Analysezeit-Anfrage auf zwei Peers verteilt, sinkt der Durchsatz von knapp 50000 Tupel/s auf unter 10000 Tupel/s, wenn die Peers auf dem selben Rechner gestartet wurden. Liegen die Peers auf verschiedenen Rechnern sinkt der Durchsatz nochmal deutlich auf einen Wert von unter 5000 Tupel/s. Dementsprechend verhält es sich auch mit der Latenz bei der Analysezeit-Anfrage. Diese liegt sowohl lokal als auch lokal verteilt bei wenigen Millisekunden. Liegen die Peers auf verschiedenen Rechnern und müssen die Daten über das Netzwerk übertragen werden, steigt die Latenz deutlich auf einen Wert zwischen 50 und 400 Millisekunden. Diese breite Streuung liegt vor allem an Schwankungen im Netzwerk.

Die Evaluation der Analysezeit-Anfrage zeigt, dass es sich insbesondere bei wenig komplexen Anfragen nicht lohnt, sie auf mehrere Peers zu verteilen. Insbesondere bei einer Verteilung auf mehrere

Rechner werden Latenz und Durchsatz deutlich schlechter. Daraus lässt sich schließen, dass die Übertragung über das Netzwerk der limitierende Faktor für die Gesamtperformanz darstellt. Lediglich bei der Verwendung von sehr leistungsschwachen Rechnern wie den Raspberry Pis könnte es sinnvoll sein, einzelne Teile der Anfrage zu verteilen. Dennoch war bei der Evaluation zu beobachten, dass der erste Peer, auf dem die Quelle installiert ist, eine hohe Auslastung hatte. Alle nachfolgenden Peers sind aufgrund der Netzwerkübertragung wenig ausgelastet, da die Datenrate im Anschluss deutlich sinkt und somit auch die Belastung des Systems.

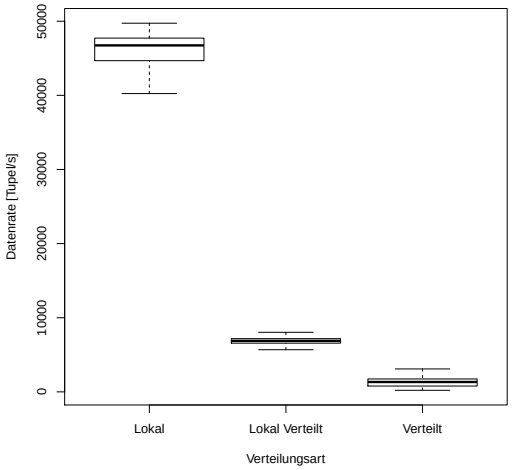


Abbildung 7.4: Vergleich des Durchsatzes (Analysezeit-Anfrage)

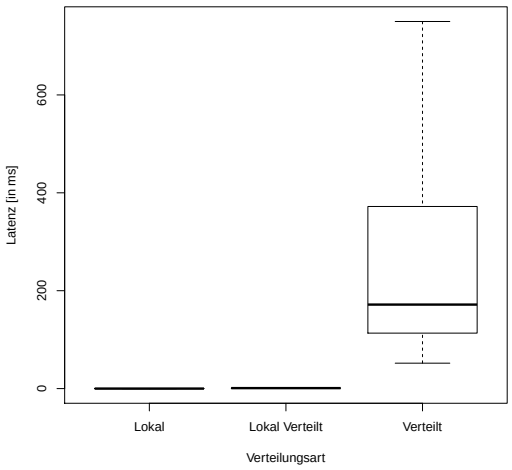


Abbildung 7.5: Vergleich der Latenz (Analysezeit-Anfrage)

7.2.2.2 Ballkontakt-Anfrage

Die Ballkontakt-Anfrage stellt eine für uns durchschnittlich komplexe Anfrage dar. Ziel hierbei ist es, die Ballkontakte der Spieler zu ermitteln. Für die Partitionierung wird im Vergleich zur Analysezeit-Anfrage nicht die Strategie *OperatorCloud*, sondern stattdessen *OperatorSetCloud* verwendet. Diese gewährleistet, dass nicht einzelne Operatoren verschoben werden, sondern zusammenhängende Blöcke mit mehreren Operatoren. Eine Verteilung auf Operatorebene ist bei dieser Anfrage nicht sinnvoll, da die Daten dann sehr oft zwischen den verschiedenen Peers ausgetauscht werden müssten. Als Allokator wird auch hier *RoundRobin* eingesetzt.

Die Abbildung 7.6 zeigt den Anfrageplan der Ballkontakte-Anfrage bei einer verteilten Ausführung. Dabei werden die verschiedenen Peers farblich hervorgehoben: *Peer6*, *Peer2*, *Peer7* und *Peer9*. Für eine verbesserte Übersicht ist die Darstellung aufgeteilt. Der erste Teil der Anfrage ist links, der zweite Teil mittig und der dritte Teil auf der rechten Seite zu finden. Die Farbe der Pfeile gibt an, welche Teile der Anfrage verbunden sind.

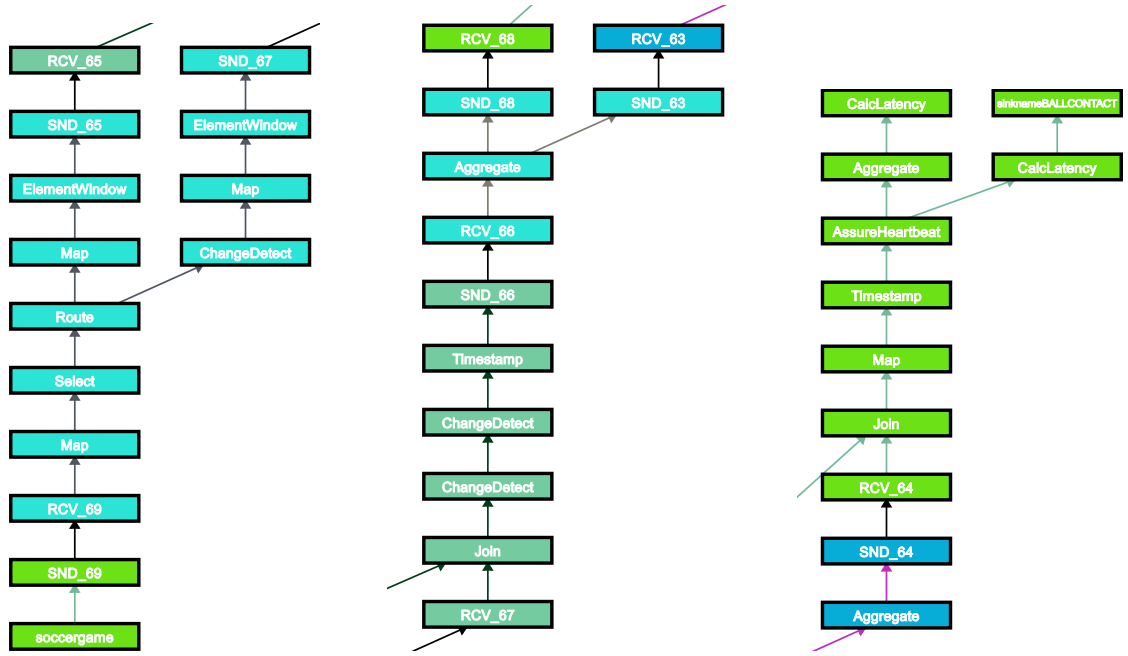


Abbildung 7.6: Verteilter Anfrageplan der Ballkontakt-Anfrage mit vier Peers

Die Abbildungen 7.7 (Durchsatz) und 7.8 (Latenz) zeigen die Ergebnisse der Auswertung der Ballkontakt-Anfrage. Sowohl Durchsatz als auch Latenz werden als Boxplots visualisiert, da auf diese Weise alle wichtigen Informationen abgelesen werden können und Ausreißer sich nur minimal auf die Ergebnisse auswirken.

Der Durchsatz und die Latenz verhalten sich bei dieser Anfrage ähnlich wie bei der zuvor beschriebenen Analysezeit-Anfrage. Im Vergleich dazu liegt der Durchsatz allerdings mit 30000 Tupel/s auch bei einer lokalen Ausführung auf einem deutlichen geringerem Niveau. Dies ist darin begründet, dass die Ermittlung von Ballkontakten deutlich komplizierter ist als das Ausgeben der Analysezeit. Wie auch bei der Analysezeit-Anfrage verringert sich der Durchsatz deutlich, wenn die Anfrage auf mehreren Peers ausgeführt wird. Bei einer Verteilung auf vier lokale Peers liegt der Durchsatz nur noch

bei 5000 Tupel/s. Wird die Anfrage auf verschiedene Rechner verteilt und ist somit eine Übertragung über das Netzwerk notwendig, sinkt der Wert auf ungefähr 1000 Tupel/s. Entsprechend analog zur Analysezeit-Anfrage verhält sich auch die Latenz in Bezug auf die jeweilige Verteilung. Lokal liegt der Wert im Bereich zwischen 100 und 200 ms. Wird die Anfrage auf lokale Peers verteilt steigt die Latenz schon auf 1000 bis 2000 ms. Darüber hinaus erhöht sich auch die Streuung der Werte, die in der Übertragung an einen anderen Peer begründet liegt. Da die Start- und Endzeitstempel für die Latenzberechnung auf dem gleichen Peer erfasst werden, kann es hierbei nicht zu dem Problem kommen, dass die Uhren der verschiedenen Rechner nicht synchronisiert sind und dadurch verzerrte Werte zustande kommen.

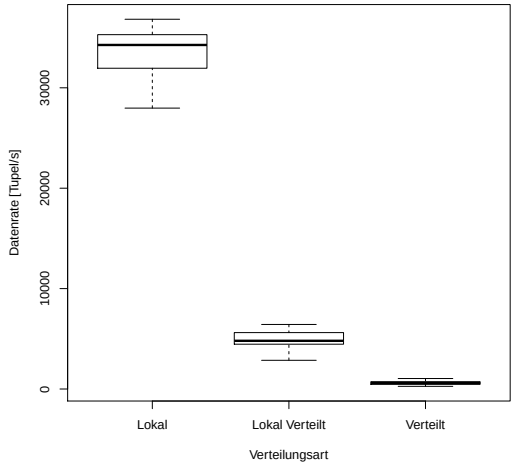


Abbildung 7.7: Vergleich des Durchsatzes (Ballkontakt-Anfrage)

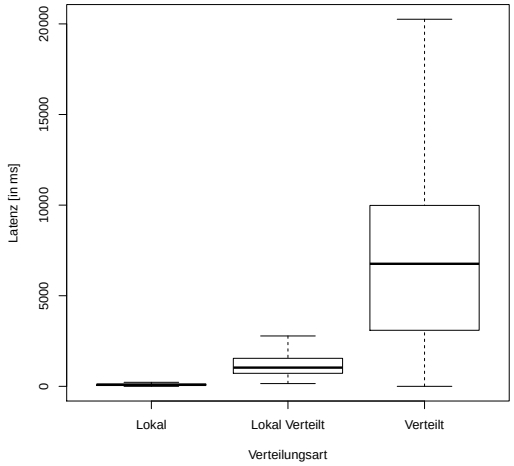


Abbildung 7.8: Vergleich der Latenz (Ballkontakt-Anfrage)

7.2.2.3 Torschuss-Anfrage

Die Torschuss-Anfrage stellt die mit Abstand komplexeste Anfrage dar, die im Rahmen der Projektgruppe entwickelt wurde. Ziel ist es, basierend auf der Flugbahn, der Geschwindigkeit und Beschleunigung des Balles sowie der Position des Tores einen möglichen Schuss auf das Tor zu erkennen. Diese Ermittlung benötigt eine Vielzahl von komplexen Berechnungen, wie zum Beispiel Aggregationen. Wie auch bei der Ballkontakt-Anfrage wurde auch hier die *OperatorSetCloud*-Strategie angewandt und der *RoundRobin*-Allokator eingesetzt.

Die Abbildung 7.9 zeigt den Anfrageplan der Torschuss-Anfrage bei einer verteilten Ausführung. Dabei werden die verschiedenen Peers farblich hervorgehoben: *Peer6*, *Peer7* und *Peer2*. Für eine verbesserte Übersicht ist die Darstellung aufgeteilt. Der erste Teil der Anfrage ist dabei links, der zweite Teil mittig und der dritte Teil auf der rechten Seite dargestellt. Die Farbe der Pfeile gibt an, welche Teile der Anfrage verbunden sind.

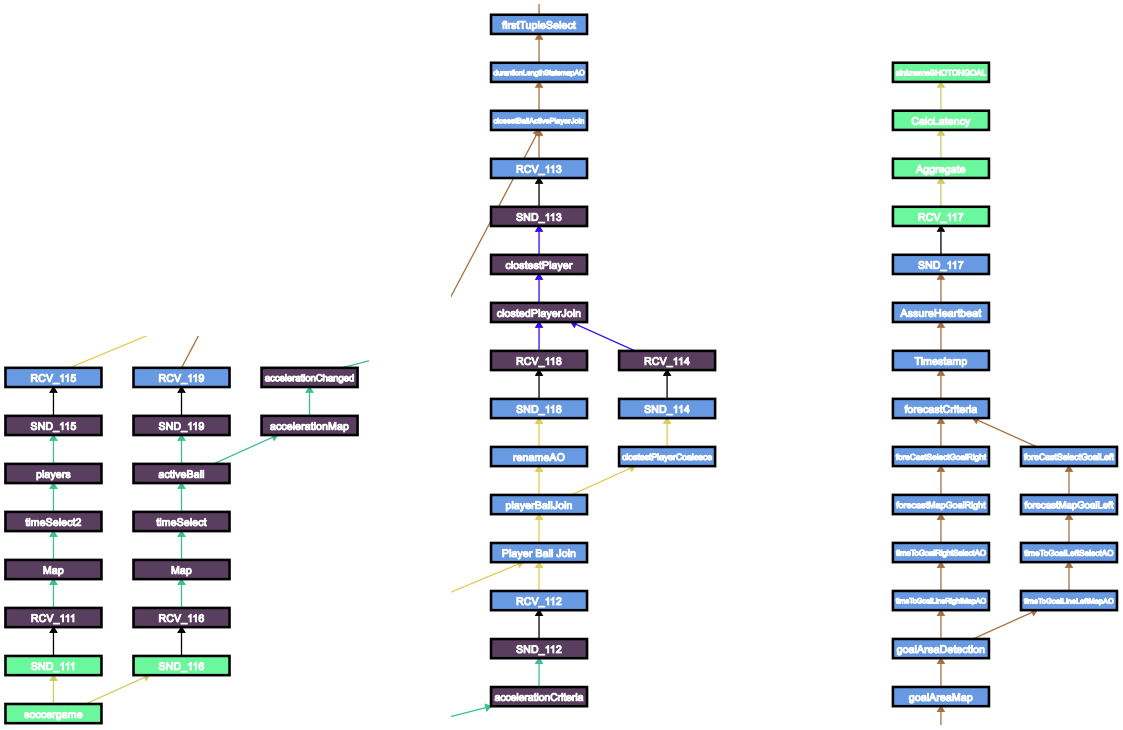


Abbildung 7.9: Verteilter Anfrageplan der Torschuss-Anfrage mit drei Peers

Die Abbildungen 7.10 (Durchsatz) und 7.11 (Latenz) zeigen die Ergebnisse der Auswertung der Torschuss-Anfrage. Sowohl Durchsatz als auch Latenz werden als Boxplots visualisiert, da auf diese Weise alle wichtigen Informationen abgelesen werden können und Ausreißer sich nur minimal auf die Ergebnisse auswirken.

Bei der Torschuss-Anfrage handelt es sich um die komplexeste Anfrage, die im Rahmen der Projektgruppe erstellt wurde. Dies ist an den gemessenen Werten deutlich zu erkennen, da der Durchsatz sowohl bei der lokalen als auch bei der lokal verteilten Ausführung im Mittel bei 3000 Tupel/s liegt. Die Anfrage verarbeitet somit 10 mal weniger Tupel pro Sekunde. Dieser Wert ist insbesondere dann entscheidend, wenn viele Anfragen gleichzeitig ausgeführt werden. Da die Daten synchronisiert abgearbeitet werden, stellt diese Anfrage aufgrund ihrer Komplexität die Limitierung des Gesamtsystems dar. Die Unterschiede bei der Latenz sind bei dieser Anfrage deutlich geringer als bei den zuvor beschriebenen Anfragen. Der Grund für die geringen Abweichungen bei der Latenz zwischen lokaler, lokal verteilter und verteilter Ausführung liegen darin, dass die Latenz maßgeblich durch die Verarbeitung innerhalb der Operatoren definiert wird und nicht durch die Übertragung zwischen den Peers über das Netzwerk.

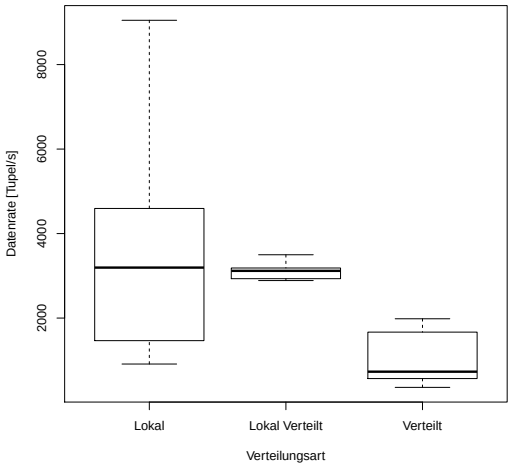


Abbildung 7.10: Vergleich des Durchsatzes (Torschuss-Anfrage)

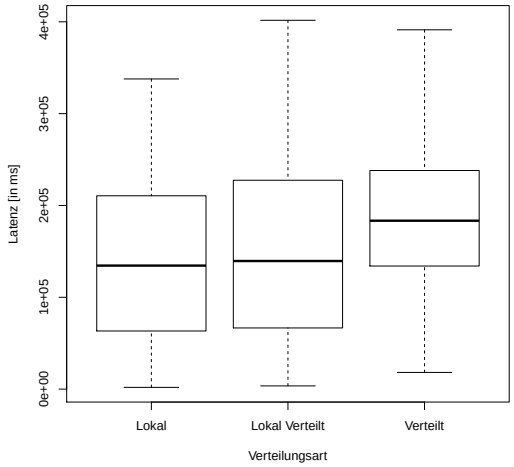


Abbildung 7.11: Vergleich der Latenz (Torschuss-Anfrage)

Diese Evaluation zeigt, dass mit steigender Komplexität der Anfragen, insbesondere durch Aggregationen und Joins, eine Verteilung der Anfrage auf mehrere Peers durchaus sinnvoll sein kann. Aufgrund der hohen Leistung der genutzten Evaluations-Hardware kam es allerdings nicht vor, dass ein Peer komplett überlastet war. Kommt es zu diesem Punkt, dass ein Peer überlastet ist und die Berechnungen nicht leisten kann, dann macht eine Verteilung auf mehrere Rechner durchaus Sinn. Es ist dann zu evaluieren, ob der Vorteil der Lastverteilung gegenüber der Netzwerkübertragung überwiegt.

7.2.2.4 Lasttest

Beim Lasttest wurde evaluiert, wie gut die Performanz bei einer gleichzeitigen Ausführung aller Anfragen ist. Hierbei wurden insgesamt 14 Anfragen (siehe Abbildung 7.12) jeweils auf dem Peer2 *lokal* und *lokal verteilt auf acht Peers* ausgeführt. Um eine Aussage über die Performanz treffen zu können, wurde für jede einzelne Anfrage die Latenz (siehe Abbildung 7.12) und die Datenrate (siehe Abbildung 7.13) gemessen. Bei der Variante *lokal verteilt* wurde folgende Verteilungsstrategie genutzt:

#PEER_ALLOCATE QUERYCOUNT
Durch diese Einstellung wurde festgelegt, dass eine Gleichverteilung der Anfragen über die Peers erfolgt. Dies bedeutet, dass die aktuelle Anzahl der zurzeit ausgeführten Anfragen bei der jeweiligen Verteilung mit einbezogen wird.

#PEER_PARTITION QUERYCLOUD
Mit der Partitionsstrategie *QUERYCLOUD* wird festgelegt, dass immer eine gesamte Anfrage an eine Peer verteilt wird. Eine weitere Partitionierung der Anfragen findet nicht statt.

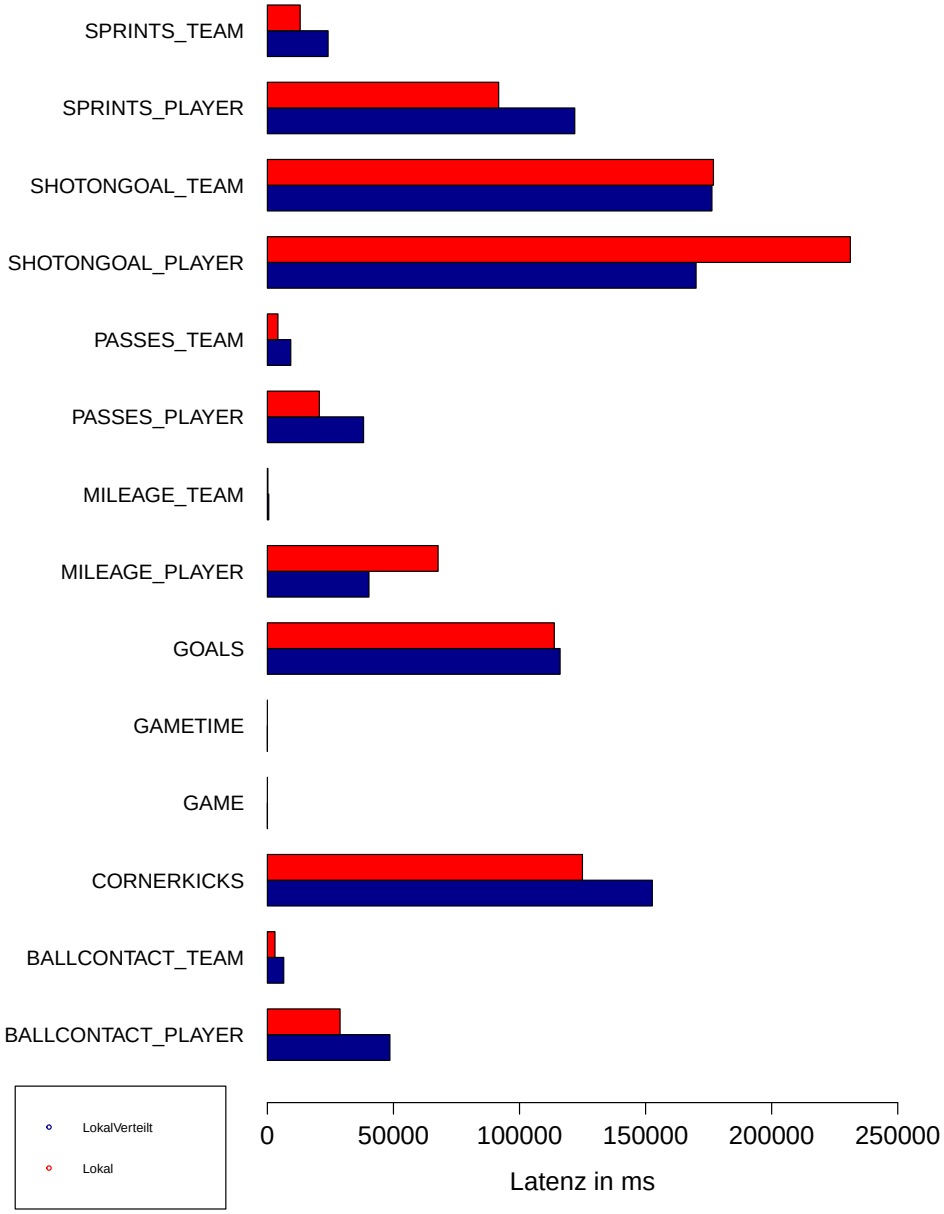


Abbildung 7.12: Vergleich der Latenz

Abbildung 7.12 stellt die gemessenen Latenzen pro Anfrage von *lokal* und *lokal verteilt* gegenüber. Die geringste Latenz konnte bei den Anfragen für die Analysezeit, Spielerposition und Laufstrecke festgestellt werden. Hierbei handelt es sich um recht einfache Anfragen, die mit wenigen Operatoren umgesetzt werden konnten. So werden bei diesen Anfragen ausschließlich kostengünstige Operationen, wie z.B. Projektionen und Selektionen genutzt, was zu einer positiven Auswirkung auf die Latenz

führt. Des Weiteren konnte bei diesen Anfragen kein signifikanter Unterschied zwischen den Latenzen bei der *lokal* und *lokal verteilten* Ausführung festgestellt werden. Hieraus lässt sich schließen, dass bei diesen Anfragen die Netzwerkübertragung keine signifikante Auswirkung auf die Latenz hat.

Andere Anfragen wie Sprints, Schüsse, Laufstrecke, Ecken und Ballkontakte weisen eine wesentlich höhere Latenz auf. Durch weitere Recherchen konnte festgestellt werden, dass diese Anfragen den Operator *AssureHeartbeatAO* einsetzen. Dieser Operator wurde in die jeweiligen Anfragen eingebunden, um kontinuierliche Ausgaben bei diesen Anfragen zu gewährleisten. Würde dieser Operator aus den jeweiligen Anfragen entfernt werden, würden sie sehr selten eine Ausgabe generieren. Dies hätte zur Folge, dass die Coach App z.B. bei den Spielerdetails unter den Pässen Nullwerte anzeigt, obwohl der Spieler bereits erfolgreiche Pässe gespielt hat. Denn nur wenn sich der Trainer in der Detailansicht eines Spielers befindet, wird der entsprechende Datenstrom von der Coach App visualisiert. Diese architektonische Entscheidung wurde getroffen, um das Tablet auf dem die Coach App ausgeführt wird, nicht zu überlasten.

Die erhöhte Latenz bei diesen Anfragen ist somit auf die Kombination von dem *AssureHeartbeatAO*- mit dem *AggregateAO-Operator* zurückzuführen. Tabelle 7.2 zeigt die Tupel-Ausgabe, die von der Torschuss-Anfrage generiert wurde. In der zweiten Spalte befindet sich die *Team-ID*, die das jeweilige Team eindeutig identifiziert. Die Spalte *Anzahl* spiegelt die Gesamtanzahl an erkannten Torschüsse der jeweiligen Mannschaft wieder. Die Angabe der Latenz gibt an, wie Lange das einzelne Tupel bei der Verarbeitung gebraucht hat. In der letzten Spalte wurde die *Latenz* + die verwendete *Heartbeat-Rate* von 5000 ms berechnet. An den Zeilen 1 bis 11 ist zu erkennen, dass sich die Anzahl der Torschüsse nicht verändert, die Latenz aber konstant um 5000 ms steigt. Dies passiert, da der *AggregateAO-Operator* durch die eingestellte *Heartbeat-Rate* von 5000 ms, alle 5000 ms dazu aufgefordert wird, ein neues Ergebnis zu berechnen. Da sich die Anzahl der Torschüsse nicht verändert hat, wird die Latenz aus der Differenz des frühsten Zeitstempels (von dem Tupel aus Zeile 1) und dem aktuellen Zeitstempel gebildet. In der Zeile 12 wird ein erneuter Torschuss erkannt, ab diesen Zeitpunkt wir die Latenz aus der Differenz des Zeitstempel des Tupels aus Zeile 12 mit dem aktuellen Zeitstempel gebildet. Hieraus ergibt sich ein weitaus geringerer Latenzwert. Deshalb ist davon auszugehen, dass die Anfragen in Wirklichkeit eine geringere Latenz aufweisen als in der Abbildung 7.12 dargestellt. Das beschriebene Verhalten, welches in Abbildung 7.12 zu erkennen ist, konnte auch bei den Anfragen für Sprints, Schüsse, Laufstrecke, Ecken und Ballkontakte festgestellt werden.

Die Torschuss-Anfrage weist mit Abstand die höchste Latenz auf. Neben den obengenannten Zusammenspiel zwischen dem *AssureHeartbeatAO*- und dem *AggregateAO-Operator* werden bei dieser Anfrage auch teure Operatoren wie Join-Operatoren verwendet. Des Weiteren weißt die Torschuss-Anfrage Optimierungspotenzial auf. So war es nicht möglich vor den verwendeten Join-Operatoren in dieser Anfrage einen Fenster-Operator korrekt einzubinden. Dies hat zur Konsequenz, dass der jeweilige Join-Operator einen großen Teil des Datenstrom vorhält und somit deutlichen Einfluss auf die Belegung des Arbeitsspeichers sowie des Prozessors hat. Dies wiederum wird vermutlich die Latenz negativ beeinflussen. Die Ursache warum es nicht möglich war ein entsprechendes Fenster vor die Join-Operatoren zu schalten, konnte nicht abschließend geklärt werden. Es wird vermutet, dass es sich um einen Sonderfall bei der Konstellation der Anfrage (wie z.B. Komplexität) und der genutzten Quelldaten handelt.

Ein weiterer wichtiger Wert für die Performanz ist die Datenrate bei der Ausführung der 14 Anfragen. Abbildung 7.13 zeigt hierbei die durchschnittlich gemessene Datenrate der Anfragen. Bei der

Zeilen Nr.	Team ID	Anzahl	Latenz (ms)	Latenz + HEARTBEAT (ms)
1	1	1.0	10886,29	10886,29
2	1	1.0	15887,97	15886,29
3	1	1.0	20889,09	20887,97
4	1	1.0	25889,49	25889,09
5	1	1.0	30892,22	30889,49
6	1	1.0	35893,80	35892,22
7	1	1.0	40898,55	40893,80
8	1	1.0	45899,02	45898,55
9	1	1.0	50904,06	50899,02
10	1	1.0	55905,69	55904,06
11	1	1.0	58743,24	60905,69
12	1	2.0	3938,67	63743,24
13	1	3.0	6162,32	8938,67
14	1	3.0	11164,80	11162,32
15	1	3.0	16165,88	16164,80
16	1	3.0	21170,90	21165,88
17	1	3.0	26174,80	26170,90
18	1	3.0	31179,63	31174,80

Tabelle 7.2: Tupel-Ausgabe der Torschuss Anfrage

Lokal ausgeführten Variante konnte ein Durchsatz von etwa 2400 Tupel/s gemessen werden. Die lokal verteilte Ausführung hat eine durchschnittliche Datenrate von 500 Tupel/s erreicht. Diese stark unterschiedliche Datenrate lässt vermuten, dass bei der lokal verteilten Ausführung die Übertragung über das Netzwerk (Loopback Adapter) einen großen Einfluss auf die Datenrate hat.

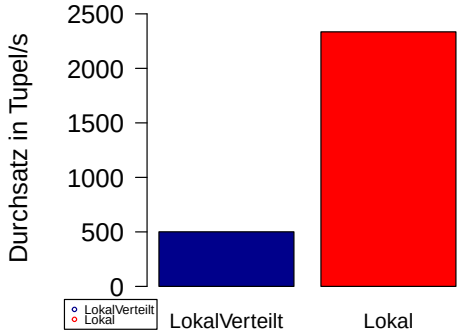


Abbildung 7.13: Vergleich der Datenrate

Der Lasttest hat gezeigt, dass die Ausführung aller Anfragen gleichzeitig möglich ist. Des Weiteren geben alle Anfragen zeitnah die berechneten Ergebnisse zuverlässig aus. Werden die Anfragen *lokal* ausgeführt, konnte überwiegend eine geringe Latenz festgestellt werden. Ausnahmen bilden die Laufstrecken-Anfrage und Torschuss-Anfrage. Diese wiesen eine nahezu gleichbleibende bzw. leicht erhöhte Latenz bei der lokalen Ausführung auf. Diese geringe Abweichung kann vermutlich auf eine Messungenauigkeit zurückgeführt werden. Vergleicht man die Latenz, die bei einer gleichzeitigen Ausführung der Anfragen gemessen wurde, mit denen die bei der Einzelausführung gemessen wurden, kann keine signifikante Änderung festgestellt werden. Hieraus lässt sich schließen, dass sich die Anfragen in Bezug auf die Latenz nicht gegenseitig beeinflussen.

7.2.3 Korrektheit der Ergebnisse

In diesem Abschnitt wird die Korrektheit der Anfragen von Herakles evaluiert. Dazu werden die Fußballdaten von der DEBS Grand Challange 2013 als Datenquelle verwendet. Des Weiteren wird die Videoaufzeichnung des Fußballspiels verwendet, um die Anzahl an Toren, Torschüssen und Ecken herauszufinden.

7.2.3.1 Tore-Anfrage

Für die Evaluation der Tore-Anfrage wird überprüft, ob die durch die Anfrage erkannten Tore mit denen aus der Videoaufzeichnung übereinstimmen. Die Klassifikationstabelle 7.3 gibt einen Überblick darüber, welche Tore von der Anfrage erkannt werden und welche nicht. Es werden durch die Anfrage 8 von 9 Toren erkannt. Somit liegt der Recall für die Tore bei 89%. Ein Tor wird zum Ende

der ersten Halbzeit nicht erkannt, da der Sensor vom Ball ca. ab der 28. Minute keine Daten mehr sendet. Die Precision der Klassifikation liegt bei 100%, da alle Tore, die erkannt werden, auch in der Videoaufzeichnung als Tore zu beobachten sind.

		Vorhergesagt		
Beobachtet		Tor	kein Tor	Summe
	Tor	8	1	9
	kein Tor	0	0	0
	Summe	8	1	9

Tabelle 7.3: Klassifikationstabelle Tore

7.2.3.2 Torschüsse-Anfrage

Zur Evaluation der Torschüsse-Anfrage werden die durch die Anfrage ermittelten Torschüsse mit denen aus der Videoaufzeichnung verglichen. In Abbildung 7.4 wird die dazugehörige Klassifikationstabelle gezeigt. Es werden von den 21 Torschüssen, die in der Videoaufzeichnung zu beobachten sind, 15 Torschüsse korrekt erkannt. Somit liegt der Recall bei 71%. Die Precision der Klassifikation liegt bei 83%, da von den 18 erkannten Torschüssen nur 15 in der Videoaufzeichnung tatsächlich zu beobachten sind. Der Grund dafür, dass einige Torschüsse nicht erkannt werden oder zu viel erkannt werden, sind vielfältig. Beispielsweise ist es mit der Anfrage nur möglich, eine gerade Flugkurve zu berechnen. Wenn ein Torschuss mit sehr viel Effet gespielt wird, kann die Anfrage den Torschuss nicht ermitteln.

		Vorhergesagt		
Beobachtet		Torschuss	kein Torschuss	Summe
	Torschuss	15	6	21
	kein Torschuss	3	14	17
	Summe	18	20	38

Tabelle 7.4: Klassifikationstabelle Torschüsse

7.2.3.3 Ecken-Anfrage

Für die Evaluation der Ecken-Anfrage werden die durch die Anfrage ermittelten Ecken mit denen aus der Videoaufzeichnung verglichen. Die Klassifikationstabelle 7.5 zeigt, wie viele Ecken korrekt erkannt werden. Es werden alle aus der Videoaufzeichnung beobachteten Ecken korrekt erkannt, sodass der Recall bei 100% liegt. Die Precision liegt bei 73%, da von 11 erkannten Ecken nur 8 korrekt sind. Der Grund dafür, dass drei Ecken zu viel erkannt werden, liegt darin, dass der Ball jeweils ins Toraus gespielt wird und anschließend in der Nähe der Eckfahne zurück ins Spielfeld gelangt. Hierbei erkennt die Anfrage fälschlicherweise eine Ecke.

		Vorhergesagt		
Beobachtet		Ecke	keine Ecke	Summe
	Ecke	8	0	8
	keine Ecke	3	0	3
	Summe	11	0	11

Tabelle 7.5: Klassifikationstabelle Ecken

7.2.3.4 Ballkontakte-Anfrage

Zur Evaluation der Ballkontakte-Anfrage wird die Anzahl an Ballkontakten in den ersten fünf Minuten des Spiels betrachtet. Dabei wird überprüft, ob die Anzahl an Ballkontakten, die mit der Anfrage ermittelt werden können, mit denen übereinstimmen, die anhand der Videoaufzeichnung gezählt werden können. Es wird nicht überprüft, ob ein durch die Anfrage erkannter Ballkontakt auch in der Videoaufzeichnung ein Ballkontakt ist. Mit Hilfe der Videoaufzeichnung werden für Team Rot 35 und für Team Gelb 47 Ballkontakte gezählt. Die Anfrage ermittelt für Team Rot 30 und für Team Gelb 46 Ballkontakte. Somit werden mit der Anfrage für Team Rot 86% und für Team Gelb 98% der Ballkontakte erkannt.

7.2.3.5 Pässe-Anfrage

Für die Evaluation der Pässe-Anfrage wird die Anzahl an Pässen in den ersten fünf Minuten des Spiels betrachtet. Dabei wird evaluiert, ob die Anzahl an Pässen, die mit der Anfrage berechnet werden können, mit denen übereinstimmen, die anhand der Videoaufzeichnung gezählt werden können. Es wird nicht überprüft, ob ein durch die Anfrage erkannter Pass auch in der Videoaufzeichnung ein Pass ist. Anhand der Videoaufzeichnung werden für Team Rot 22 Pässe gezählt, während für Team Gelb 39 Pässe ermittelt werden können. Die Anfrage ermittelt für Team Rot 21 und für Team Gelb 39 Pässe. Somit werden mit der Anfrage für Team Rot 95% und für Team Gelb 100% der Pässe erkannt.

7.2.3.6 Sprints-Anfrage

Die Sprints der Spieler können anhand der Videoaufzeichnung nur sehr schwierig ermittelt werden. Daher wird im Folgenden darauf verzichtet und nur eine Abschätzung abgegeben, wie realistisch die erkannte Anzahl an Sprints durch die Anfrage ist. Das Balkendiagramm in Abbildung 7.14 zeigt die Anzahl an Sprints an, die durch die Anfrage pro Spieler ermittelt werden. Anhand des Diagramms kann gesehen werden, dass die beiden Torhüter (1,11) nur jeweils einen Sprint gelaufen sind, während die Feldspieler 4, 16 und 17 die meisten Sprints gelaufen sind. Daran lässt sich erkennen, dass die Anfrage korrekt ermittelt hat, dass die Torhüter i.d.R. weniger Sprints laufen als die Feldspieler. Im Durchschnitt laufen die Feldspieler 16,142 Sprints während der Gesamtspieldauer von 70 Minuten. Hochgerechnet auf 90 Minuten sind dies 20,75 Sprints. Um die Güte dieses Wertes zu beurteilen, wird der Wert mit der Anzahl an Sprints verglichen, die in der Bundesliga von einem Spieler gelaufen werden. Nach [Spr13] laufen die Spieler mit den meisten Sprints in der Bundesliga im Durchschnitt zwischen 25 und 30 Sprints. Somit kann der ermittelte Wert durch die Anfrage von 20,75 als realistisch eingestuft werden.

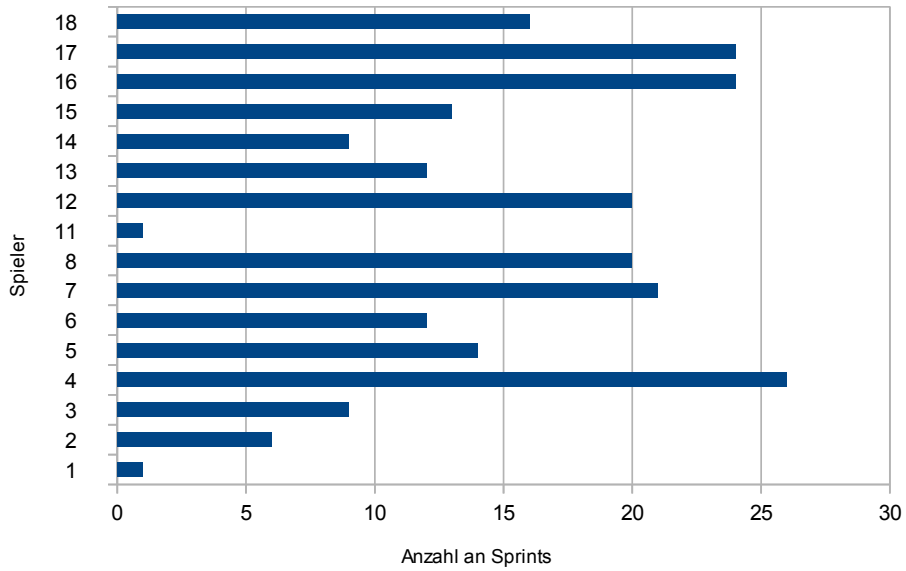


Abbildung 7.14: Balkendiagramm für die Anzahl der Sprints pro Spieler

7.2.3.7 Laufstrecke-Anfrage

Die Laufstrecke kann ebenfalls nur sehr schwierig anhand der Videoaufzeichnung ermittelt werden. Aus diesem Grund wird im Folgenden nur eine Abschätzung abgegeben, ob die ermittelte Laufstrecke realistisch ist. In Abbildung 7.15 wird ein Balkendiagramm dargestellt, welches anzeigt, wie viele Kilometer ein Spieler gelaufen ist. Es fällt auf, dass die Anfrage korrekt ermittelt hat, dass die Torhüter (1,11) i.d.R. deutlich weniger laufen als die Feldspieler. Die Feldspieler laufen im Durchschnitt 4,22 Kilometer in der Gesamtspieldauer von 70 Minuten. Auf 90 Minuten hochgerechnet beträgt die durchschnittliche Laufstrecke 5,43 Kilometer. Nach [Lau13] laufen die Spieler mit der höchsten Laufstrecke in der Bundesliga ca. 12 Kilometer. Der durch die Anfrage ermittelte Wert ist somit deutlich geringer. Dennoch kann der Wert als realistisch angesehen werden, da das Spiel auf einem Halbfeld statt finden und die Spieler nur Amateure sind.

7.2.3.8 Weitere Anfragen

Für die folgenden Anfragen lässt sich eine genaue Evaluation nicht durchführen, da sie entweder trivial oder zu aufwändig wäre. Daher werden die Anfragen nur grob auf Korrektheit beurteilt. Die Analysezeit-Anfrage kann als korrekt angesehen werden, da sie die Analysezeit aufsteigend nach Minuten und Sekunden ausgibt und bei Minute 0 beginnt. Die Spielerpositionen-Anfrage liefert ebenfalls korrekte Ergebnisse, da die Spieler auf dem Spielfeld innerhalb der Coach Application ähnlich positioniert sind, wie sie es auch in der Videoaufzeichnung sind. Des Weiteren kann die Heatmap-Anfrage als korrekt angesehen werden, da die Kacheln einer Heatmap in den Bereichen hervorgehoben werden, in denen sich ein Spieler nach der Videoaufzeichnung auch am häufigsten befindet.

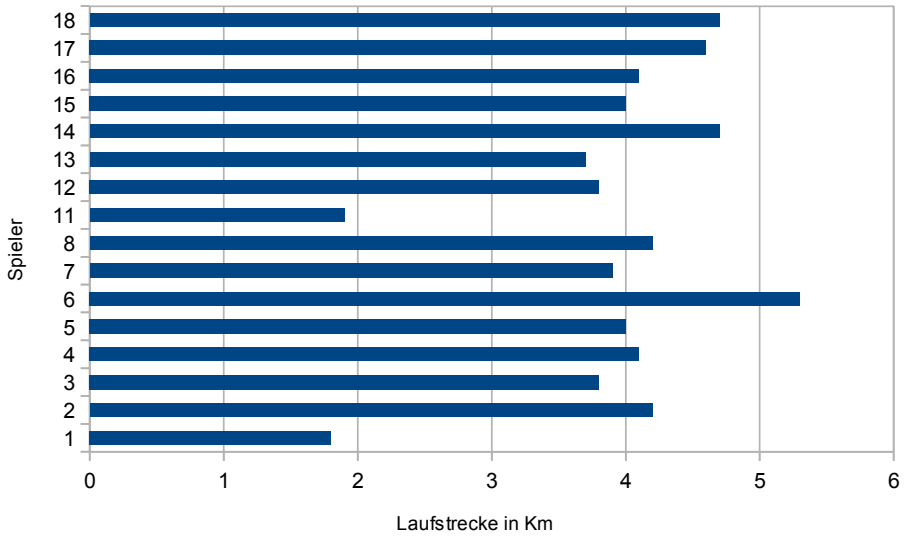


Abbildung 7.15: Balkendiagramm für die Laufstrecke pro Spieler

7.3 Load-Balancing

Um die Funktionsfähigkeit und die Qualität des in Abschnitt 5.5 konzipierten und gemäß Abschnitt 6.6 implementierten Load-Balancings zu überprüfen, wurde eine experimentelle Evaluation durchgeführt. Da in der Projektgruppe vor allem Methoden zur Verschiebung laufender Anfragen entwickelt wurden, wird auch in der Evaluation der Schwerpunkt auf diese Punkte gelegt. Deshalb werden sowohl die Neuverteilung zustandsloser Anfragen durch den Parallel-Track-Kommunikator, als auch die Neuverteilung zustandsbehafteter Anfragen durch der Moving-State-Kommunikator evaluiert, während die implementierte Load-Balancing-Strategie und der Round-Robin-Allokator nicht explizit evaluiert werden. Allerdings werden diese Komponenten durch die durchgeführten Evaluationen implizit mitgetestet, so dass zumindest deren Funktionalität gezeigt wird. Im Folgenden wird zunächst die Vorgehensweise bei der Evaluation erläutert. Danach werden getrennt jeweils die Ergebnisse für die Übertragung zustandsloser und zustandsbehafteter Anfragen vorgestellt. Zum Schluss werden diese Ergebnisse kurz zusammengefasst.

7.3.1 Vorgehensweise

Um das Load-Balancing zu evaluieren, wird zunächst jeweils eine Anfrage mit und ohne zustandsbehaftete Operatoren ohne eingeschaltetes Load-Balancing auf zwei Peers verteilt und ausgeführt. Dabei werden Prozessorlast, Latenz und Durchsatz gemessen. Als Peers werden Peer1 und Peer11 verwendet. Danach wird die gleiche Anfrage mit aktiviertem Load-Balancing ausgeführt und die Werte erneut gemessen. Danach werden diese Werte den Referenzwerten gegenübergestellt und ausgewertet. Für die Übertragung von zustandslosen Anfragen wird der Parallel-Track-Kommunikator und für die Übertragung zustandsbehafteter Anfragen der Moving-State-Kommunikator verwendet.

Als Datenquelle wird die DEBS-Quelle soccergame_ 10_ 20.csv verwendet und jede Anfrage wird aus Zeitgründen nach 10 Minuten beendet. Um zu vermeiden, dass Ausreißer die Ergebnisse beeinflussen wird jedes Versuchsszenario insgesamt fünfmal durchgeführt.

Bei der Verteilung der Anfragen wird bewusst versucht, eine möglichst ungünstige Ausgangsverteilung zu erzeugen. Dazu wird als Fragmentierungsstrategie `QUERYSETCLOUD` und zur Verteilung `ROUNDROBIN` genutzt. Dadurch entsteht ein hohes Lastgefälle zwischen Peer11 und Peer1: Während fast die komplette Berechnung der Anfragen auf Peer11 verlagert wird, ist Peer1 beinahe nur mit dem Auslesen und Senden der Daten beschäftigt. Dadurch wird eine Situation hergestellt, in der ein intelligentes Load-Balancing sinnvoll ist.

Als Anfrage für die zustandslose Übertragung wurde die Spielerpositions-Anfrage ausgewählt. Diese Anfrage besitzt zwar keinen Zustand, erzeugt aber eine sehr hohe Datenmenge. Dadurch erzeugt die Anfrage ausreichend Prozessorlast um ein sinnvolles Load-Balancing durchzuführen. Listing A.4 im Anhang zeigt das verwendete SportsQL um die Anfrage zu starten. Der Anfrageplan in Abbildung A.1 im Anhang zeigt, wie dieser Anfrageplan auf die beiden Peers verteilt wurde. Als Auslösungsschwelle für das Load-Balancing wird eine Prozessorauslastung von 70% festgelegt und der Timeout für die Synchronisation wird auf eine Minute festgesetzt.

Für die zustandsbehaftete Übertragung wurde als Anfrage die Ballkontakte-Anfrage ausgewählt. Zwar gibt es deutlich komplexere Anfragen, wie zum Beispiel die Torschuss-Anfrage, diese erzeugen aber nur sporadisch Ergebnisse. Um einen gleichmäßigeren und damit besser auswertbaren Ausgangsstrom zu haben wurde deshalb die Ballkontakte-Anfrage ausgewählt. Listing A.5 im Anhang zeigt auch hier das verwendete SportsQL. Des Weiteren zeigt Abbildung A.2 im Anhang, wie dieser Anfrageplan auf die beiden Peers verteilt wurde. Als Auslösungsschwelle für das Load-Balancing wird hier eine Prozessorauslastung von 65% festgelegt.

Nachdem nun die Vorgehensweise erläutert wurde, folgt im nächsten Abschnitt die Vorstellung der Ergebnisse der Evaluation.

7.3.2 Ergebnisse

In diesem Abschnitt werden die Ergebnisse der Evaluation vorgestellt. Dabei wird zwischen dem Load-Balancing für zustandslose Anfragen und dem Load-Balancing für zustandsbehaftete Anfragen unterschieden. Zustandslose Anfragen werden mit dem Parallel-Track-Kommunikator und zustandsbehaftete Anfragen werden mit den Moving-State-Kommunikator übertragen. Zum Schluss folgt eine kurze Auswertung der Ergebnisse.

7.3.2.1 Load-Balancing bei zustandslosen Anfragen

In diesem Abschnitt werden die Ergebnisse der Evaluation des Load-Balancings für zustandslose Anfragen vorgestellt. Die (inhaltliche) Vergleichbarkeit der Ergebnisse von Läufen mit und ohne Load-Balancing wurde überprüft, indem die Ergebnisse der Läufe miteinander verglichen wurden. Da durch das Netzwerk ein deutlicher Geschwindigkeitsunterschied zwischen den jeweiligen Datenströmen auf beiden Peers aufgetreten ist und kein Peer tatsächlich an seiner Leistungsgrenze war, wurde dabei beim Synchronisations-Operator fast immer nach einer Minute der Timeout ausgelöst. Dieses vorzeitige Ende der Synchronisierung führt zwangsläufig zu einem Tupelverlust in der Verarbeitung. Dessen Ausmaß hängt insbesondere von der Datenrate der Anfrage, dem gewählten Timeout

und dem Netzwerk ab und wurde daher nicht weiter gemessen. Grundsätzlich lässt sich dieser Verlust durch eine bessere Strategie (wie zum Beispiel die in Abschnitt 6.6 beschriebene punctuation-basierte Synchronisation) komplett vermeiden oder durch einen höheren Timeout umgehen. Generell und besonders bei zustandslosen Anfragen stellt dieser Datenverlust aber kein größeres Problem dar, da sich Datenströme im Laufe der Zeit selbst regenerieren.

Lauf	**1**	**2**	**3**	**4**	**5**
durchschnittl. Prozessorlast Peer1	4,03%	3,15%	2,73%	3,09%	3,33%
durchschnittl. Prozessorlast Peer11	84,69%	80,94%	67,60%	81,29%	84,59%
durchschnittl. Latenz (in ms)	1.385	1.842	3.731	2.119	1.606
durchschnittl. Durchsatz (Tupel/s)	1.790	2.309	4.466	3.772	3.192

Tabelle 7.6: Durchschnittswerte Spielerposition ohne Load-Balancing

Tabelle 7.6 zeigt die Durchschnittswerte für Durchsatz, Latenz und Prozessorlast auf beiden Peers für alle Referenzläufe ohne Load-Balancing. Grundsätzlich scheint die Prozesslorlast auf Peer11 etwa bei 80-85% zu liegen (mit Ausnahme von Lauf 3), während die Auslastung von Peer1 stets unter 5% ist. Der durchschnittliche Durchsatz liegt zwischen 1.790 und 4.466 Tupeln pro Sekunde und die durchschnittliche Latenz zwischen 1.385 und 3.731 Millisekunden. Insgesamt lässt sich erkennen, dass, insbesondere bezüglich der Prozessorauslastung, deutliche Optimierungsmöglichkeiten vorhanden sind.

Lauf	**1**	**2**	**3**	**4**	**5**
durchschnittl. Prozessorlast Peer1	18,01%	5,18%	14,23%	20,94%	19,17%
durchschnittl. Prozessorlast Peer11	46,26%	57,98%	43,93%	48,44%	48,37%
durchschnittl. Latenz (in ms)	899	2.782	10.682	4.864	8.956
durchschnittl. Durchsatz (Tupel/s)	3.990	4.476	5.405	4.899	5.168

Tabelle 7.7: Durchschnittswerte Spielerposition mit Load-Balancing

Tabelle 7.7 zeigt die Durchschnittswerte für Durchsatz, Latenz und Prozessorlast mit aktiviertem Load-Balancing für die verschiedenen Läufe. Es fällt auf, dass die durchschnittliche Prozessorlast für Peer11 zwischen 43% und 57% liegt und damit deutlich niedriger ausfällt als ohne Load-Balancing. Gleichzeitig erhöht sich die durchschnittliche Auslastung von Peer1. Sie liegt nun zwischen 5% und 20%. Die großen Unterschiede zwischen den verschiedenen Läufen sind zum Teil durch unterschiedliche Load-Balancing-Zeitpunkte, zum anderen durch unterschiedliche Umverteilungen erklärbar. Je nachdem, welcher Anfrageteil verschoben wird, wird auch mehr oder weniger zusätzliche Prozessorlast erzeugt. Die durchschnittliche Latenz liegt zum Teil niedriger, zum Teil höher als bei den Vergleichsläufen ohne Load-Balancing, der Durchsatz liegt generell etwa 1.000-2.000 Tupel pro Sekunde höher. Dies deutet darauf hin, dass durch das Load-Balancing Geschwindigkeitsvorteile erzielt werden können, aber nicht zwangsläufig auch müssen. Je nachdem welcher Teil der Anfrage verschoben wird, wirkt sich dies unterschiedlich aus. Dies wird im Folgenden am zeitlichen Verlauf der Messwerte für einzelne Läufe näher untersucht.

Prozessorlast

Abbildung 7.16 zeigt den Verlauf der Prozessorlast auf den beiden Peers ohne aktiviertes Load-Balancing. Man erkennt auch hier, dass eine sehr ungünstige Lastverteilung vorliegt: Die Auslastung von Peer1 liegt konstant nur bei etwa 5%, während die Auslastung von Peer11 durchgehend bei etwa 80% liegt.

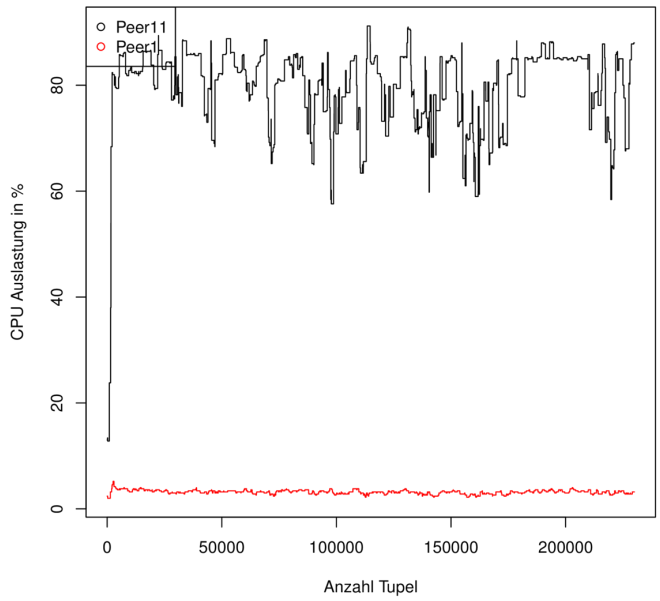


Abbildung 7.16: Prozessorlast Spielerpositionen ohne Load-Balancing

Abbildung 7.17 zeigt, wie sich die Prozessorlast beider Peers mit eingeschaltetem Load-Balancing im Evaluationsverlauf verhält. Der auslösende Schwellwert für das Load-Balancing liegt bei 70% Auslastung und ist als graue Linie eingezeichnet. Zu Beginn steigt die Last auf Peer11 bis auf etwa 80% an, während Peer1 kaum ausgelastet ist. Nach etwa 80.000 Tupeln erkennt man einen deutlichen Abfall der Prozessorlast von Peer11 auf etwa 40%, während die Prozessorlast auf Peer1 auf etwa 20% ansteigt. Die Auslastung bleibt dann auf beiden Peers etwa konstant, allerdings sind zwischendurch sehr kurzfristige Lastspitzen erkennbar. Der starke Abfall der Last auf Peer11 ist durch das Eingreifen des Load-Balancings zu erklären, welches eine Überlast erkannt hat. Ein Teil der Anfrage wird an Peer1 übertragen und beide Peers pendeln sich auf einem ausgeglicheneren Lastniveau ein. Da der Schwellwert für das Load-Balancing danach nicht langfristig überschritten wird, wird kein weiterer Vorgang ausgelöst. Die Lastspitzen, welche vermutlich auf Hintergrundprozesse zurückzuführen sind führen nicht zu einem Load-Balancing, da die Strategie in diesem Moment nicht die Auslastung überprüft.

Latenz

Abbildung 7.18 zeigt den Verlauf der Latenz als Durchschnitt über die fünf Versuche. Nach einem anfänglichen Anstieg bleibt die Latenz bis auf kurzfristige Schwankungen etwa im Bereich von zwi-

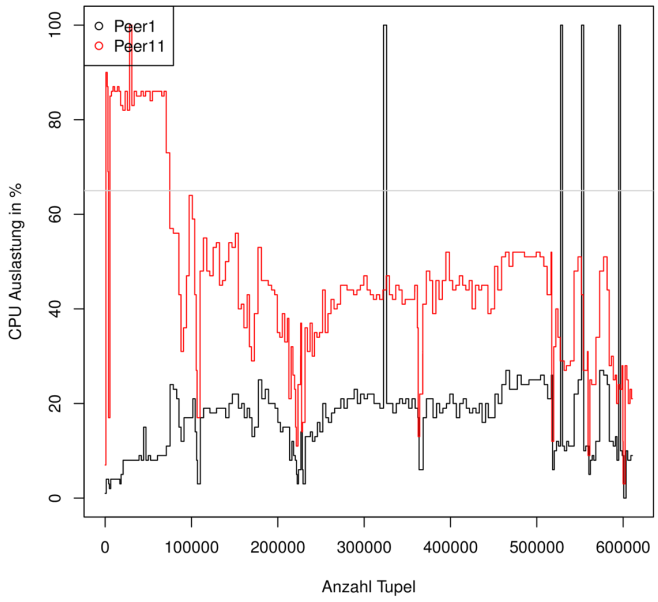


Abbildung 7.17: Prozessorlast Spielerpositionen mit Load-Balancing (Parallel-Track)

schen 1500 und 2000 Millisekunden. Die Schwankungen sind vermutlich darauf zurückzuführen, dass bei der Übertragung im Netzwerk Verzögerungen, z.B. durch verlorene Pakete, aufgetreten sind.

Führt man die gleiche Anfrage mit aktiviertem Load-Balancing und dem Parallel-Track-Kommunikator aus, zeigen sich zwei alternierende Verlaufsmuster der Latenz. Dies ist auf eine unterschiedliche Reihenfolge der installierten Teilanfragen auf Peer11 zurückzuführen. Bei der initialen Verteilung kann es passieren, dass der Peer die empfangenen Anfragen in unterschiedlichen Reihenfolgen installiert. Bei der hier genutzt Load-Balancing Strategie wird allerdings immer die erste installierte Anfrage verschoben. Dadurch kann es zu unterschiedlichen Situationen mit verschiedenen charakteristischen Latenzverläufen kommen. Abbildung 7.19 zeigt die erste Situation: Die Latenz verhält sich analog zum Latenzverlauf ohne Load-Balancing: Die Latenz ist, von gelegentlichen Ausreißern abgesehen, relativ stabil. Das Load-Balancing ist im Verlauf nicht zu erkennen. In dieser Situation wird durch das Verschieben kein zusätzlicher Netzwerksprung erzeugt, weil der verschobene Teil der Anfrage bereits einen Netzwerksprung enthält. Dieser wird durch das Verschieben nur vorgezogen oder nachgelagert.

Im Gegensatz dazu erkennt man in der zweiten Situation, dargestellt durch den Latenzverlauf in Abbildung 7.20 einen deutlichen Anstieg der Latenz zum Zeitpunkt der Neuverteilung von etwa 1.000 auf etwa 10.000 Millisekunden. In dieser Situation wird ein Teil der Anfrage verschoben, der selbst keinen Netzwerksprung enthält. Dadurch entstehen zwei neue Netzwerksprünge (von Peer1 zu Peer11 und zurück). Da Netzwerknutzung deutlich langsamer ist als lokale Verarbeitung führt dies zu dem beobachtbaren Anstieg der Latenz.

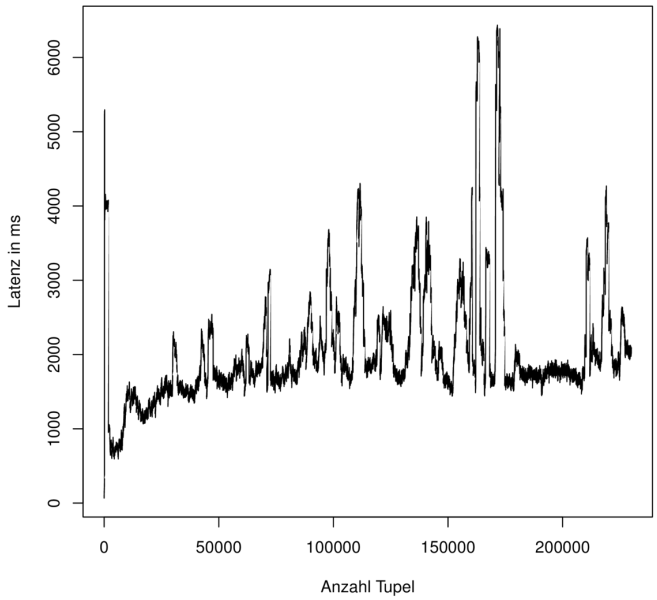


Abbildung 7.18: Latenzverlauf Spielerpositionen ohne Load-Balancing

Durchsatz

Aufgrund der hohen Anzahl von Ausgabewerten wurden die Verlaufskurven für den Durchsatz für diese Anfrage geglättet. Dazu wurden immer Durchschnittswerte aus jeweils 1.000 Tupeln gebildet. Zur besseren Nachvollziehbarkeit sind die originalen Werte in grau hinter den jeweiligen Kurven hinterlegt. Abbildung 7.21 zeigt den Verlauf des Durchsatzes in Abhängigkeit von der Anzahl bereits verarbeiteter Tupel bei der Durchführung der Anfrage ohne aktiviertes Load-Balancing. Man erkennt, dass der Durchsatz relativ gleichmäßig um etwa 1.500 Tupel pro Sekunde schwankt.

Führt man die Anfrage mit aktiviertem Load-Balancing aus, ergibt sich ein Durchsatzverlauf wie in Abbildung 7.22 dargestellt. Man erkennt nach kurzer Zeit einen deutlichen Anstieg des Durchsatzes von etwa 2.000 auf 6.000 Tupel pro Sekunde, der sich im Folgenden bei etwa 5.000 Tupel pro Sekunde einpendelt. Dies ist auf ein gelungenes Load-Balancing zurückzuführen. Durch ein geschickte Neuverteilung gelingt es hier, eine günstigere Verteilung der Teilanfragen zu finden. Dadurch können die Ressourcen besser ausgenutzt werden und mehr Tupel verarbeitet werden. Dies zeigt, dass die Parallel-Track-Strategie grundsätzlich dazu geeignet ist, den Durchsatz einer Anfrage zu erhöhen.

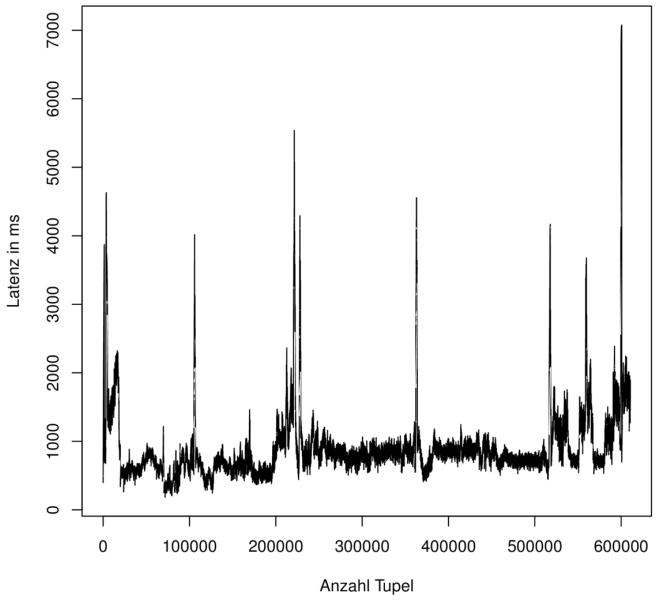


Abbildung 7.19: Latenzverlauf Spielerpositionen mit Load-Balancing (Situation 1)

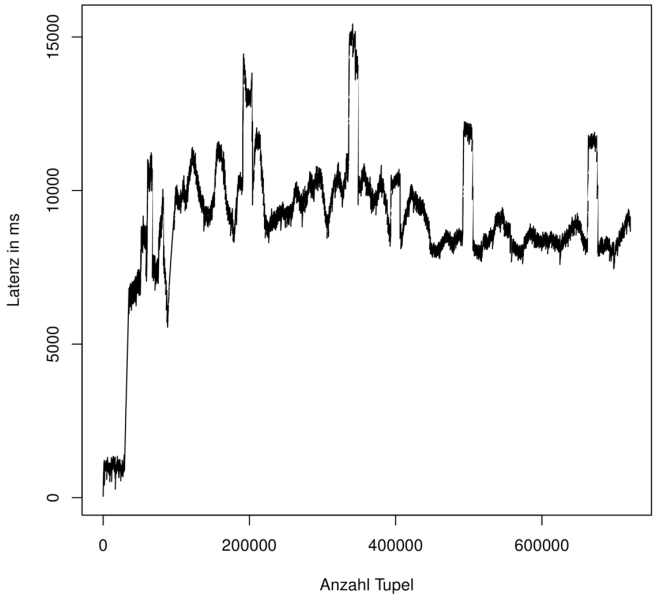


Abbildung 7.20: Latenzverlauf Spielerpositionen mit Load-Balancing (Situation 2)

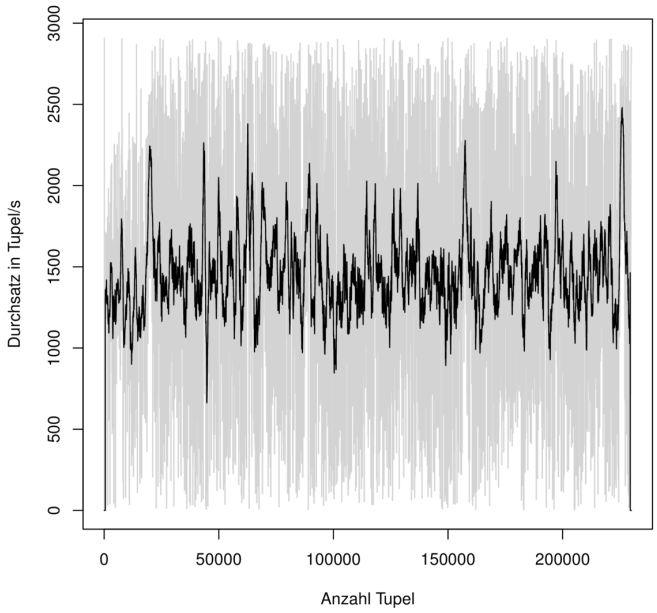


Abbildung 7.21: Durchsatz Ballkontakte ohne Load-Balancing

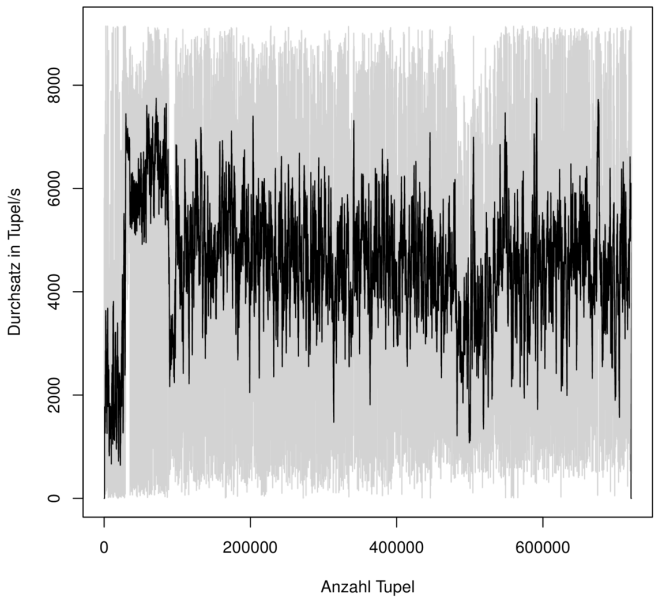


Abbildung 7.22: Durchsatz Ballkontakte mit Load-Balancing (Parallel-Track)

7.3.2.2 Load-Balancing bei zustandsbehafteten Anfragen

In diesem Abschnitt werden die Ergebnisse der Evaluation für das Load-Balancing zustandsbehafteter Anfragen vorgestellt. Auch hier kam es beim Übertragen von Anfragen zum Teil zu einem Verlust von Tupeln. Da ein Tupelverlust bei zustandsbehafteten Anfragen zu veränderten Ausgangswerten führen kann, ist es nicht möglich, die Anzahl der verlorenen Tupel exakt festzustellen. Im Vergleich zur Parallel-Track-Methode ist dieser Verlust allerdings sehr gering. Die Ursache liegt darin, dass in der hier vorliegenden Implementierung das Puffern des Datenstroms zeitgleich mit dem Ersetzen der Sender-Operatoren beginnt. Dadurch können die Sender- und Receiver-Puffer noch Tupel enthalten, die noch im Datenstrom verarbeitet werden, obwohl der Datenstrom eigentlich angehalten ist. Dieser Fehler ließe sich dadurch beheben, dass die Pufferung des Datenstroms deutlich früher, etwa als erster Schritt des Load-Balancing-Vorgangs erfolgt und auf die Leerung der Netzwerkpuffer gewartet wird. Für diese Evaluation wurde als Notbehelf eine feste Wartezeit von einer Sekunde eingebaut, die in den meisten Fällen ausreicht um einen Tupelverlust zu verhinden. Auch hier gilt aber, dass sich Datenströme selbstständig regenerieren und daher ein kleiner Tupelverlust durchaus akzeptabel ist.

Lauf	**1**	**2**	**3**	**4**	**5**
durchschnittl. Prozessorlast Peer1	2,28%	2,23%	2,24%	2,15%	2,05%
durchschnittl. Prozessorlast Peer11	70,48%	70,95%	69,48%	72,80%	69,67%
durchschnittl. Latenz (in ms)	5.474	5.682	5.735	5.713	5.670
durchschnittl. Durchsatz (Tupel/s)	2.650	4.035	3.378	3.418	3.961

Tabelle 7.8: Durchschnittswerte Ballkontakte ohne Load-Balancing

Tabelle 7.8 zeigt die Durchschnittswerte für Latenz, Durchsatz und die Prozessorauslastung beider Peers für alle 5 Referenzläufe der Ballkontakte Anfrage ohne Load-Balancing. Man erkennt auch hier analog zur zustandslosen Anfrage einen deutlichen Unterschied in der Auslastung der beiden Peers: Peer 11 hat in allen Läufen eine durchschnittliche Auslastung von etwa 70%, während Peer1 in allen Läufen bei etwa 2% liegt. Hier ist folglich ein großes Optimierungspotential vorhanden. Die durchschnittliche Latenz liegt in allen Läufen etwa bei 5.500 Millisekunden und der durchschnittliche Durchsatz zwischen 2.650 und 4.035 Tupeln pro Sekunde.

Lauf	**1**	**2**	**3**	**4**	**5**
durchschnittl. Prozessorlast Peer1	1,72%	2,47%	6,57%	11,34%	11,63%
durchschnittl. Prozessorlast Peer11	69,66%	69,33%	61,18%	34,19%	34,20%
durchschnittl. Latenz (in ms)	140.087	108.547	83.779	8.590	8.615
durchschnittl. Durchsatz (Tupel/s)	5.163	2.797	3.576	9.146	10.440
Load-Balancing erfolgreich	nein	nein	nein	ja	ja

Tabelle 7.9: Durchschnittswerte Ballkontakte mit Load-Balancing

Tabelle 7.9 zeigt die Durchschnittswerte für Latenz, Durchsatz und die Prozessorauslastung beider Peers für die Läufe der Ballkontakte Anfrage mit aktiviertem Load-Balancing. Bei dieser Evaluation sind, wie später erläutert wird, einige Läufe fehlgeschlagen. Deshalb ist zusätzlich vermerkt, ob

das Load-Balancing erfolgreich durchgeführt wurde oder nicht. Man erkennt, dass sich bei den fehlgeschlagenen Läufen 1-3 die Prozessorauslastung nicht nennenswert verändert. Außerdem steigt die Latenz im Vergleich stark an und liegt zwischen 83.779 und 140.087 Millisekunden. Der Durchsatz liegt bei den fehlgeschlagenen Versuchen zwischen 2.797 und 5.163 Tupeln pro Sekunde. Bei den erfolgreichen Load-Balancing Vorgängen 4 und 5 sieht man sehr deutlich, dass die Auslastung von Peer11 auf etwa 34% fällt, während die Auslastung von Peer1 auf etwa 11% ansteigt. Außerdem erhöht sich die durchschnittliche Latenz bei diesen Vorgängen leicht, auf etwa 8.500 Millisekunden. Dies ist vermutlich auf das Puffern des Datenstroms bei der Moving-State-Methode zurückzuführen. Der durchschnittliche Durchsatz steigt durch das erfolgreiche Load-Balancing auf etwa 9.000-10.000 Tupel pro Sekunde an. Um genauere Erkenntnisse über die Auswirkungen des Load-Balancings auf diese Einflussgrößen zu haben, werden die Prozessorlast, die Latenz und der Durchsatz im Folgenden noch näher untersucht.

Prozessorlast

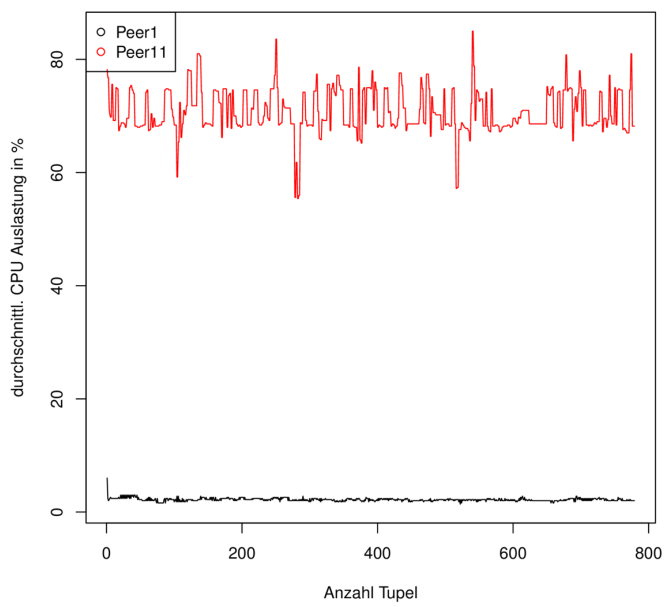


Abbildung 7.23: Prozessorlast Ballkontakte ohne Load-Balancing

Abbildung 7.23 zeigt den Verlauf der Prozessorlast auf den beiden Peers ohne aktiviertes Load-Balancing. Peer1 hat nur eine sehr geringe konstante Auslastung (etwa 2-3%), während Peer11 durchgehend eine Prozessorauslastung von etwa 70-75% aufweist. Dies ist vor allem auf die sehr ungünstige initiale Verteilung zurückzuführen.

Aktiviert man das Load-Balancing und führt die Anfrage erneut aus, ergibt sich (im erfolgreichen Fall) ein Lastverlauf wie in Abbildung 7.24 gezeigt. Als Auslösewert für die Strategie wurde 65% Prozessorlast festgelegt. Dies ist im Bild durch eine graue Linie gekennzeichnet. Peer11 hat zu Beginn eine Auslastung von etwa 70%, Peer1 dagegen nur von etwa 2%. Dies entspricht der Lastverteilung

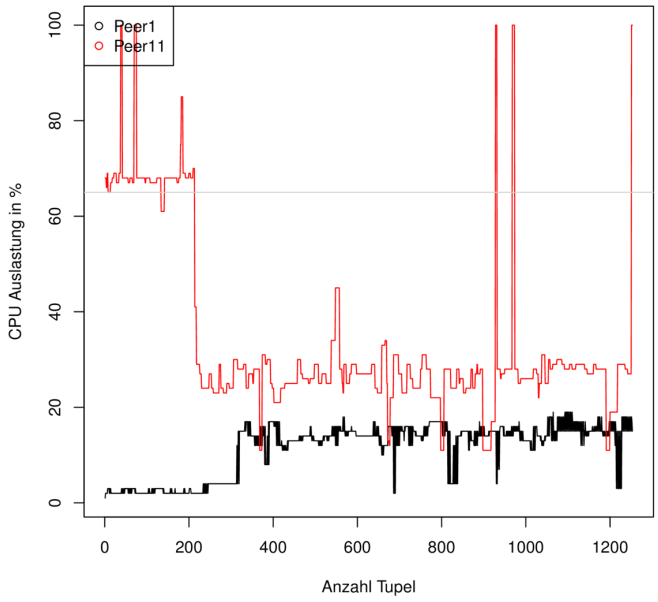


Abbildung 7.24: Prozessorlast Ballkontakte mit Load-Balancing (Moving-State)

ohne aktiviertes Load-Balancing. Nach ca. 200 Tupeln erkennt man, wie die Auslastung auf Peer11 rapide auf etwa 30% absinkt, während kurz später die Auslastung von Peer1 auf etwa 20% ansteigt. Es hat also eine Lastumverteilung stattgefunden. Danach wird kein weiterer Load-Balancing Vorgang ausgelöst. Die kurzen Lastspitzen auf Peer11 sind vermutlich durch Hintergrundprozesse verursacht. Sie lösen kein Load-Balancing aus, da sie nur kurzfristig auftreten und daher von der Strategie nicht erkannt werden.

Latenz

Führt man die Anfrage ohne aktiviertes Load-Balancing aus, ergibt sich der in Abbildung 7.25 dargestellte Latenzverlauf. Man erkennt, dass die Latenz stark schwankt und im Mittel bei etwa um 5.000 Millisekunden liegt. Die Schwankungen sind vermutlich auf den Assure-Heartbeat-Operator zurückzuführen. Dieser Operator sendet gegebenenfalls alte Werte erneut heraus um den Datenstrom nicht zu unterbrechen, dabei steigt die Latenz. Dies wird in Abschnitt 7.2 näher erläutert.

Bei aktiviertem Load-Balancing zeigen sich zwei unterschiedliche Verläufe: Abbildung 7.26 zeigt einen gelungenen Load-Balancing Versuch. Zunächst verhält sich die Latenz analog zum Verlauf ohne Load-Balancing: Der Wert schwankt um etwa 5.000 Millisekunden. Nach etwa 200 Tupeln erkennt man einen starker Anstieg der Latenz bis auf etwa 50.000 Millisekunden, bevor die Latenz nach etwa 300 Tupeln wieder auf etwa 5.000 Millisekunden abfällt. Danach bleibt der Lastverlauf bis auf einen Ausreißer nach etwa 800 Tupel unauffällig. Die Latenzspitze nach 200 Tupeln lässt sich auf das Load-Balancing zurückführen: Während der Load-Balancing Vorgang läuft wird der Datenstrom gepuffert. Dies führt zu steigenden Latenzen, da die Tupel immer länger im System verbleiben. Wird das Puffern beendet läuft der Datenstrom weiter und die Latenz fällt wieder auf das ursprüngliche Niveau zurück.

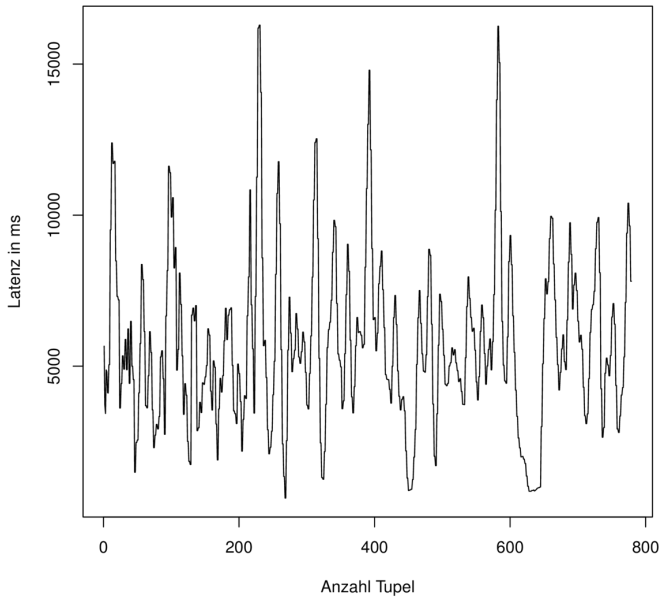


Abbildung 7.25: Latenzverlauf Ballkontakte ohne Load-Balancing

Abbildung 7.27 zeigt den Lastverlauf eines fehlgeschlagenen Load-Balancing Versuchs: Nach etwa 200 Tupeln erkennt man einen starken, sehr gleichmäßigen Anstieg der Latenz, der sich bis zum Ende der Evaluation unbegrenzt fortsetzt. Nähere Untersuchungen haben gezeigt, dass dieses Verhalten auf eine Kombination von zwei Faktoren zurückzuführen ist: Durch einen Timing-Fehler im Load-Balancing wird das Puffern des Datenstroms nicht beendet und dadurch empfangen hintere Teile der Anfrage keine Tupel mehr. Enthalten diese Teile einen Assure-Heartbeat-Operator kommt es zu dem hier beobachtbaren Verlauf. Der Assure-Heartbeat-Operator sendet unabhängig vom Rest des Datenstroms in regelmäßigen Abständen Punctuations. Diese werden von einer nachgelagerten Aggregation empfangen. Die Aggregation gibt daraufhin erneut den aktuellen Wert aus, allerdings erhöht sich dabei jeweils die Latenz des Ausgabetupels um die Zeitspanne des Heartbeats. Um dieses Problem zu beheben müsste man zunächst den Timing-Fehler beheben. Außerdem sollten vom Datenstrom unabhängige Operatoren, wie etwa der Assure-Heartbeat-Operator ebenfalls angehalten werden, wenn der Datenstrom gepuffert wird. Dies könnte durch eine zusätzliche Nachricht an die entsprechenden Peers bewerkstelligt werden.

Durchsatz

Der Durchsatz für die Durchführung der Anfrage ohne Load-Balancing ist in Abbildung 7.28 in Abhängigkeit zur Anzahl der verarbeiteten Tupel abgebildet. Da diese Anfrage relativ wenig Ausgabetupel produziert wurde hier auf eine Glättung der Werte verzichtet. Man erkennt, dass der Durchsatz relativ stark zwischen etwa 10 und 340 Tupeln pro Sekunde schwankt und im Mittel bei etwa 150-200 Tupeln pro Sekunde liegt.

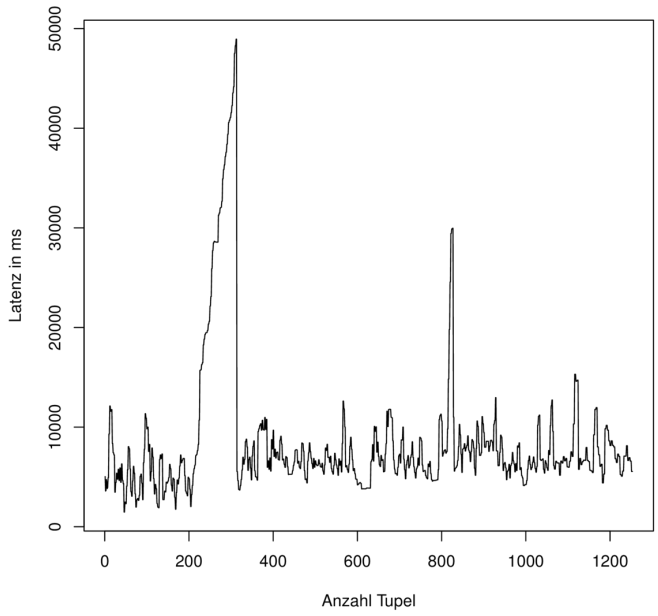


Abbildung 7.26: Latenzverlauf für Ballkontakte mit gelungenem Load-Balancing (Moving-State)

Abbildung 7.29 zeigt den Durchsatzverlauf bei gelungenem Load-Balancing. Anfänglich schwankt der Wert etwa um 100 Tupel pro Sekunde. Nach etwa 200 Tupeln erkennt man einen deutlichen Anstieg des Durchsatzes auf etwa 280 Tupel pro Sekunde. Dieser Anstieg ist darauf zurückzuführen, dass durch das gelungene Load-Balancing nach ca. 200 Tupeln eine günstigere Lastverteilung als vorher zwischen den beiden Peers besteht, so dass mehr Tupel verarbeitet werden können und sich der Durchsatz erhöht. Dies zeigt, dass durch das Load-Balancing bei geschickter Umverteilung durchaus Geschwindigkeitsvorteile realisiert werden können.

Der Verlauf des Durchsatzes mit fehlgeschlagenem Load-Balancing, abgebildet in 7.30, bestätigt den Verdacht, dass die ausgegebenen Werte durch den Assure-Heartbeat-Operator verursacht werden: Zunächst verhält sich der Durchsatz ähnlich zum Verlauf ohne Load-Balancing. Nach etwa 200 Tupeln entsteht allerdings ein oszillierender Verlauf, bei dem der Durchsatz zwischen zwei Werten schwankt. Dies ist darauf zurückzuführen, dass, wie oben beschrieben, nur noch Werte durch den Assure-Heartbeat-Operator produziert werden. Da es sich um eine Team-Anfrage handelt, gibt diese jeweils immer zwei Werte aus, deren Durchsatz sich nie ändert. Dadurch treten nach dem fehlgeschlagenen Load-Balancing nur noch zwei verschiedene Durchsatz-Werte auf, die zu dem beobachtbaren oszillierenden Verlauf führen.

7.3.2.3 Auswertung

Durch die Evaluation des Load-Balancings wurde zunächst einmal gezeigt, dass das vorgestellte Load-Balancing-Konzept grundsätzlich funktioniert. Auch wenn es noch einige Punkte gibt, in denen

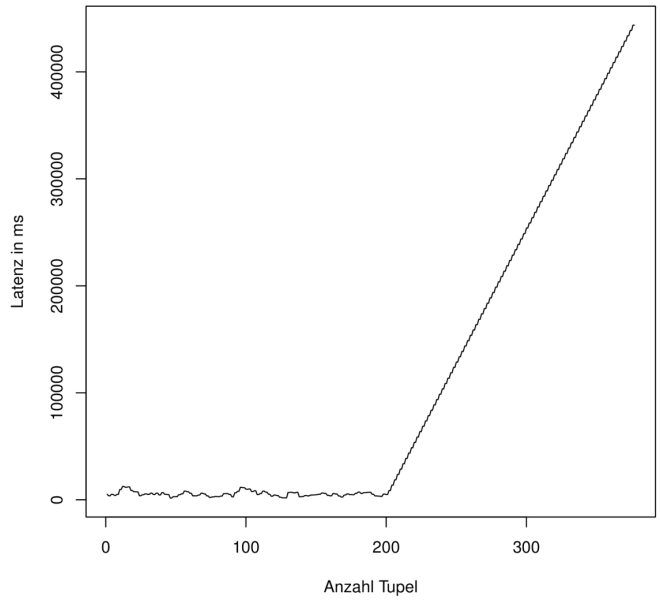


Abbildung 7.27: Latenzverlauf für Ballkontakte mit fehlgeschlagenem Load-Balancing (Moving-State)

die Implementierung noch verbessert werden müsste, sind sowohl Parallel-Track-Kommunikator, als auch Moving-State-Kommunikator in der Lage, laufende Anfragen von einem Peer zum anderen zu verschieben.

Dabei hat sich gezeigt, dass eine simple Load-Balancing Strategie und eine Round-Robin Allokation nicht ausreichen um wirklich ein optimales Ergebnis zu erzielen. Der Performancegewinn durch das Load-Balancing ist davon abhängig, wie intelligent Teilanfragen verschoben werden. Wird der richtige Teil an den richtigen Peer übertragen kann dies einen enormen Performancegewinn bedeuten, bei einer ungünstigen Umverteilung sind aber sogar Performanceverluste möglich. Eine wichtige Aufgabe für zukünftige Arbeiten ist daher das Entwickeln intelligenter Strategien und Allokatoren um wirklich alle Optimierungspotentiale optimal auszunutzen.

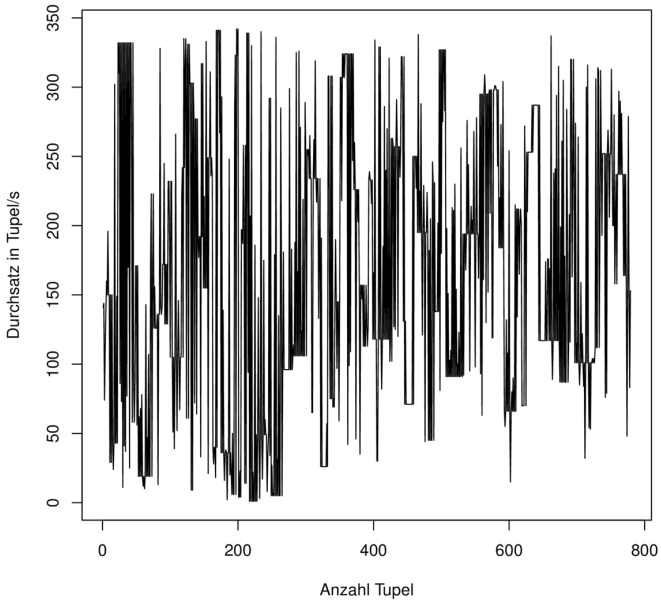


Abbildung 7.28: Durchsatz für Ballkontakte ohne Load-Balancing

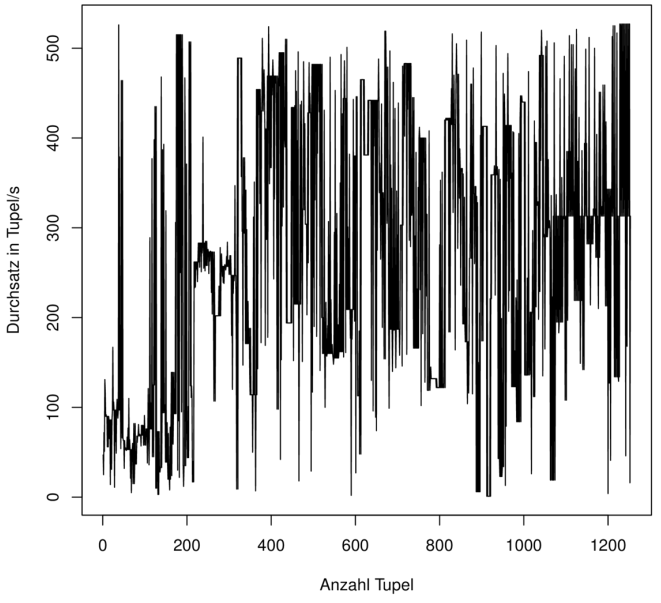


Abbildung 7.29: Durchsatz für Ballkontakte mit gelungenem Load-Balancing (Moving-State)

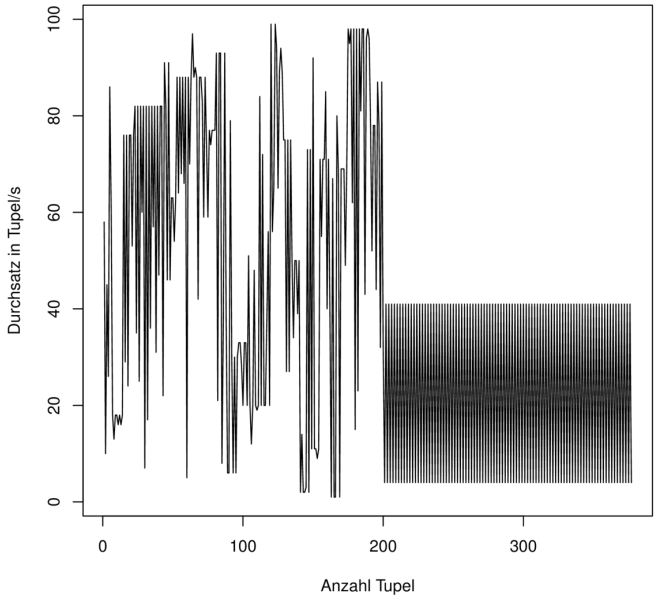


Abbildung 7.30: Durchsatz für Ballkontakte mit fehlgeschlagenem Load-Balancing (Moving-State)

7.4 Recovery

Durch die Evaluation des Recoverys soll die Funktionsfähigkeit und die Güte dessen überprüft und festgehalten werden. Um das Recovery zu evaluieren, gibt es im Wesentlichen vier Parameter, die angepasst werden können. Das sind die verwendete Quelle, die berechneten Anfragen, die eingestellte Strategie und die Größe des Netzwerks. Um die Anzahl der Evaluationsszenarien im überschaubaren und durchführbaren Rahmen zu halten, werden nicht alle möglichen Kombinationen evaluiert. Stattdessen konzentriert sich die Evaluation auf eine kleine Teilmenge dieser Szenarien.

7.4.1 Aufbau

Im Folgenden wird beschrieben, wie die Testszenarien aufgebaut wurden, um eine einheitliche Messgrundlage zu schaffen.

7.4.1.1 Quelle

Als Datengrundlage wird die DEBS-Quelle *soccergame_ 10_ 20.csv* verwendet. Dies ist eine Variante der originalen DEBS-Daten, bei der die Anzahl der Positionsdaten verringert wurde. Zu der Datenquelle wird das passende DDC genutzt und über die DEBS-Quellendefinition eingebunden. Da die Berechnung dieser Quelle zur Evaluation relativ lange braucht, wird diese nur für die Analysezeit-Anfrage genutzt, bei der die fehlenden Tupel manuell gezählt werden können und die Daten deshalb nicht bis zum Ende berechnet werden müssen. Für die Pässe-Anfrage wird der deutlich verkleinerte DEBS-Datenstrom *soccergame_ recovery.csv* genutzt. Die Berechnung dieser benötigt deutlich weniger Zeit. Daher eignet sie sich für eine vollständige Berechnung.

7.4.1.2 Verwendetes Produkt

Als Produkt wird das LSOP-Produkt mit eingeschaltetem Autoimport und Autoexport genutzt. Dies sorgt dafür, dass Quellen, die an einem Peer importiert wurden, auch auf allen anderen Peers vorhanden sind. Das ist notwendig, um es zu ermöglichen, dass auch ein Peer mit einer Quelle recovert werden kann.

7.4.1.3 Anfragen

Als Testanfrage werden die Analysezeit- und die Pässe-Anfrage genutzt. Die Analysezeit eignet sich, da sie in genauen Abständen eine leicht zu überblickende Anzahl an Tupeln ausgibt, die per Hand auf Gültigkeit (im Hinblick auf Vollständigkeit und Reihenfolge) geprüft werden kann. So muss nicht die gesamte Anfrage bis zum Ende der Quelle berechnet werden, bevor eine Aussage getroffen werden kann.

Die Pässe-Anfrage wird zusätzlich genutzt, da auch eine Anfrage mit einem zustandsbehafteten Operator wie etwa einem *Aggregate* genutzt werden soll. Hier ist es interessant zu beobachten, welche Auswirkungen der Ausfall eines Peers mit zustandsbehafteten Operator auf die Ergebnisse hat.

7.4.1.4 Strategie

Es werden sowohl die Standardstrategie („Simple“) als auch das Active-Standby evaluiert. Die zu nutzende Strategie ist in dem jeweiligen Evaluationsszenario angegeben.

7.4.1.5 Größe des Netzwerks

Das Netzwerk besteht aus einer Anzahl von heterogenen Rechnern. Die Größe des Netzwerks ist in dem jeweiligen Evaluationsszenarien angegeben.

7.4.1.6 Messungen

Für die jeweiligen Evaluationsszenarien müssen unterschiedliche Werte gemessen werden, um eine Aussage aus der Evaluation ziehen zu können. Die möglichen Messungen werden nun beschrieben. Bei den Szenarien wird jeweils angegeben, welche Messungen durchzuführen sind.

Erfolgreiche Durchführung

Das Recovery war erfolgreich, wenn nach dem Recovery die Tupel wieder von der Quelle bis zur Senke fließen und die Anfrage weiterhin sinnvolle Ergebnisse liefert.

Dauer

Dies ist die Dauer, die vom Entdecken des Ausfalls bis zum fertigen Installieren aller Query-Parts auf anderen Peers vergeht. Dies schließt nicht die Zeit mit ein, die zum neu-Verknüpfen der Sender und Receiver (u.a. "GoOn") benötigt wird, da dies von anderen Peers aus geschieht. Die Zeitmessung wird automatisch im RecoveryCommunicator vorgenommen und in den Debug-Nachrichten ausgegeben.

Anzahl verlorene Tupel

Hier werden die Tupel gezählt, die in der Ausgabe der Anfrage fehlen. Wenn dies gemessen wird, muss also eine Anfrage genommen werden, bei der regelmäßig Tupel bis ans Ende gelangen, wie zum Beispiel die Analysezeit-Anfrage. Diese liefert für jede Sekunde im Datenstrom genau ein Tupel. Wenn die Analysezeit-Anfrage genutzt wird, muss diese nicht ganz durchlaufen werden, um zu registrieren, wie viele Tupel am Ende verloren gehen. Es genügt, nach der Fortführung der Bearbeitung nach dem Recovery die Bearbeitung zu beenden und zu zählen, wie viele Sekunden-Tupel nicht ausgegeben worden sind. Wird die Pässe-Anfrage genutzt, muss die Berechnung vollständig durchlaufen. Die Ergebnisse werden anschließend mit einer vorbereiteten Musterlösung verglichen, die auf einem Peer ohne Ausfall berechnet wurde.

Latenz

Die Latenzmessung soll ergeben, ob das Recovery die Latenz beeinflusst. Dazu werden die Latenzwerte aus der CSV-Datei genutzt und die Werte vor, während und nach dem Recovery verglichen.

Durchsatz

Die Durchsatzmessung soll ergeben, ob das Recovery den Durchsatz beeinflusst. Dazu werden die Durchsatzwerte aus der CSV-Datei genutzt und die Werte vor, während und nach dem Recovery verglichen.

7.4.1.7 Szenarien

Hier werden die Szenarien beschrieben, die zur Evaluation durchgeführt wurden. Zu jedem Szenario sind die Ergebnisse angegeben. Eine kurze Einschätzung erläutert die Ergebnisse der Evaluationsszanarien. Für alle Szenarien gilt die folgende Durchführung:

1. Die Odysseus Instanzen werden gestartet
2. Der DEBS-Datenstrom wird eingebunden (an Peer1, die anderen erhalten die Daten automatisch)
3. Die jeweilige Anfrage wird von Peer1 aus installiert
4. Der jeweilige Peer wird geschlossen
5. Die gemessenen Werte werden dokumentiert

7.4.2 Ergebnisse

In den folgenden Untersektionen werden die Ergebnisse der Durchführung der in Tabelle 7.10 beschriebenen Szenarien ausgewertet. Jedes Szenario wurde hierbei fünfmal durchgeführt und die Ergebnisse nach den oben genannten Kriterien für die Messung aufgezeichnet.

7.4.2.1 Szenario 1: Analysezeit (Sink)

Beim ersten Szenario wird das Verhalten von Herakles auf den Ausfall des *Sink*-Operators getestet. Hierzu wird die Analysezeit-Anfrage mit der Partitionierungsstrategie *Operatorsetcloud* in einem Netzwerk mit vier Peers verteilt. Der initial verteilte Anfrageplan von Durchlauf 2 (D2) ist auf der linken Seite in Abbildung 7.31 dargestellt und das Skript für die Anfrage ist unter Listing A.6 im Anhang zu finden. Da die Anfrage simpel aufgebaut ist, wurde sie durch die Partitionierungsstrategie in lediglich zwei Query-Parts aufgeteilt. Somit wurde die Anfrage nur auf zwei Peers verteilt. Der *Sink*-Operator ist der letzte Operator im Anfrageplan und bildet die Schnittstelle zur Coach-Applikation. Es ist zu erwarten, dass zwischen dem tatsächlichen Peerausfall und dem Erkennen des Ausfalls eine begrenzte Zahl an Tupeln verloren geht. Nachdem der Ausfall bekannt ist und die *HoldOnMessage* versendet wurde (beschrieben im Implementierungskapitel 6.7), sollten die Tupel im vorgelagerten Sender gepuffert werden. Nach dem Recovery wird der Buffern wieder geleert, die zwischengespeicherten Tupel also weitergesendet.

Tabelle 7.11 zeigt die Auswertung dieser Testreihe. Wie erwartet sind bei dem Ausfall nur sehr wenige Tupel verloren gegangen. Im Durchschnitt fehlte in der Ausgabe nur ein Tupel. Die Durchläufe D1 und D3 können trotzdem nicht als erfolgreich bewertet werden, da nach dem Recovery Ergebnisse doppelt ausgegeben wurden. Bei D1 und D3 wurde das Recovery gleichzeitig von zwei Peers

ID	Anzahl Rechner	Installierte Anfragen	Genutzte Quelle	Ausfallender Peer	Genutzte Strategie	Gemessene Werte
1	4	Analysezeit	soccergame_10_20.csv	Peer, auf dem die FileSink ist	Simple	Erfolgreiche Durchführung, Dauer, Anzahl der verloren gegangenen Tupel, Latenz, Durchsatz
2	4	Analysezeit	soccergame_10_20.csv	Peer, auf dem die FileSink nicht ist	Simple	Erfolgreiche Durchführung, Dauer, Anzahl der verloren gegangenen Tupel, Latenz, Durchsatz
3	4	Analysezeit	soccergame_10_20.csv	Peer, auf dem der RecoveryMerge ist	Active-Standby	Erfolgreiche Durchführung, Dauer, Anzahl der verloren gegangenen Tupel, Latenz, Durchsatz
4	4	Analysezeit	soccergame_10_20.csv	Peer, auf dem der Recovery-Merge nicht ist	Active-Standby	Erfolgreiche Durchführung, Dauer, Anzahl der verloren gegangenen Tupel, Latenz, Durchsatz
5	4	Pässe	soccergame_recovery.csv	Peer, auf dem der letzte Aggregate-Operator ist	Simple	Erfolgreiche Durchführung, Dauer, Anzahl der verloren gegangenen Tupel, Latenz, Durchsatz
6	4	Pässe	soccergame_recovery.csv	Peer, auf dem der letzte Aggregate-Operator ist	Active-Standby	Erfolgreiche Durchführung, Dauer, Anzahl der verloren gegangenen Tupel, Latenz, Durchsatz

Tabelle 7.10: Überblick über die Szenarien

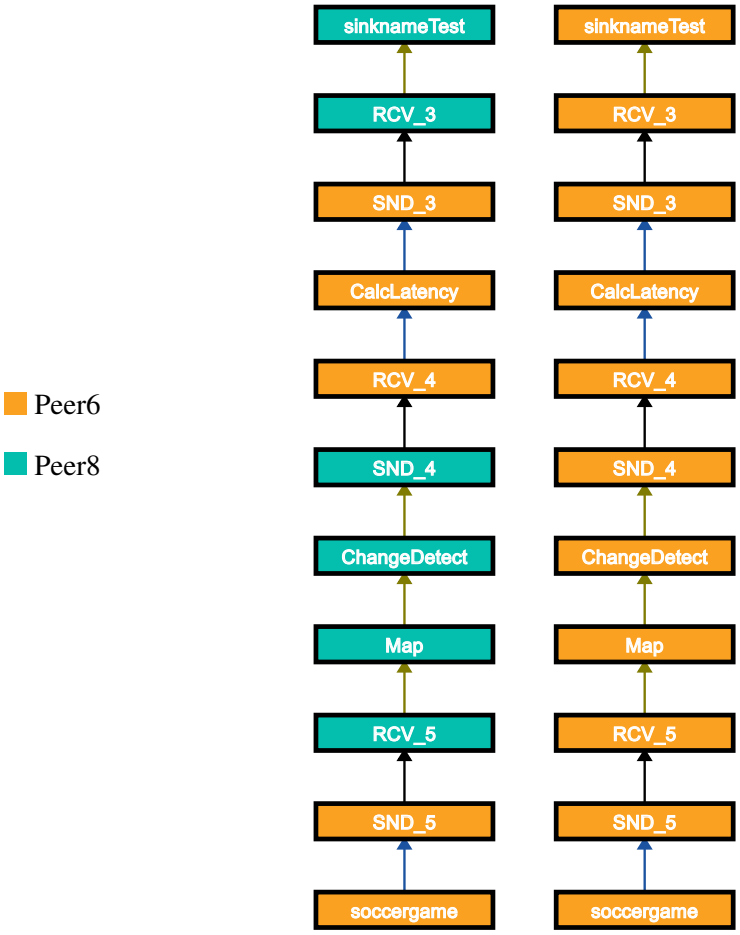


Abbildung 7.31: Analysezeit(Sink)-Anfrageplan vor dem Ausfall (links) und nach dem Recovery (rechts)

durchgeführt, sodass die ausgefallenen Query-Parts doppelt wiederhergestellt wurden. Grund dafür ist, dass in der Verhandlungsphase eine zu geringe Zeit auf eine Antwort der anderen Peers gewartet wird, ob diese bereits ein Recovery durchführen. Die genutzte Zeit von drei Sekunden genügte in dieser Netzwerkkonfiguration nicht immer. Bei D3 kam es zusätzlich zu einem Fehler während des Recovery-Prozesses, sodass dieser erst nach drei Minuten abgeschlossen werden konnte. Dieser zeigte jedoch, dass das Recovery auch unter Umständen, die für P2P-Netzwerke nicht unüblich sind, funktionstüchtig ist. Die Kommunikation zwischen zwei Peer brach während des Recoverys ab, sodass der ursprünglich ausgewählte Peer zur Installation des ausgefallenen Anfrageteils nicht mehr erreichbar war. Das Warten auf Antworten von diesem Peer hat entsprechend lange gedauert. Positiv ist hierbei jedoch, dass die Wiederherstellung trotzdem auf einem anderen Peer funktioniert hat. Bei den anderen vier Durchläufen konnte der Recovery-Prozess nach spätestens fünf Sekunden abgeschlossen werden.

Vergleicht man die Latenzen und Durchsätze vor dem Ausfall, während der Pufferungsphase und nachdem der Puffer abgebaut wurde, sieht man hier deutliche Unterschiede. Vor dem Ausfall liegen die Latenzwerte alle um 150ms und der Durchsatz ist sehr gering. Während der Pufferphase sind beide Werte deutlich erhöht. Nachdem der Puffer abgebaut wurde ist die Latenz noch deutlich unter den

Analysezeit (Sink)		D1	D2	D3	D4	D5	Ø
Ø-Latenz (in ms)	vorher	195,6	86,1	96,7	135,6	297,7	162,3
	puffernd	29034,9	20791,7	236083,3	23886,4	19386,6	65836,6
	nachher	61,7	0,7	1,0	0,6	0,8	13,0
Ø-Durchsatz (in Tupel/s)	vorher	2034,31	2213,98	1910,88	846,57	2275,60	1856,27
	puffernd	14486,01	21495,48	28575,03	19745,01	20035,42	20867,39
	nachher	5145,56	5971,77	5294,17	6243,06	6085,35	5747,98
Erfolgreich		Nein	Ja	Nein	Ja	Ja	3 von 5
Dauer (in ms)		4558	4539	174457	4057	4793	38480,8
Ausfallzeitpunkt (Tupel)		182	116	156	40	159	130,6
Ende Pufferung (Tupel)		1914	887	6296	899	883	2175,8
Verarbeitete Tupel		4323	3030	6734	3028	2098	3842,6
Verlorene Tupel		2	1	1	0	1	1

Tabelle 7.11: Szenario1: Analysezeit (Sink)

Ausgangszustand gesunken und der Durchsatz ist deutlich größer. Dies liegt daran, dass die Query-Parts des ausgefallenen Peers bei dem Peer wiederhergestellt wurden, der vorher schon den zweiten Teil der Anfrage ausgeführt hat (vgl. Abbildung 7.31). Somit wird die Anfrage nun lokal ausgeführt, was die deutlich niedrigere Latenz erklärt (vgl. Abschnitt 7.2.2.1). Bei den beiden fehlgeschlagenen Durchgängen sind die Latenzwerte nachher etwas erhöht, weil durch die doppelte Durchführung des Recoverys die Last erhöht ist, da einige Query-Parts doppelt bearbeitet werden müssen.

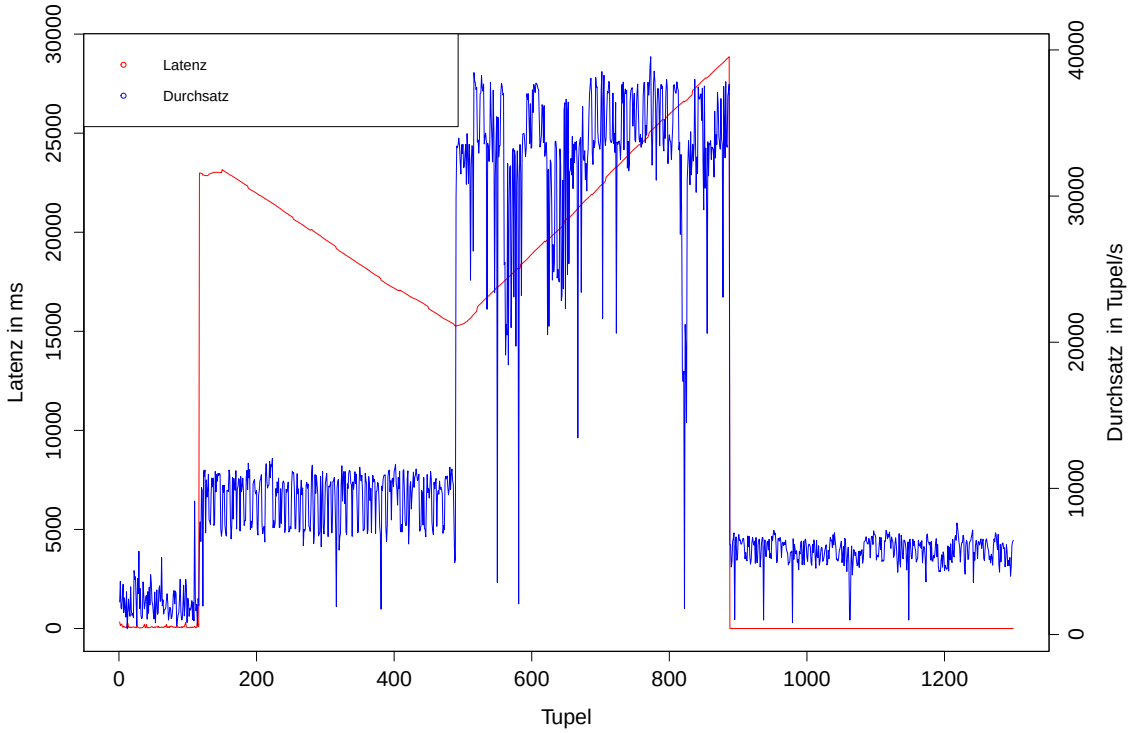


Abbildung 7.32: Durchsatz- und Latenzverlauf bei D2 des Szenario 1 der Recovery Evaluation

Abbildung 7.32 zeigt den Verlauf der Latenz und des Durchsatzes in Durchlauf 2. Der Peerausfall tritt bei diesem Durchlauf bei Tupel 116 auf. Bis zu diesem Zeitpunkt wird die Anfrage ganz normal mit der initialen Verteilung im Netzwerk ausgeführt. Sowohl die Latenz als auch der Durchsatz sind vergleichsweise gering.

Nach Bekanntwerden des Ausfalls wird im Netzwerk eine *HoldOnMessage* versendet. Ab diesem Zeitpunkt werden die Tupel beim Sender vor dem ausgefallenen Peer gepuffert, sodass die Latenz schlagartig ansteigt. Während des Pufferaufbaus gibt es zwei Phasen. In der ersten ist der Durchsatz leicht erhöht. Er liegt in diesem Beispiel durchschnittlich bei etwa 10000 Tupel pro Sekunde. In der zweiten Phase ist der Durchsatz deutlich höher. Er steigt auf bis zu 40000 Tupel pro Sekunde. Der Grund für diesen plötzlichen Anstieg ist nicht bekannt. Eventuell wird der Durchsatz erhöht, da das Puffern der Daten schneller ist als Weitersenden. Dies ist allerdings nur eine wage Vermutung.

Nach Abschluss des Recovery-Prozesses wird eine *GoOnMessage* versendet und es beginnt die Phase des Pufferabbaus. In diesem Beispiel ist dies ab Tupel 887 der Fall (vgl. Tabelle 7.11). Im Puffer befinden sich nun alle Tupel zwischen 116 und 887. Während der Puffer geleert wird, kommen keine neuen Tupel mehr in den Puffer. Die Latenz zeigt in dieser Phase einen v-förmigen Verlauf. Zunächst sinkt die Latenz mit jedem weiteren Tupel, was bedeutet, dass die Tupel schneller aus dem Puffer abgearbeitet werden als sie hineingekommen sind. Da nach dem Recovery die Anfrage lokal ausgeführt wird (vgl. Abbildung 7.31), ist dies auch trotz des etwas erhöhten Durchsatzes möglich.

Ab einem gewissen Zeitpunkt steigt die Latenz jedoch wieder an. Dies ist auf die plötzliche Erhöhung des Durchsatzes zurückzuführen. Die Daten kamen zu dieser Zeit schneller in den Puffer hinein als sie nachher, auch trotz lokaler Ausführung, weiterverarbeitet werden konnten. Jedes weitere Tupel verblieb somit länger im Puffer und die Latenz steigt. Nach der Leerung des Puffers (ab Tupel 888) wird die Anfrage wieder normal ohne Pufferung ausgeführt. Da die Anfrage nun lokal läuft, ist der Durchsatz etwas höher und die Latenz niedriger als vor dem Ausfall. Dieses Verhalten der Analysezeit-Anfrage wurde auch in Abschnitt 7.2.2.1 festgestellt.

Bei einer Nutzung von Sensoren als Datenquelle sollte dieses Phänomen während der Pufferphase nicht auftreten, da diese mit einer konstanten Datenrate senden. Problematisch ist allerdings die Phase des Pufferabbaus. Während dieser Zeit staut sich der Datenstrom, da keine Tupel in den Puffer fließen können. Es wird in dieser Zeit wohl zu weiteren Tupelverlusten kommen, da die Datenrate im Gegensatz zur Verwendung einer CSV-Datei nicht verringert werden kann.

7.4.2.2 Szenario 2: Analysezeit (Mitte)

Analysezeit (Mitte)		**D1**	**D2**	**D3**	**D4**	**D5**	**Ø**
Ø-Latenz (in ms)	vorher	1542,6	820,9	304,8	493,1	937,9	819,9
	puffernd	118920,2	71553,5	97914,6	94085,2	53536,0	87201,9
	nachher	-	-	-	-	-	-
Ø-Durchsatz (in Tupel/s)	vorher	900,28	704,32	1291,71	594,68	1468,00	991,80
	puffernd	9765,69	35189,32	19725,51	17666,30	25479,88	21565,34
	nachher	-	-	-	-	-	-
Erfolgreich		Ja	Ja	Ja	Ja	Ja	Ja
Dauer (in ms)		5359	4065	4193	9659	3768	5408,80
Ausfallzeitpunkt (Tupel)		203	60	42	22	32	71,80
Ende Pufferung (Tupel)		-	-	-	-	-	-
Verarbeitete Tupel		506	336	477	321	442	416,4
Verlorene Tupel		0	8	0	0	0	1,60

Tabelle 7.12: Szenario 2: Analysezeit (Mitte)

Beim zweiten Szenario wird der Ausfall eines Peers simuliert, welcher weder den ersten Teil (Quelle) noch den letzten Teil (Senke) der Anfrage besitzt. Hierfür wurde *Operatorcloud* als Partitionierungsstrategie verwendet, da bei einer Verteilung mit *Operatorsetcloud*, wie sie im ersten Szenario verwendet wurde, kein Peer die nötigen Voraussetzungen erfüllt hätte. Eine beispielhafte initiale Verteilung der Anfrage ist in Abbildung A.3 im Anhang zu sehen und das Skript für die Anfrage ist unter Listing A.7 im Anhang zu finden. Simuliert wird somit der Ausfall des Peers mit dem *Map*-Operator (D1, D2, D3, D5) oder dem *ChangeDetect*-Operator (D4). Zu erwarten ist wiederum ein Verlust von Tupeln zwischen dem tatsächlichen Ausfallzeitpunkt und dem bemerken des Ausfalls (*HoldOnMessage*).

Die Ergebnisse der fünf Durchgänge sind in Tabelle 7.12 dargestellt. Bei allen Durchläufen ist das Recovery erfolgreich gewesen. Die benötigte Zeit für das Recovery liegt mit Ausnahme von D4 bei vier bis fünf Sekunden. Der vierte Durchgang benötigte etwa zehn Sekunden. Beim vierten Durchlauf ist ein unbekannter Fehler während des Prozesses aufgetreten, sodass das Recovery erst beim zweiten Anlauf erfolgreich abgeschlossen werden konnte. Hervorzuheben ist, dass das Recovery trotz dieses Fehlers erfolgreich durchgelaufen ist, indem ein anderer Peer zur Installation des ausgefallenen Anfrageteils ausgewählt wurde. Dies zeigt die Robustheit der Implementierung.

Entgegen der Erwartung sind in vier von fünf Durchgängen keine Tupel verloren gegangen. Lediglich bei D2 fehlten acht Tupel in der Ausgabe. Die Anzahl der verlorenen Tupel hängt immer davon ab, wie schnell der Vorgänger des ausgefallenen Peers mitbekommt, dass dieser ausgefallen ist und er somit seine Ergebnisse puffern muss. Dies bei D2 etwas länger gedauert.

Leider wurden bei diesem Szenario die Anfragen nach dem Recovery zu früh beendet, sodass bei keinem Durchlauf die Pufferphase abgeschlossen wurde und somit keine Latenz- und Durchsatz-Werte für die Phase danach zur Verfügung stehen. Vergleicht man die Latenz und Durchsatz-Werte vor dem Ausfall mit dem vorangegangenen Szenario, sieht man, dass diese im Schnitt etwas schlechter ausfallen. Dies liegt an der geänderten Partitionierungsstrategie, wodurch der *Map*- und *Change-Detect*-Operator nun auf verschiedene Query-Parts partitioniert und diese dann auf unterschiedliche Peers verteilt werden. Somit ist ein weiterer Netzwerksprung zwischen diesen Operatoren nötig. Die hohen Latenz- und Durchsatz-Werte während der Pufferungsphase erklären sich äquivalent zum ersten Szenario. Die Gründe für die teilweise großen Schwankungen zwischen den Durchläufen, vor allem beim Durchsatz, sind unbekannt. Es kann unter anderem an dem Netzwerk liegen, welches häufig einen Einfluss unbekannter Größe darstellt.

7.4.2.3 Szenario 3: Analysezeit (RecoveryMerge)

AnalysezeitAS (RMerge)		**D1**	**D2**	**D3**	**D4**	**D5**	**Ø**
Ø-Latenz (in ms)	vorher	607,3	1076,3	155,4	193,0	119,3	430,3
	puffernd	-	-	-	-	-	-
	nachher	20797,6	797,7	157,5	229,2	115,0	4419,4
Ø-Durchsatz (in Tupel/s)	vorher	475,99	651,20	1019,18	470,19	1074,80	738,27
	puffernd	-	-	-	-	-	-
	nachher	263,62	194,90	848,87	534,70	1183,47	605,11
Erfolgreich		Nein	Nein	Ja	Ja	Ja	3 von 5
Dauer (in ms)		135017,5	9880	64518	65118	-	68633,38
Ausfallzeitpunkt (Tupel)		27	27	45	32	44	35
Ende Pufferung (Tupel)		-	-	-	-	-	-
Verarbeitete Tupel		231	76	229	230	404	234
Verlorene Tupel		11	1	1	50	1	12,8

Tabelle 7.13: Szenario3: AnalysezeitAS (RecoveryMerge)

In diesem Szenario wurde getestet, wie das Verhalten des Active-Standbys beeinflusst wird, wenn der Peer mit dem Operator *RecoveryMerge* ausfällt. Dieser ist dafür verantwortlich, die Ergebnisse zweier parallel verarbeiteten, identischen Query-Parts zu koordinieren. Als Partitionierungsstrategie wurde *Operatorsetcloud* verwendet. Der initiale Zustand kann unter Abbildung A.5 betrachtet werden und das Skript für die Anfrage ist unter Listing A.8 zu finden. Bei einem Ausfall des *RecoveryMerge* ist daher davon auszugehen, dass Tupel verloren gehen, da dieser im Gegensatz zu anderen Operatoren nicht abgesichert ist.

Die Ergebnisse aus Tabelle 7.13 spiegeln dies wider. Wie bei der simplen Strategie dauert es hier bis zur *HoldOnMessage* bis die Bearbeitung angehalten wird. In den Durchläufen D3 bis D5 konnte das Recovery erfolgreich durchgeführt werden. Hier ist keine signifikante Änderung der Latenz oder des Durchsatzes nach dem Ausfall zu erkennen. Bei D1 und D2 kam es zu einer Ausgabe von doppelten Tupel nach dem Recovery, weshalb diese Durchläufe als gescheitert gewertet werden. Die Ursache dafür bleibt ungeklärt. Zudem kann beobachtet werden, dass durch die doppelte Ausgabe auch der Durchsatz in etwa halbiert wurde, da dies vermutlich eine höhere Netzwerklast verursacht hat. Die lange Dauer des Recovery-Prozess in D1, D3 und D4 kann dadurch erklärt werden, dass Peer7 den *RecoveryMerge* nicht installieren konnte und deshalb neu allokiert werden musste. Wie auch in anderen Szenarien zu beobachten, trat auch hier das Problem auf, dass Query-Parts mehrfach durch verschiedene Peers wiederhergestellt wurden. Dies ist auf eine zu geringe Wartezeit in der Verhandlungsphase zurückzuführen, die in einem Netzwerk mit Peers auf verteilten Rechnern höher als die eingestellten drei Sekunden sein muss. Ein Beispiel für den Zustand nach dem Recovery lässt sich im Anhang unter Abbildung A.6 finden. In diesem Fall wird D1 gezeigt.

An diesem Szenario kann erkannt werden, dass das Active-Standby mit der simplem Standardstrategie (Amnesia) zusammenarbeitet. Durch den automatischen Rückfall auf Amnesia in dem Fall, dass Active-Standby nicht funktioniert, da ein ungünstiger Peer ausfällt, kann die Ausfallsicherheit erhöht werden.

7.4.2.4 Szenario 4: Analysezeit (Mitte)

Innerhalb dieses Szenarios wurde getestet, wie das Verhalten des Active-Standbys beeinflusst wird, wenn ein Peer mit einem zustandslosen Operator ausfällt. Die Operatoren *RecoveryMerge*, *CalcLatency*, die Quelle und die Senke wurden dabei ausgenommen, um das Active-Standby unter günstigen Bedingungen testen zu können. Als Partitionierungsstrategie wurde *Operatorsetcloud* verwendet. Der initiale Zustand kann unter Abbildung A.7 betrachtet werden und das Skript für die Anfrage ist unter Listing A.8 zu finden. Bei einem Ausfall des Operators ist zu erwarten, dass keine Tupel verloren gehen, da dieser repliziert wird.

Aus Tabelle 7.14 ist ersichtlich, dass dies bei allen Durchläufen der Fall war. Alle Durchläufe lassen sich als erfolgreich einstufen und auch die Dauer des Recoverys liegt mit durchschnittlich sechs Sekunden im normalen Bereich der Versuchsreihe. Bemerkenswerte Werte ergeben sich für die Latenz und den Durchsatz. Während diese vor dem Ausfall im normalen Bereich liegen, steigen beide in der Pufferungsphase, also der Zeit zwischen *HoldOnMessage* und *GoOnMessage*, massiv an und normalisieren sich dann wieder. Dies ist darauf zurückzuführen, dass Peer6, welcher die Quelle und den *CalcLatency* besitzt, nach Ausfall eines der darauffolgenden Peers lokal die Berechnungen fortführt und seine Geschwindigkeit nicht mehr an die des Netzwerks anpasst und die Zwischenergebnisse puffert. Sobald die *GoOnMessage* ankommt wird der Puffer langsam abgebaut. Der Durchsatz wurde jedoch bereits von Peer6 berechnet und der Startzeitpunkt für die Berechnung der Latenz ebenfalls

AnalysezeitAS (Mitte)		D1	D2	D3	D4	D5	Ø
Ø-Latenz (in ms)	vorher	13511,6	159,7	150,1	576,9	154,3	2910,5
	puffernd	148480,4	27632,2	32929,2	35502,2	55218,6	59952,5
	nachher	-	67,4	143,3	59,3	136,1	101,5
Ø-Durchsatz (in Tupel/s)	vorher	1104,47	986,70	979,66	1149,86	1023,35	1048,81
	puffernd	28557,71	29444,99	33304,28	30987,35	29912,67	30441,40
	nachher	-	787,75	2085,62	983,43	1883,89	1435,17
Erfolgreich		Ja	Ja	Ja	Ja	Ja	Ja
Dauer (in ms)		4246	9722,5	5325	-	4051,5	5836,25
Ausfallzeitpunkt (Tupel)		94	89	15	20	10	45,6
Ende Pufferung (Tupel)		398	408	256	291	299	330,4
Verarbeitete Tupel		398	412	566	440	394	442
Verlorene Tupel		0	0	0	0	0	0

Tabelle 7.14: Szenario4: Analysezeit (Mitte)

bereits gesetzt. Ist der Puffer abgebaut, normalisieren sich die Werte wieder, da die Bearbeitung im Normalbetrieb fortgesetzt wird. Die Pufferung beim Active-Standby ist vermutlich dadurch zu erklären, dass obwohl es einen gültigen weiterverarbeitenden Peer gibt, dennoch eine *HoldOnMessage* versendet wird. Für die Bearbeitung hat es bis auf den Aufbau eines Puffers bei Peer6 jedoch keine weiteren Auswirkungen gezeigt. Dies ist für den Einsatz von Dateien als Quelle nicht problematisch, wenn jedoch Sensoren genutzt werden, könnten dadurch in der Zeit, in der die Pufferung stattfindet, Tupel verloren gehen, da diese nicht verarbeitet werden können. Nach dem Recovery konnte es vorkommen, dass ein Query-Part doppelt auf einem Peer vorlag, da dieser auf den bereits besitzenden Peer wiederhergestellt wurde und somit keine erhöhte Ausfallsicherheit mehr gegeben war. Dies liegt daran, dass die genutzte Allokierungsstrategie diesen Fall nicht gesondert behandelt. Ein Beispiel für den Zustand nach dem Recovery lässt sich im Anhang unter Abbildung A.8 finden. In diesem Fall wird D1 gezeigt.

Aus diesem Szenario gehen einige Verbesserungsmöglichkeiten für das Active-Standby hervor. Zum einen sollte das Pufferungs-Verhalten angepasst werden, um unnötige Pausen zu vermeiden. Außerdem ist das verwendete Allokationsverfahren nicht optimal für diese Strategie.

7.4.2.5 Szenario 5: Pässe (Aggregate)

Das fünfte Szenario testet das Verhalten von Herakles bei dem Ausfall eines Peers, welcher einen zustandsbehafteten Operator (hier: *Aggregate*) besitzt. Um den *Aggregate*-Operator bei der Verteilung zu isolieren, wurde in diesem Szenario die *User*-Strategie als Partitionierungsstrategie verwendet. Abbildung A.10 im Anhang zeigt die initiale Verteilung der Anfrage am Beispiel des ersten Durchlaufs. Das Skript für die Anfrage ist unter Listing A.9 zu finden. Es ist zu erwarten, dass die im Aggregate gespeicherten Zustände verloren gehen und somit die Zählung der Pässe nach dem Recovery bei null anfängt.

Pässe (Aggregate)		D1	D2	D3	D4	D5	Ø
Ø-Latenz (in ms)	vorher	35178,3	18366,6	17582,8	17410,5	12351,8	20178,0
	puffernd	-	-	-	-	-	-
	nachher	167150,4	79230,5	71430,4	55165,8	97426,3	94080,7
Ø-Durchsatz (in Tupel/s)	vorher	315,63	954,44	2011,35	746,93	809,32	967,53
	puffernd	-	-	-	-	-	-
	nachher	151,52	1028,72	955,84	1169,84	1178,35	896,85
Erfolgreich		Ja	Ja	Ja	Ja	Ja	Ja
Dauer (in ms)		6879	4477	5705	5966	7201	6045,6
Ausfallzeitpunkt (Tupel)		20	39	26	26	87	39,6
Ende Pufferung (Tupel)		-	-	-	-	-	-
Verarbeitete Tupel		450	472	490	463	549	484,8
Verlorene Tupel		alle bis zur Hold-On Message					-

Tabelle 7.15: Szenario5: Pässe (Aggregate)

Tabelle 7.15 zeigt wiederum die Ergebnisse der fünf durchgeführten Durchläufe. Bei allen Durchläufen war das Recovery erfolgreich und hat im Durchschnitt sechs Sekunden gedauert. Wie erwartet sind durch den Verlust des Zustandes des Aggregate-Operators alle Tupel bis zum erkennen des Ausfalls verloren gegangen. In Abbildung A.10 im Anhang sieht man, dass das Recovery auch in diesem Fall teils mehrfach durchgeführt wurde. Da hier allerdings nur ein Query-Part die Ergebnisse an den nachfolgenden Operator weitersendet, hat dies keinen signifikanten Einfluss auf die weitere Bearbeitung. Auch bei den Latenz- und Durchsatz-Werten ist hier kein Einfluss zu erkennen.

Beim Betrachten der Latenz und Durchsatz-Werte fällt auf, dass keine Pufferungsphase aus den Daten erkennbar ist. Dies liegt wahrscheinlich daran, dass Pässe nicht so regelmäßig auftreten wie die Sekunden bei der Analysezeit-Anfrage. Durch die geringere Zahl an Ergebnistupeln ist diese Phase in den Daten nicht so gut zu erkennen wie bei der Analysezeit, weil der Großteil der gepufferten Tupel nie bei der Senke ankommen und somit für diese auch kein Latenzwert vorhanden ist. Die Durchsatz-Werte unterscheiden sich vor und nach dem Recovery nur unwesentlich. Einzig beim dritten Durchlauf ist diese aus unbekannten Gründen deutlich abgesunken. Während des ersten Durchlaufs lief das Netzwerk sehr instabil, was die schlechteren Werte sowohl beim Durchsatz als auch bei der Latenz im Gegensatz zu den anderen Durchläufen erklären könnte. Beim Vergleich der Latenzwerte vor und nach dem Ausfall sieht man bei allen Durchläufen eine leicht steigende Tendenz. Dies ist darauf zurückzuführen, dass die Zahl der verarbeiteten Tupel nach dem Ausfall deutlich höher ist als vor dem Ausfall. Durch die Verwendung des *Heartbeat*-Operators steigt die Latenz mit der Zeit tendenziell an. Dieses Phänomen ist auch in Abschnitt 7.2.2.4 beschrieben.

7.4.2.6 Szenario 6: Pässe (Aggregate)

In diesem Szenario wurde getestet, wie das Verhalten des Active-Standbys beeinflusst wird, wenn ein Peer mit einem zustandsbehafteten Operator (*Aggregate*) ausfällt. Die Operatoren *RecoveryMerge*,

PässeAS (Aggregate)		D1	D2	D3	D4	D5	Ø
Ø-Latenz (in ms)	vorher	20704,9	13268,8	14642,4	12039,3	13116,0	14754,3
	puffernd	-	-	-	-	-	-
	nachher	26997,4	14268,0	24227,2	11781,7	15622,5	18579,4
Ø-Durchsatz (in Tupel/s)	vorher	603,48	901,74	870,38	950,61	703,60	805,96
	puffernd	-	-	-	-	-	-
	nachher	792,16	927,64	721,91	1627,59	1283,01	1070,46
Erfolgreich		Ja	Ja	Ja	Ja	Ja	Ja
Dauer (in ms)		5097	4294	6563	5400	-	5338,5
Ausfallzeitpunkt (Tupel)		222	73	32	78	35	88
Ende Pufferung (Tupel)		-	-	-	-	-	-
Verarbeitete Tupel		404	135	103	375	450	293,4
Verlorene Tupel		0	0	0	0	0	0

Tabelle 7.16: Szenario6: PässeAS (Aggregate)

CalcLatency, die Quelle und die Senke wurden dabei ausgenommen, um das Active-Standby unter günstigen Bedingungen testen zu können. Als Partitionierungsstrategie wurde *Querycloud* verwendet. Der initiale Zustand kann unter Abbildung A.11 betrachtet werden und das Skript für die Anfrage ist unter Listing A.10 zu finden. Bei einem Ausfall des Operators ist zu erwarten, dass keine Tupel verloren gehen, da dieser repliziert wird.

Wie in Szenario 4 (s. Abschnitt 7.4.2.4) lässt sich auch dieses Mal feststellen, dass dies bei allen Durchläufen der Fall war (vgl. Tabelle 7.16). Alle Durchläufe wurden erfolgreich durchgeführt. Die Besonderheit dieses Szenarios liegt darin, dass in diesem Fall ein zustandsbehafteter Operator wiederhergestellt wird. Da es sich um ein *Aggregate* handelt, kann dieser beispielsweise Summen enthalten. Diese gehen bei einem Ausfall verloren, da der Zustand mit der Strategie nicht wiederhergestellt wird. Ein auf einem neuen Peer neu wiederhergestelltes *Aggregate* fängt also wieder im Nullzustand an. Das beim Active-Standby parallel mitlaufende *Aggregate*, welches nicht ausgefallen ist, behält hingegen den aufgebauten Zustand. Die Ergebnisse des entsprechenden Query-Parts mit dem richtigen Zustand werden deshalb korrekterweise bevorzugt. Die Dauer des Recoverys liegt mit durchschnittlich gut fünf Sekunden im normalen Bereich der Versuchsreihe, bei D5 konnte die Dauer nicht nachvollzogen werden, da ein entsprechender Log-Eintrag nicht vorlag. Die Anfrage wurde von der Verteilungsstrategie auf drei Peers aufgeteilt. Damit gab es einen Peer, der vor dem Ausfall keinen Query-Part besaß. Bei D1 bis D3 bekam dieser Peer den ausgefallen Query-Part, bei D4 und D5 hat jedoch einer der beiden bereits teilnehmenden Peers den Query-Part bekommen. Dies führte dazu, dass sie nur noch zu zweit die Bearbeitung durchgeführt haben und in Tabelle 7.16 lässt sich erkennen, dass dies zu einem höheren Durchsatz nach dem Recovery geführt hat. Das Problem der Allokierungsstrategie aus Szenario 4 (s. Abschnitt 7.4.2.4) tritt auch hier wieder auf. Ein Beispiel für den Zustand nach dem Recovery lässt sich im Anhang unter Abbildung A.12 finden. In diesem Fall wird D1 gezeigt.

7.4.3 Fazit

Aus den Ergebnissen der Szenarien 1 bis 6 lässt sich feststellen, dass der Recovery-Prozess in 26 von 30 Fällen erfolgreich durchgeführt werden konnte, sodass nach Abschluss des Recoverys gemäß der angegebenen Definition korrekte Ergebnisse ausgegeben werden konnten. Für die Latenz und den Durchsatz kann in der Regel keine massive Veränderung vor dem Ausfall eines Peers und dem Abschluss des Recovery-Prozesses festgestellt werden, solange das Netzwerk stabil bleibt.

In Einzelfällen kommt es in der Pufferungsphase zwischen *HoldOnMessage* und *GoOnMessage* zu einer lokalen Bearbeitung und damit dem Aufbau eines großen Puffers, da die Datenrate in die Höhe steigt. Der Puffer wird nach dem Recovery langsam wieder abgebaut. Die Evaluation wurde aus Kapazitätsgründen mit csv-Dateien als Datenquelle durchgeführt. Bei einer Nutzung von Sensoren wäre die Datenrate zusätzlich durch die Datenrate der Sensoren limitiert und nicht nur durch die Kapazität des ersten Peers und würde daher vermutlich weitestgehend stabil bleiben. Der Puffer würde dennoch aufgebaut werden, aber nicht so schnell wachsen. In der Zeit, in welcher der Puffer abgebaut wird, könnte der bearbeitende Peer keine neuen Daten von den Sensoren verarbeiten und damit würden in dieser Zeit Tupel verloren gehen. Dies stellt auch gleichzeitig ein grundsätzliches Problem des Puffer dar.

Es ist möglich, dass Query-Parts mehrfach durch verschiedene Peers wiederhergestellt werden, wenn eine zu geringe Wartezeit in der Verhandlungsphase eingestellt ist. Diese sollte für ein Netzwerk mit Peers auf verteilten Rechnern erhöht werden, damit die Peers mehr Zeit haben, um miteinander zu kommunizieren und auszuhandeln, wer das Recovery übernimmt. Ein Fehler, der möglicherweise im Zusammenhang mit dem letztgenannten Problem steht, hat in 4 der 30 Fälle eine doppelte Ausgabe von Tupeln verursacht. Dies scheint ein allgemeines Problem der Recovery-Implementierung zu sein und bedarf gründlicher Fehlersuche.

Das Active-Standby sorgt durch die Replikation für zusätzliche Ausfallsicherheit und kann unter Einschränkungen den Datenverlust durch verlorene Tupel beseitigen (vgl. Szenario 4 und 6). Wenn ein Peer mit einem nicht replizierten Query-Part, wie z.B. die Quelle oder dem *RecoveryMerge*, ausfällt, gilt dies jedoch nicht (vgl. Szenario 3). Des weiteren können Zustände von Operatoren, wie beispielsweise dem *Aggregate*, nicht wiederhergestellt werden und sind verloren (vgl. Szenario 6). Um zu verhindern, dass nach Abschluss des Recoverys zwei identische Query-Parts auf einem Peer laufen und somit die Ausfallsicherheit nicht mehr gegeben ist, muss eine entsprechende Allokierungsstrategie für das Active-Standby entworfen werden, die dies verhindert. Die Pufferung beim Active-Standby, wie sie z.B. in Szenario 4 aufgetreten ist, obwohl es einen gültigen weiterverarbeitenden Peer gibt, muss verhindert werden. Für den Einsatz von Dateien als Quelle ist dieses Problem nicht problematisch, wenn jedoch Sensoren genutzt werden, könnten dadurch in der Zeit, in der die Pufferung stattfindet, Tupel verloren gehen, da diese nicht verarbeitet werden können. Um dies zu verhindern, muss die Fehlerquelle identifiziert und überarbeitet werden, sodass die Bearbeitung weitergeführt werden kann, wenn es einen oder mehrere aktive Peers gibt. Wird dies umgesetzt, ergibt sich für das Active-Standby, dass beim Einsatz von Sensoren keine Tupel durch eine Pufferung verloren gehen.

Insgesamt lässt sich zusammenfassen, dass die Erweiterung von Odysseus P2P um das Recovery erfolgreich durchgeführt wurde. Kleinere Fehler sind zwar noch vorhanden und weitere Verbesserungen notwendig, das Grundprinzip ist jedoch umgesetzt und funktioniert und sorgt somit für mehr Ausfallsicherheit.

7.5 Coach App

In diesem Teilkapitel wird die Evaluation der Coach App beschrieben. Neben einem Performance-Test bzgl. der Dauer bis zur endgültigen Darstellung von Ergebnissen wurden hauptsächlich funktionale Tests durchgeführt.

7.5.1 Funktionalität

Um die Coach App zu evaluieren, wurde ein Funktionstest mit einem Odysseus-Peer und einem Android Tablet, auf dem die App installiert ist, durchgeführt. Als Datenquelle dient der DEBS-Datenstrom. Zudem befinden sich das Tablet sowie die Odysseus-Instanz in demselben WLAN und in derselben Peer-Group (*LSOP*). Im Folgenden werden die Funktionen der Coach App aufgelistet und hinsichtlich ihrer Erfüllung überprüft.

1. Die App findet automatisch das P2P-Netzwerk und tritt diesem bei

Nach dem Start der Coach App wird keine feste IP Adresse des Peers vorgegeben, sodass sie das Netzwerk selbst finden muss. Mit einer Tippgeste auf dem Sport-Symbol in der Sportauswahl wird der Vorgang gestartet. Dabei stellt sich heraus, dass die Coach App das Netzwerk nicht findet – es wird eine Fehlermeldung „No peer found in peer group: LSOP“ ausgegeben. Dies ist damit zu erklären, dass das Netzwerk UDP-Pakete blockiert, die für der Broadcast und somit für das Auffinden der Peers in dem Netzwerk genutzt werden.

2. Die Anfragen lassen sich aus der Coach App heraus steuern

In der gestarteten App werden nacheinander die Funktionen *Pausieren*, *Starten*, *Stoppen* und *Löschen* der Anfragen getestet. Durch einen Klick auf *Pausieren* werden die Anfragen in der Odysseus-Instanz pausiert und auf *INACTIVE* gesetzt. Die Statistiken und Spielerpositionen ändern sich dadurch nicht mehr. Ein Klick auf *Starten* führt dazu, dass die Anfragen wieder gestartet werden und die Statistiken und Positionsdaten fortgeführt werden. Über die *Stoppen*-Funktion erhalten die Anfragen in Odysseus den Status *INACTIVE* und werden angehalten, sodass die Coach App keine neuen Daten mehr erhält. Allerdings können die Anfragen mit der *Starten*-Funktion nicht erneut gestartet werden – sie bleiben *INACTIVE*. Die *Löschen*-Funktion führt dazu, dass die Anfragen in der Odysseus-Instanz entfernt werden. Allerdings können auf Grund eines Fehlers in Odysseus nicht mit Hilfe der *Starten*-Funktion die Anfragen installiert werden. Es muss zunächst die Quelle erneut installiert werden.

3. Es lassen sich unterschiedliche Sportarten auswählen

Im Auswahlbildschirm werden jeweils nacheinander die Sportarten Fußball und Basketball ausgewählt. Beide starten die für ihre Sportart festgelegten gültigen Anfragen. Zudem werden abhängig von der gewählten Sportart unterschiedliche Statistiken geladen und das Spielfeld geladen.

4. Spieler können zu einer Kette hinzugefügt werden, die auf dem Spielfeld visualisiert wird

Die auf dem Spielfeld dargestellten Spieler lassen sich auswählen. Anschließend wird über die Actionbar (in diesem Fall die CAB) eine Kette angelegt, wodurch sich ein Dialog öffnet, in dem die

Farbe und eine Bezeichnung festgelegt wird. Die Kette wird zwischen den ausgewählten Spielern mit Linien in der festgelegten Farbe dargestellt. Um die Funktion zu testen, mit der Spieler aus einer Kette wieder entfernt werden können, wird ein Spieler aus einer Kette angewählt und in der CAB die entsprechende Schaltfläche getippt. Der Spieler wird aus der Kette entfernt. Zudem lassen sich angelegte Ketten über die Seitenleiste in ihrer Farbe und Bezeichnung bearbeiten sowie ausblenden und löschen.

5. Die Laufpfade von Spielern können verwaltet und auf dem Spielfeld visualisiert werden

Um die Funktion der visualisierten Laufpfade zu testen, wird ein Spieler auf dem Spielfeld ausgewählt und die *Laufpfad anzeigen*-Funktion aufgerufen. An dieser Stelle können Farbe und Länge des Pfades festgelegt werden. Um die Korrektheit dieser Funktionen zu testen, wurden mehrere Kombinationen vorgenommen und anschließend auf dem Spielfeld visualisiert. Auch wurde getestet, ob die Pfade mehrerer Spieler gleichzeitig dargestellt werden können. Dazu wurden mehrere Spieler ausgewählt und für sie die Pfade dargestellt. Auf dem Spielfeld werden dann die zuvor ausgewählten Spieler mit ihrem Laufpfad angezeigt. Zu Letzt wird noch die *Löschen*-Funktion getestet indem ein Pfad ausgewählt und entfernt wird, sodass er nicht mehr auf dem Spielfeld angezeigt wird.

6. Die Statistiken für einzelne Spieler können abgerufen und es können Spieler miteinander verglichen werden

Spieler können über die Spielerliste angewählt werden, wodurch sich in der Seitenleiste eine kleine Statistik-Ansicht des Spielers öffnet. In der Ansicht finden sich verschiedene Statistiken wieder, deren Werte automatisch aktualisiert werden. Über die Ansicht-Auswahl in der Actionbar steht die Möglichkeit bereit eine Vollbildansicht für Spieler-Statistiken anzuwählen. Diese wird angewählt und ein Spieler, für den die Statistiken angezeigt werden sollen, in der Auswahlliste angeklickt. Es werden die Statistiken des Spielers angezeigt. Auch hier werden die Statistiken automatisch aktualisiert. Um zwei Spieler zu vergleichen, werden in dieser Ansicht über Checkboxen in der Spielerliste zwei Spieler markiert, die verglichen werden sollen. Nachdem die Spieler ausgewählt wurden, werden diese Spieler in der Ansicht gegenüber gestellt und deren Statistikwerte mit Balkengrafiken visualisiert.

7. Es lassen sich Teamstatistiken anzeigen und anordnen

Am unteren Bildschirmrand werden die Teamstatistiken in Form von Ringdiagrammen angezeigt, deren Werte sich dynamisch ändern. Hier werden auch nur die Statistiken angezeigt, deren Anfragen gestartet wurden – also zuvor in den Einstellungen der Coach App aktiviert wurden. Über eine Schaltfläche rechts in der Teamstatistik-Ansicht wird über einen Menüknopf ein Dialog geöffnet, über den sich die Statistiken anordnen und ausblenden lassen. Eine Änderung bewirkt eine Neusortierung bzw. blendet die Ringdiagramme ein oder aus.

8. Es werden die Bereiche, in denen sich ein Spieler aufhält, in einer Heatmap dargestellt

Um die Heatmap anzuzeigen, wird zunächst ein Spieler ausgewählt und anschließend die entsprechende Funktion über die Actionbar aufgerufen. Dadurch wird auf dem Spielfeld eine Heatmap eingeblendet. Die Farben ändern sich je nach dem, ob sich der Spieler an einer Position länger aufhält oder nicht.

9. Auf dem Spielfeld kann gezeichnet werden

Über die Actionbar wird die Funktion für das Zeichnen auf dem Spielfeld aufgerufen. Es können verschiedene Farben und Zeichenstärken ausgewählt werden, mit denen auf dem Spielfeld gezeichnet wird. Mit der *Radiergummi*-Funktion ist es möglich, das Gezeichnete an den gewünschten Stellen wieder zu entfernen. Die *Mülltonnen*-Funktion löscht die gesamte Zeichnung. Anschließend wird noch die *Aus*- bzw. *Einblenden*-Funktion getestet, was ebenfalls funktioniert.

7.5.2 Performanz

Ziel der Evaluation ist die Bestimmung der Zeit, die zur Darstellung der Ergebnisse einer Anfrage vergeht, um daraus Rückschlüsse bzgl. der Performance gewinnen zu können. Zum Versuchsaufbau zählt dabei ein Tablet der Marke Samsung (SM-T535) mit der Android Version 4.4.2, um die Ergebnisse zu präsentieren. Zur Durchführung einer Anfrage wurde Odysseus auf Peer 12 (siehe 7.1) installiert.

7.5.2.1 Versuchsdurchführung

Innerhalb der Coach App wurde eine Stoppuhr implementiert, die beim Eingang von Resultaten für eine Anfrage gestartet wird. Sobald die Ergebnisse auf dem Tablet dargestellt werden, wird die Uhr gestoppt und die vergangene Zeit in die Konsole eingetragen. Mittels eines Excel-Dokuments wurden die Zeiten ausgewertet und in Boxplots dargestellt.

7.5.2.2 Ergebnisse

In der Abbildung 7.33 sind die Ergebnisse der Evaluation zu sehen. Dabei wurde die Zeit vom Eingang des Ergebnisses (per Socketverbindung) bis zur endgültigen Darstellung gemessen. Die Positionsanfrage *Game* wurde als Erstes ausgewählt. Das Ergebnis der Anfrage ist die Darstellung von Elementen auf dem Spielfeld. Der mittlere Wert für die Verarbeitung und die Anzeige eines Ergebnisses beträgt 0,072 ms. Während der minimale Wert von 0,042 ms nur gering vom mittleren Wert abweicht, ist der maximale Wert mit 17,39 ms relativ hoch. Die zweite Anfrage bestimmt die aktuelle Laufstrecke des Teams (*MileageTeam*). Ziel dieser Anfrage ist es, einen geeigneten Wert für die Laufstrecke eines gesamten Teams im Verhältnis zum anderen Team in einem Ringdiagramm darzustellen. Hier ist allgemein festzustellen, dass die Verarbeitungszeit zugenommen hat und der mittlere Wert bei ca. 8,19 ms liegt. Die längste Zeit wurde mit knapp 121,25 ms gemessen. Die schnellste Anzeige brauchte nur knapp 0,15 ms. Als dritte Anfrage wurde die aktuelle Laufstrecke eines einzelnen Spielers ausgewählt (*MileagePlayer*). Hier wird mittels eines Textfelds der Wert auf dem Tablet dargestellt. Zeitlich ordnet sich der mittlere Wert von 2,83 ms zwischen der *MileageTeam*-Anfrage und der *Game*-Anfrage ein. Erwähnenswert ist besonders der hohe maximale Wert von 146,13 ms. Durch die *MileageComparison*-Anfrage als vierte Anfrage sollen beim Spielervergleich Ergebnisse mittels Balkendiagramm dargestellt werden. Hier steigt der mittlere Wert wieder an und ist auch gleichzeitig der höchste mit einer Verarbeitungsdauer von knapp 12,32 ms. Der maximale Wert beträgt 49,54 ms. Die schnellste Verarbeitung dauerte gerade einmal 0,11 ms. Als letzte Anfrage wurde die *SprintsPlayer*-Anfrage getestet. Der mittlere Wert von 4,39 ms ist hier im Verhältnis ungefähr auf Höhe des Wertes der *MileagePlayer*-Anfrage. Das Ziel ist ebenfalls die textuelle Darstellung. Die Höchste Verarbeitungsdauer liegt bei 24,34 ms.

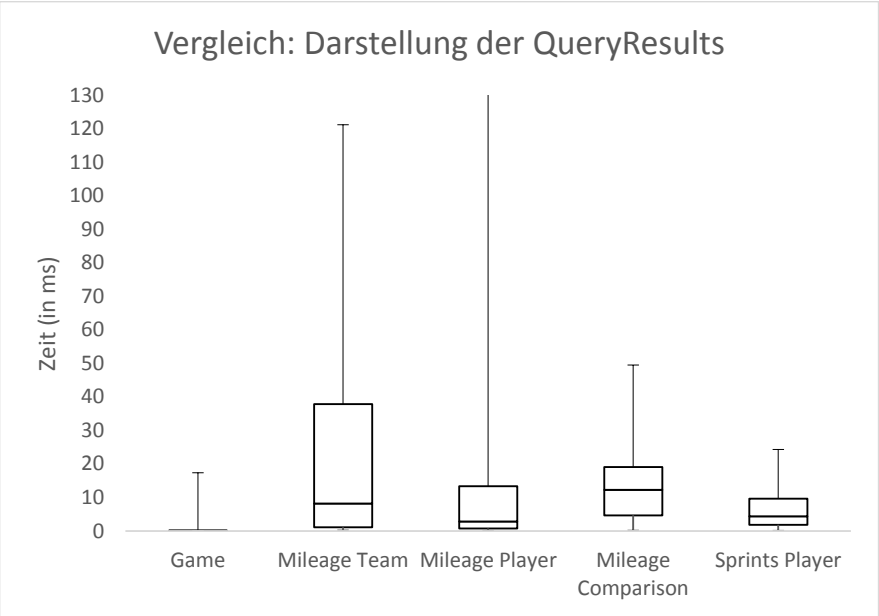


Abbildung 7.33: Zeit bis zur Darstellung der QueryResults

7.5.2.3 Auswertung

Die *Game*-Anfrage hat die kürzeste Verarbeitungszeit, was an der geringen Anzahl an Verarbeitungsschritten liegt. *MileageTeam*-Anfrage und *MileageComparison*-Anfrage besitzen im Verhältnis zu den anderen Anfragen die längsten Verarbeitungszeiten. Dies liegt daran, dass während der Verarbeitung nicht mit einfachen Texten gearbeitet wird, sondern graphische Elemente genutzt werden, um die Ergebnisse zu präsentieren. Hinzu kommt bei den Teamstatistiken und Vergleichen die Berechnung der prozentualen Anteile in den unterschiedlichen Diagrammtypen. Die zum Teil sehr hohen Maximalwerte lassen sich damit erklären, dass Android-Prozesse im Hintergrund aktiv sind und die Evaluation beeinflussen. Die geringeren Werte kommen zustande, weil vom System mittels Heartbeat-Werte in einem festgelegtem Zeitintervall ankommen und diese nicht zwangsläufig Änderungen an Diagrammen nach sich ziehen. Insgesamt kann also festgestellt werden, dass die Verarbeitung der Anfragen bis zur endgültigen Anzeige auf dem Tablet sehr schnell ausgeführt wird. Die Abweichungen zwischen den Anfragen kommen daher, dass mehr Zeit für graphische Darstellungen benötigt wird als für textuelle Darstellung.

7.6 Live-Positionsbestimmung

Um die Positionen von Spielern live überwachen zu können, wurden im Konzept im Abschnitt 5.1 Möglichkeiten zur Echtzeitlokalisation vorgestellt, die mit der GPSender App umgesetzt wurden. Die Implementierung dieser ist in Abschnitt 6.1 beschrieben. Um die Funktionsfähigkeit und Einsetzbarkeit dieser Methoden unter realistischen Bedingungen überprüfen zu können, wurden sowohl die Aufzeichnung mittels GPS als auch mittels Triangulation über WLAN-Zugangspunkte getestet.

7.6.1 GPS Ortung

Die GPSender App mit GPS-Ortung zur Positionsbestimmung der Spieler wurde auf einem Sportplatz unter freiem Himmel getestet. 13 Spieler und ein Ball wurden mit Smartphones ausgestattet, auf denen die GPSender App aktiv war. Die Spieler trugen die Geräte in den Taschen. Der Schaumstoffball war so präpariert, dass ein Smartphone in diesen Ball gesteckt werden konnte. Die Geräte waren über einen WLAN-Zugangspunkt am Rand des Spielfeldes verbunden und sendeten über diesen die Daten an ein Notebook mit der GPServer Anwendung. Die Daten wurden in einer CSV-Datei gespeichert.

Der Test ergab, dass die Positionsdaten der Spieler genau genug sind, um daraus Spielstatistiken auf Basis von Herakles berechnen zu können. Jedoch kann es vorkommen, dass die Positionen der einzelnen Geräte voneinander abweichen. Das Smartphone im Ball etwa, ein älteres „Samsung Galaxy S2“ lieferte ungenauere Daten als neuere Geräte. So kam es vor, dass ein Spieler, der den Ball in der Hand hielt, etwa einen Meter neben dem Ball angezeigt wurde. Trotz dieser Ungenauigkeiten war es möglich, die Daten mit den erstellten Anfragen auszuwerten, da diese gegenüber leichten Ungenauigkeiten robust sind.

Der Test der GPS Ortung wurde als Erfolg gewertet und zeigte, dass grundsätzlich auch mit günstiger bzw. vorhandener Hardware eine Spielanalyse möglich ist.

7.6.2 Triangulation mittels WLAN

Neben der GPS-Ortung wurde die im Konzept beschriebene Ortung über die Triangulation mithilfe von WLAN-Zugangspunkten umgesetzt und getestet. Dieses bietet grundsätzlich den Vorteil, dass sie auch innerhalb von geschlossenen Räumen funktioniert, in denen eine Ortung via GPS nicht möglich ist. Um den Test durchzuführen, wurden in einer Sporthalle vier WLAN-Zugangspunkte installiert. Drei von diesen dienten zur Triangulation, einer zum Empfangen der Positionsdaten. Die Abstände zwischen den drei Zugangspunkten zur Ortung wurden ausgemessen, um diese zur Berechnung der Positionen nutzen zu können. Wie auch bei der Ortung via GPS waren die Spieler und der Ball mit Smartphones ausgestattet, auf denen die GPSender App aktiv war. Der Versuchsaufbau ist in Abbildung 7.34 skizziert.

Nach dem Aufbau nach obigem Schema wurde auf dem Spielfeld gespielt. Die Daten kamen korrekt bei dem Notebook an und konnten auch auf dem Tablet über die Coach App visualisiert werden. Jedoch waren die Daten viel zu ungenau, um daraus irgendwelche Statistiken ziehen zu können oder auch nur einen Überblick über das Spiel zu erhalten. Die Positionen der Spieler änderten sich sprunghaft über das gesamte Feld und darüber hinaus. Für dieses Fehlverhalten kommen einige Ursachen in Frage, die vermutlich zusammen die große Ungenauigkeit hervorgerufen haben.

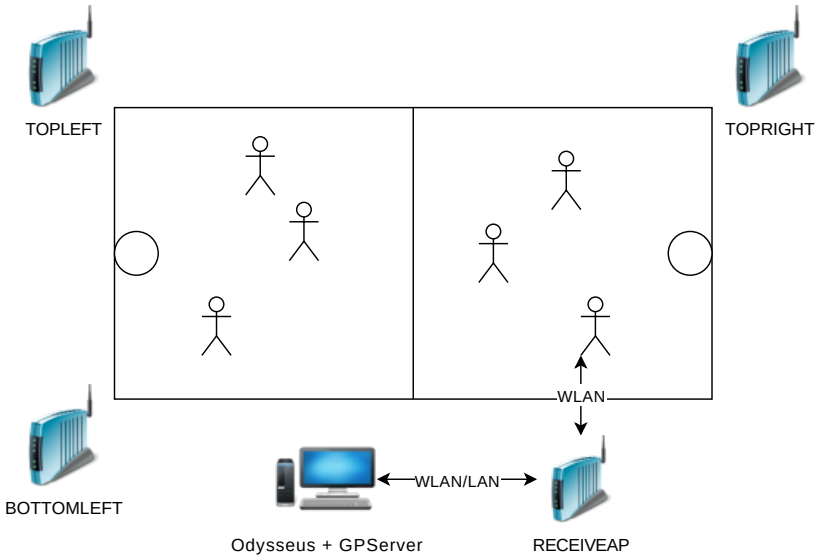


Abbildung 7.34: Versuchsaufbau bei der WLAN Triangulation

Zum einen waren sowohl die WLAN-Zugangspunkte als auch die Smartphones heterogen. Die Zugangspunkte haben also vermutlich unterschiedlich stark gesendet und jedes Smartphone hat durch die Bauart unterschiedlich guten Empfang. Um die Positionen der Spieler genau zu bestimmen ist es jedoch notwendig, dass die Zugangspunkte als auch die Empfänger, in diesem Fall Smartphones, möglichst homogen arbeiten. Eine zweite mögliche Ursache sind Reflexionen. Die Strahlung der WLAN-Zugangspunkte kann an den Wänden der Sporthalle reflektiert werden, sodass an einigen Orten der Empfang für die Smartphones besser ist als in einem idealen, leeren Raum zu erwarten wäre. Beide Fehlerursachen lassen sich teilweise durch Algorithmen herausrechnen, was jedoch eine schwierige Problemstellung ist. Der verwendete Algorithmus in unserer Anwendung ist jedoch nur sehr einfach, weshalb diese Ursachen deutliche Einbußen in der Qualität der Daten zur Folge haben.

Eine dritte Störquelle sind Hindernisse, die den Empfang stören können. In diesem Fall sind dies die anderen Spieler auf dem Feld, die kurzfristig und unvorhergesehen die Ausbreitung des WLAN-Signals beeinflussen können. Um diesen Störfaktor auszuschließen, wurde ein Test mit nur einem Spieler unternommen. Dies brachte jedoch kaum Besserung, da anscheinend andere Fehlerursachen die Qualität der Daten schon zu stark beeinträchtigen.

Die Größe des Feldes ist ein weiteres mögliches Problem. Die Smartphones konnten nicht zu jeder Zeit alle Zugangspunkte erkennen. Um diesen Fehler einzudämmen, wurde das Spielfeld in unterschiedlichen Größen eingerichtet. Bei dem größten Feld, das etwa 28 m x 15 m groß war, traten diese Probleme auf. Nach einer Halbierung der langen Seite, also einer Reduzierung auf etwa 14 m x 15 m, war das Problem beseitigt, dass einige Geräte die Zugangspunkte nicht mehr sehen konnten. Jedoch blieb das Problem der ungenauen Daten unverändert bestehen.

Abschließend betrachtet war die Aufzeichnung mittels Triangulation über WLAN-Zugangspunkte nicht erfolgreich. Es wurde festgestellt, dass diese Technik einen deutlich höheren Entwicklungsaufwand benötigt, um auf dem Spielfeld funktionstüchtig zu sein. Mögliche Fehlerursachen wurden

ermittelt, jedoch aufgrund des Fokus der Projektgruppe auf die Datenanalyse und Aufbereitung und nicht der Aufzeichnung der Daten nicht weiter verfolgt.

7.7 Monitoring App

Zur Evaluation der Monitoring App wurde ein Funktionstest mit drei Odysseus-Peers ausgeführt. Im Folgenden wird die Monitoring App hinsichtlich der funktionalen Anforderungen überprüft.

1. Die Monitoring App gibt eine Übersicht über alle verfügbaren Odysseus-Peers in dem Netzwerk

Nachdem alle drei Peers gestartet wurden, zeigt die Monitoring App die drei Peers in Form einer Tabelle an. Zu jedem Peer werden u.a. Daten wie Name des Peers, Anzahl an Anfragen oder Auslastung der CPU dargestellt. Die Daten werden als korrekt angesehen, da sie mit den Daten aus den Odysseus-Studios der Peers übereinstimmen. Die Aktualität der Daten ist ebenfalls gewährleistet, da sie alle sechs bis acht Sekunden aktualisiert werden. Zudem wird ein Odysseus-Peer im Durchschnitt in weniger als 30 Sekunden nach dem Starten des Peers in der Monitoring App angezeigt. Nach Beenden eines Peers wird dieser ebenfalls im Durchschnitt in weniger als 30 Sekunden nicht mehr von der Monitoring App angezeigt. Es kommt manchmal vor, dass ein Peer in der Monitoring App nicht mehr angezeigt wird, obwohl dieser im Netzwerk aktiv ist und auch von anderen Odysseus-Peers gefunden wird. Dabei kann ausgeschlossen werden, dass dies auf einen Anzeigefehler zurückzuführen ist, da aus den Logs von JXTA ermittelt werden konnte, dass JXTA den Peer ebenfalls nicht findet. Die Einschränkung muss letztlich akzeptiert werden, da die Monitoring App selber ein Peer im Netzwerk ist und es in einem P2P-Netzwerk keine Garantie geben kann, dass jeder Peer alle anderen Peers sieht.

2. Innerhalb der Monitoring App ist es möglich, die Peers über eine Konsole zu steuern

In der Konsole ist es über ein Textfeld möglich, einen Befehl an diejenigen Peers zu senden, die in der darunter angezeigten Tabelle ausgewählt wurden. Es wurden zum Testen die Befehle „lsPeers“, „peerStatus“, und „help“ ausgeführt. Zu allen drei Befehlen wurde eine Antwort in angemessener Zeit geliefert. Die Korrektheit konnte erfolgreich überprüft werden, in dem die Antworten mit den Antworten verglichen wurden, die die Konsole innerhalb des Odysseus-Studios zu den Befehlen geliefert hat. Zum Ausführen der Befehle ist ein Login notwendig gewesen, der über den Befehl „loginPeer“ möglich war. Ohne Login konnten zwar Befehle an die Peers gesendet werden, es sind jedoch keine Antworten angekommen. Zum Ausloggen konnte der Befehl „logoutPeer“ verwendet werden.

3. Die Monitoring App bietet die Möglichkeit, die Ping-Map eines Peers anzeigen zu lassen

Nach Öffnen einer Ping-Map wird in der Mitte der Peer als Punkt dargestellt, zu dem die Ping-Map aufgerufen wird. Die Monitoring App wird als grüner Punkt in der Ping-Map visualisiert, sodass dieser einfach von den Odysseus-Peers unterschieden werden kann. Um einen Punkt in der Ping-Map zu einem Peer zuzuordnen, kann ein Tooltip angezeigt werden, indem die Maus über den Punkt bewegt wird. Der Tooltip zeigt den Namen des Peers und den Ping an. Die Korrektheit und Aktualität der Ping-Map konnte erfolgreich überprüft werden, indem die Ping-Map mit der Ping-Map aus dem Odysseus-Studio verglichen wurde. Hier wurden nur minimale Abweichungen erkannt. Ein Problem bei der Ping-Map entsteht, wenn die Peers zu weit auseinander liegen, sodass diese nicht mehr in der

Ping-Map zu sehen sind. Es ist zwar möglich, aus der Ping-Map herauszuzoomen, jedoch werden die Punkte dann so klein, dass sie kaum noch zu sehen sind. Das Problem dabei ist, dass das Zoomen immer nur um den Mittelpunkt der Ping-Map geschieht.

4. Der Log eines Peers kann mit der Monitoring App angezeigt werden

Zum Überprüfen des Logs wurden in der OSGi-Konsole des Odysseus-Peers die Befehle „log error test“, „log debug test“ und „log info test“ aufgerufen. Die Monitoring App hat lediglich den ersten Log-Eintrag angezeigt. Der Grund dafür ist, dass der Standard-Log-Level in Odysseus auf „error“ gesetzt ist. Nach Ändern des Log-Levels auf „info“ wurden nach einem erneuten Test alle drei Log-Einträge in der Monitoring App angezeigt. Das Filtern der Log-Einträge nach Level und Schlüsselwörter hat einwandfrei funktioniert.

5. Verteilte Anfragen können über die Monitoring App visualisiert werden

Verteilte Anfragen werden in der Monitoring App visualisiert, indem ein Anfrageplan mit allen Operatoren dargestellt wird, die zu der verteilten Anfrage gehören. Um festzustellen, welcher Operator zu welchem Peer gehört, werden diese entsprechend mit unterschiedlichen Farben gekennzeichnet. Zur Überprüfung der Korrektheit eines verteilten Anfrageplans wurde die Ecken-Anfrage verteilt und der verteilte Anfrageplan mit den lokalen Anfrageplänen aus den Odysseus-Studios verglichen. Grundsätzlich stimmen die Anfragepläne überein, jedoch wird die Quelle unterschiedlich gehandhabt. Da die Quelle als View definiert wird, besteht sie nicht nur aus einem Access-Operator sondern auch aus weiteren Operatoren. Während die Monitoring App die Quelle als einen einzigen Knoten darstellt, zeigt der lokale Anfrageplan in Odysseus-Studio die einzelnen Operatoren an, aus der die Quelle zusammengesetzt ist. Je nachdem wie sehr der Nutzer an der Zusammensetzung der Quelle interessiert ist, haben beide Darstellungsweisen ihre Vor- und Nachteile. Für die Monitoring App wäre es empfehlenswert, wenn der Nutzer aussuchen könnte, welche Darstellungsweise er präferiert.

7.8 Zusammenfassung

In diesem Kapitel wurde die Evaluation von Herakles beschrieben. Dabei wurden zunächst die Anfragen von Herakles auf Performanz und Korrektheit überprüft. Die Ergebnisse der Anfragen sind dabei weitestgehend korrekt. Im Gegensatz dazu ist die Performance im verteilten Szenario deutlich schlechter als bei lokal ausgeführten Anfragen. Dies ist vor allem auf das Netzwerk zurückzuführen. Anschließend wurde eine Evaluation des Load-Balancings durchführt. Dabei hat sich herausgestellt, dass das Load-Balancing grundsätzlich funktioniert und die Prozessorlast auf den ausgelasteten Peers damit gesenkt werden kann. Es gibt jedoch Verbesserungspotential beim Neuverteilen von Teilanfragen, so dass die Last aktuell noch nicht optimal über mehrere Peers verteilt wird. Die Evaluation des Recoverys hat gezeigt, dass das Recovery in den meisten Fällen funktioniert. Durchsatz und Latenz bleiben dabei nach einem Recovery-Prozess meistens stabil. Ein Problem ist, dass es in einigen Situationen vorkommen kann, dass ein Peer sehr viele Datenstromelemente puffern muss und dieser Puffer erst langsam wieder abgebaut werden kann. Für die Coach App wurde ein Funktions- und Performanztest durchgeführt. Während der Funktionstest gezeigt hat, dass die meisten Anforderungen durch die Coach App umgesetzt wurden, konnte mit dem Performanztest festgestellt werden, dass die Dauer vom Empfang bis zur Anzeige der Daten sehr gering ist. Die Messung von Positionsdaten mit Hilfe der GPSender App hat gezeigt, dass das Messen mittels GPS Ortung erfolgreich war, während

eine Messung mit der Triangulation von WLAN-Zugangspunkten nicht erfolgreich war. Abschließend wurde ein Funktionstest der Monitoring App durchgeführt, bei dem eine erfolgreiche Umsetzung der Anforderungen festgestellt werden konnte.

8 Vorgehen im Projekt

Im Folgenden Kapitel wird das Vorgehen der Projektgruppe zur Erreichung des Projektziels vorgestellt. Zunächst wird dabei der Einsatz von Scrum beschrieben. Anschließend werden die Rollen vorgestellt, welche im Projekt von Bedeutung waren. Es folgt eine Zusammenfassung der durchgeführten Trainerinterviews, welche zu Projektbeginn mit zwei Trainern durchgeführt wurden um Anforderungen an Herakles zu erheben. Danach werden markante Ereignisse während der Projektlaufzeit in Form eines Projekttagebuchs präsentiert. Abschließend werden noch die Projektplanung und unsere Bemühungen um einen Kooperationspartner dargestellt.

8.1 Scrum

Wie bereits im Grundlagenkapitel erwähnt, handelt es sich beim Scrum um ein Agiles Vorgehensmodell. In den Nachfolgenden Abschnitten wird Scrum wie es in der Projektgruppe angewendet wurde näher erläutert.

8.1.1 User Stories

Da Scrum Flexibilität und Agilität in der Entwicklung gewährleistet, werden Anforderungen in der Regel nicht für den kompletten Projektzeitraum bereits in einer Planungsphase zu Beginn des Projektes ermittelt und ausgearbeitet, sondern zunächst grob beschrieben und dann je nach Bedarf verfeinert. Weiterhin werden Anforderungen nicht wie herkömmlich dokumentiert und festgeschrieben, sondern in Form von User Stories ermittelt.

User Stories beschreiben Anforderungen oder Funktionen aus Sicht eines Nutzers (Users). Diese werden nach folgendem Muster aufgebaut:

Als **<Rolle>** möchte ich **<Funktion>**, sodass/um **<Begründung>**.

Die Begründung zu einer Story ist relevant, um sich selbst bzw. dem Nutzer vor Augen zu führen, warum diese Anforderung/User Story sinnvoll und notwendig ist. Um eine User Story weiter zu konkretisieren, ist es möglich, zusätzliche Anforderungskriterien festzulegen.

Generell sollte bei der Erstellung von User Stories folgendes INVEST-Prinzip berücksichtigt werden:

- **I** (ndependent) - User Stories sollten möglichst unabhängig sein und nicht voneinander abhängen (natürlich können diese aufeinander aufbauen)
- **N** (egotiable) - User Stories sind verhandelbar und können auch nachträglich noch geändert werden
- **V** (aluable) - User Stories müssen einen festgelegten Mehrwert für den Anwender bringen
- **E** (stimateable) - User Stories müssen abschätzbar und somit überblickbar sein
- **S** (mall) - User Stories sollten möglichst klein sein, sodass diese in einem Sprint erledigt werden können und besser abschätzbar sind

- **T** (estable) - Das Ergebnis einer User Story sollte testbar/messbar sein

Übergeordnete User Stories deren Umfang nicht direkt abgeschätzt werden kann, werden in der Regel als Epic bezeichnet. Diese müssen für das eigentliche Planning in kleinere konkretere User Stories aufgeteilt werden. In der Regel werden Anwenderinteressen zunächst als Epic definiert und je nach Position im Backlog in kleinere Stories aufgeteilt.

8.1.2 Definition of Done

Innerhalb von Scrum-Teams ist es üblich, dass für die Abarbeitung von Tasks eine *Definition of Done* festgelegt wird. Diese Definition konkretisiert, welche Teilaufgaben bei der Bearbeitung von Tasks berücksichtigt werden. Bei einer konsequenten Einhaltung dieser Regeln, kann die gesamte Qualität der Arbeit und somit auch des Endproduktes gesteigert werden.

Innerhalb unseres Projektes haben wir uns für die folgenden Punkte entschieden, die allerdings immer von dem jeweiligen Aufgabengebiet abhängig waren. So ist das Schreiben von Unit-Tests beispielsweise nicht bei der Erstellung von Dokumentationen sinnvoll.

- Die geforderte Funktion ist vollständig umgesetzt
- Wenn möglich oder sinnvoll sind Unit-Tests vorhanden (evtl. Evaluationsframework)
- Testfälle sind in Confluence vorhanden
- Komponente, Funktion oder Schnittstelle ist in Confluence beschrieben
- Quellcode ist dokumentiert (JavaDoc verpflichtend für Klassen und Methoden, Kommentare wenn sinnvoll)
- Falls möglich: Evaluationsergebnisse / -ansätze sind dokumentiert (ggf. weitere User Story / Task in Jira erstellt)
- Falls erforderlich: Glossar wurde erweitert

Das Ziel dieser genannten Punkte war es vor allem, dass die Dokumentation aktuell bleibt und somit den momentanen Stand der Entwicklung beschreibt.

8.1.3 Scrum-Rollen

In unserem Projekt wurde bei Scrum zwischen den drei Rollen Product Owner, Scrum Master und Entwicklungsteam unterschieden. Zusammengefasst bilden diese drei Rollen das Scrum-Team. In den nachfolgenden Abschnitten werden die einzelnen Aufgaben, die diese Rollen während des Projektes beinhalteten, näher erläutert.

8.1.3.1 Product Owner

Die Rolle des Product Owners hat unser Betreuer Timo Michelsen übernommen. Timo Michelsen hat zu Beginn und während des Projektes Ziele vorgegeben, die für ein erfolgreiches Resultat erreicht werden sollten. Hierzu zählten unter anderem folgende übergeordnete Anforderungen:

- Die Umsetzung eines grundlegenden Load-Balancing Konzeptes
- Die Umsetzung eines grundlegenden Recovery-Prozesses
- Evaluation verschiedener Sensorsysteme und Techniken (RedFir, Piksi usw.)
- Die Unterstützung verschiedener Sportarten

Eine ausführliche Liste der Anforderungen ist im Kapitel 3 zu finden.

8.1.3.2 Scrum Master

Chris Tönjes-Deye und Pascal Schmedt haben die Rolle des Scrum Master übernommen. Sie haben während des Projektes stets darauf geachtet, dass das Konzept des Scrum Prozesses eingehalten wurde. Des Weiteren haben Sie dafür gesorgt, dass die einzelnen Sprints geplant und korrekt abgeschlossen werden können. Darüber hinaus haben Sie organisatorische Hindernisse sowie störende Einflüsse für das Scrum-Team beseitigt.

8.1.3.3 Entwicklungsteam

Zu der Rolle des Entwicklungsteams zählten Michael Brand, Tobias Brandt, Carsten Cordes, Simon Eilers, Simon Küspert, Torben Pehlke, Marc Preuschaft, Thomas Prünie, Pascal Schmedt, Thore Stratmann, Chris Tönjes-Deye und Marc Wilken. Während des Projektes wurden sämtliche Funktionen, Planungen und Dokumentationen vom Entwicklungsteam übernommen und durchgeführt. Durchschnittlich wurden von jedem Gruppenmitglied mehr als 14 Stunden pro Woche in das Projekt investiert.

8.1.4 Meetings

Anders als es beim Scrum vorgesehen ist, wurde nicht ein tägliches Meeting, sondern ein wöchentliches Meeting durchgeführt. Diese Anpassung war notwendig um die unterschiedlichen Vorlesungspläne der einzelnen Projektgruppenmitglieder zu berücksichtigen. Innerhalb des wöchentlichen Meetings wurden neben dem *Weekly Scrum* auch aktuelle Umsetzungen, Probleme oder Anmerkungen besprochen, um dem gesamten Projektteam eine bestmögliche Projektübersicht zu verschaffen. Im Nachfolgenden Abschnitt werden die einzelnen Bestandteile des wöchentlichen Meetings näher erläutert.

8.1.4.1 Weekly Scrum

Während des wöchentlichen Meetings wurde zu Beginn jeweils ein Weekly Scrum durchgeführt. Hierzu haben die Projektgruppenmitglieder über die geleistete Arbeit für die Projektgruppe berichtet. Dazu wurden neben den Erfolgen auch aktuelle Probleme kurz genannt, um den anderen Projektgruppenmitgliedern eine Übersicht des aktuellen Standes der Arbeit zu geben.

8.1.4.2 Review, Retrospektive und Planning

Am Ende eines jeden Sprints wurde das Review, die Retrospektive und das Planning durchgeführt. Das Review dient dazu, den Sprint zu reflektieren und zu kontrollieren, was im Sprint erreicht wurde und welche Ziele nicht erfüllt werden konnten. Hierzu wurden zunächst die Tasks und User Stories aus dem Tool Jira begutachtet, ob diese Abschlossen bzw. nicht erfüllt sind. Dabei wurde stets darauf geachtet, dass die Akzeptanzkriterien der jeweiligen Tasks erfüllt waren. Im Rahmen dessen wurde die neu umgesetzte Funktionalität, welche in das Produktinkrement einfloss, vorgestellt und von dem Entwicklungsteam sowie dem Product Owner begutachtet. Das Ergebnis des Reviews wurde entsprechend im Confluence dokumentiert. Nicht abgeschlossene User Stories wurden zurück in das Product Backlog verschoben und beim kommenden Planning erneut priorisiert und geschätzt.

Im Anschluss des Reviews wurde die Retrospektive durchgeführt. In diesem Teil des Meeting wurden die positiven und negativen Eindrücke bezüglich des Sprints besprochen. Diese wurden anschließend mit der Projektgruppe besprochen und wenn notwendig eine Lösung erarbeitet, um die jeweilige Problematik zu lösen.

Damit ein neuer Sprint gestartet werden konnte, wurde vor Beginn eines Sprints das sogenannte Planning durchgeführt. Im Vorfeld wurde zunächst eine Kapazitätsplanung durchgeführt, in der ermittelt wurde, wie viele Stunden insgesamt von den Projektgruppenmitgliedern für den kommenden Sprint aufgewendet werden können. In der Regel wurde diese Anzahl an Stunden pro Projektgruppenmitglied wie folgt berechnet: (Anzahl Wochen im Sprint) x (Wochenarbeitszeit - Sitzungsdauer) z.B. (3 x (14 - 2) = 36 Stunden). Die Stundenanzahl aller Projektgruppenmitglieder in Verbindung mit der Velocity konnte nun die Anzahl an Story Points berechnet werden, die in dem Sprint umgesetzt werden konnte. In Absprache mit dem Product Owner wurden die groben Ziele für den kommenden Sprint abgesprochen und die dazugehörigen User Stories neu priorisiert. Weiterhin wurden zu jeder User Story die entsprechenden Subtasks ermittelt und in Jira erfasst. Die Priorisierung sowie das erstellen der User Stories und deren Subtasks erfolgte überwiegend in den Tagen vor dem Meeting. Für das Planning wurden die Tasks ausgedruckt und von den Projektgruppenmitgliedern geschätzt. Diese Aufwandsschätzung wurde zu Beginn des Projektes mit Hilfe des Planning Pokers durchgeführt. Diese Schätzmethode wurde aber durch ihre Zeitintensivität nicht weiter genutzt. Stattdessen wurde die Aufwandsschätzung mit dem Magic Estimation Verfahren durchgeführt. Je nachdem wie viele Story Points zur Verfügung standen, wurden anschließend einzelne User Storys in den neuen Sprint übernommen. Weitere Erklärungen zu den einzelnen Aufwandsschätzmethoden sind im Abschnitt 8.1.5 zu finden.

8.1.5 Aufwandsschätzung

Es gibt verschiedene Methoden, dokumentierte Anforderungen zu schätzen. Im Laufe des Projektes wurden die beiden Methoden Planning Poker und Magic Estimation eingesetzt. Im nachfolgenden Abschnitt werden diese beiden Methoden in Bezug auf das Projekt näher erläutert.

8.1.5.1 Scrum-Karten

Um die Tasks, die sich im Backlog befanden, schätzen zu können, wurden diese für jedes Planning als Scrum-Karten exportiert. Auf einer Scrum-Karte befanden sich Informationen zu dem jeweiligen Task wie Beschreibung, Bezeichner, Ersteller und zugehörige User Story. Durch diese Informationen

war es später im Planning möglich, sich schnell und einfach einen Überblick über den jeweiligen Task zu verschaffen.

8.1.5.2 Planning Poker

Das Planning Poker wurde zu beginn der Projektgruppe im Sprint LSOP 2014-01 eingesetzt. Hierbei wurden zunächst die einzelnen Tasks vorgestellt. Im Anschluss daran, hat jedes Teammitglied den Aufwand des Tasks geschätzt. Diese Entscheidung wurde von jedem Teammitglied verdeckt durchgeführt. Sobald alle Teammitglieder eine Entscheidung getroffen hatten, forderte der Scrum-Master alle auf, die Planning Poker Karten offen zu zeigen. Im Anschluss wurden die Ausreißer explizit gefragt, wie sie zu dieser Entscheidung gekommen sind. Teilweise hat sich bei dieser Befragung herausgestellt, dass einzelne Teammitglieder den Task anders verstanden hatten bzw. mehr Hintergrundwissen zu diesem Task hatten. Nachdem diese Missverständnisse beseitigt wurden, fand eine erneute Schätzung statt. Dieser Vorgang wurde so lange wiederholt, bis es zu einer Einigung im Team gekommen ist. Dadurch, dass bei dieser Methode jeder einzelne Task vorgestellt und zunächst eine verdeckte Schätzung durchgeführt wurde, war diese Methode sehr zeitintensiv. Dies hatte zur Folge, dass das Planning alleine in etwa zwei Stunden gedauert hatte. Um eine schnellere Schätzung durchführen zu können, wurde im weiterem Verlauf des Projektes die Magic Estimation Methode eingesetzt.

8.1.5.3 Magic Estimation

Das Magic Estimation Verfahren eignet sich besonders gut, um größere Product Backlogs einzuschätzen. Dabei wurde ein Satz der Planning Poker Karten auf einem Tisch verteilt. Anschließend wurden die ausgedruckten Tasks, die sich im Product Backlog befanden gleichmäßig auf die Teammitglieder verteilt. Jedes Gruppenmitglied schätzt die Tasks, die es bekommen hat und ordnet sie der entsprechenden Planing Poker Karte auf dem Tisch zu. Hierbei wurde darauf geachtet, dass keine verbale/nonverbale Kommunikation zwischen den Teammitgliedern statt findet. Somit war gewährleistet, dass sich die einzelnen Mitglieder nicht gegenseitig beeinflussen. Sobald alle Tasks auf dem Tisch verteilt waren, hatte jedes Mitglied die Chance, Tasks umzusortieren, die seiner Meinung nach nicht korrekt zugeordnet wurden. Jedes mal wenn ein Task verschoben wurde, erhielt dieser eine Markierungen. Sobald ein Task drei Markierung erhalten hatte, wurde dieser aussortiert und anschließend mit dem Projektteam diskutiert, um eine Schätzung festzulegen.

8.1.6 Retrospektive

In der Retrospektive geht es darum, zunächst als Einzelperson und dann als Gruppe einen Sprint hinsichtlich des Arbeitsprozesses zu reflektieren. Im Folgenden werden Methoden hierfür beschrieben.

8.1.6.1 Handzettel

Hierbei geht es darum, dass jeder der Gruppe mitteilt, was seiner Meinung nach im letzten Sprint gut oder auch nicht so gut gelaufen ist. Damit bei diesem Vorgehen nicht zu viel Zeit verschwendet wurde, hatte jedes Gruppenmitglied fünf Minuten Zeit, seine Anmerkungen auf jeweils eine Karte zu schreiben. Im Anschluss hat jedes Teammitglied seine Punkte vor der Gruppe kurz präsentiert und an die Tafel geklebt. Hierbei wurde darauf geachtet, dass der Vortragende nicht unterbrochen wurde.

Sobald sich alle Teammitglieder geäußert hatten, wurden ähnliche Anmerkungen an der Tafel zusammengelegt und Maßnahmen in der Gruppe gesucht, sodass vor allem die negativen Punkte beseitigt werden und das Positive beibehalten wird. Die Ergebnisse wurden im Sprint-Bereich hinterlegt, sodass nachvollziehbar ist, welche Punkte bei der Retrospektive geäußert wurden.

8.1.6.2 Starfish

Die Starfish Methode funktioniert ähnlich wie die Handzettelmethode und wurde bei den Sprints *LSOP 2014-01*, *LSOP 2014-02*, *LSOP 2014-03* und *LSOP 2014-09* eingesetzt. Dabei wurden die Vorschläge und Punkte der Gruppe nicht nur in positiv und negativ unterteilt, sondern in 5 verschiedene Kategorien:

- Start doing - Aktivitäten oder Tätigkeiten, die das Team einbringen möchte um die Produktivität zu steigern.
- Keep doing - Normalerweise sind dies gute Aktivitäten oder Praktiken, die die Team-Mitglieder behalten möchten. Diese Aktivitäten werden bereits angewendet.
- Stop doing - Dies sind die Aktivitäten, die nicht zum Wert / Fortschritt zum Team oder einem Kunden bringen. Diese Aktivitäten verschwenden eventuell Ressourcen bzw. nutzen diese nicht optimal.
- More of - Hierbei handelt es sich um Techniken die vermehrt eingesetzt werden sollten, sodass die Produktivität gesteigert werden kann. Beispielsweise der vermehrte Einsatz von Pair-Programming.
- Less of - Dies sind Tätigkeiten, bei denen der Arbeitsaufwand zur Durchführung solcher Aktivitäten viel kleiner ist als der erbrachte Vorteil. Oder es handelt sich um Aktivitäten, die in der Vergangenheit eingebracht wurden, aber keine generellen Verbesserungen des Verfahrens zeigen.

Zu Beginn gab es eine 5-10 minütige Runde in der sich alle Teammitglieder auf Karten einzelne Punkte aufgeschrieben haben, die während des Sprints aufgefallen sind. Hierbei wurde darauf geachtet, dass sich die Teammitglieder nicht unterhalten oder absprechen. Nach Abschluss stellte jeder reihum seine Punkte vor, indem er die Karte mit Klebeband an die Tafel klebt und die Punkte kurz erläuterte. Dabei wurde darauf geachtet, dass die Person die an der Tafel steht nicht unterbrochen wird oder ein Diskussion über die vorgestellten Punkte geführt wird. Gab es ähnliche oder gleiche Punkte, wurde diese während der Vorstellung zusammengelegt. Im Anschluss wurden vor allem für die Punkte *stop doing* und *less of* diskutiert und Maßnahmen zur Verbesserung diskutiert und beschlossen.

8.1.6.3 Speedboat

Ähnlich wie die Starfish-Methode funktioniert auch die Speedboat-Methode. Grundlegendes Szenario ist dabei, dass sich die Gruppe auf einem Schiff befindet und man in das gelobte Entwicklerland gelangen möchte. Dazu wird das Schiff durch einen Propeller/Wind angetrieben, es gibt Rettungsringe, aber auch Hindernisse wie Felsen/Piraten bzw. Anker, die die Fahrt bzw. Entwicklung verlangsamen. Idee dieses Verfahrens ist es durch ein Szenario, die Beteiligung an der Technik zu verbessern. Hierbei wurden Punkte (Speedboat, Sailboat) in folgende Bereiche unterteilt:

- *Propellors/Wind* (Was bringt uns voran?) - Hier sollten Punkte genannt werden die zur Zeit gut laufen und auch in Zukunft weiter gefördert werden sollten.
- *Life Preserver/Island* (Was kann uns retten oder helfen?) - Was läuft gut und unterstützt die Entwicklung. Was sichert uns ggf. ab.
- *Anchors* (Was hält uns zurück?) - Was blockiert uns oder verlangsamt den Entwicklungsprozess?
- *Rocks/Pirates* (Was könnte zu Problemen führen?) - Welche Dinge sollten wir aus dem Weg schaffen, damit ein ungestörtes Weiterarbeiten gut möglich ist.

Das Verfahren selber wurde analog zur Starfish-Methode durchgeführt.

8.1.7 Prozessunterstützung durch Jira Agile

Um stets einen Überblick über die aktuellen User Stories, Tasks und Bugs zu haben, wurde die Prozessunterstützung JIRA Agile eingesetzt. Diese Prozessunterstürzung bot unter anderem ein virtuelles Scrum-Board an, in dem die aktuellen Aufgaben übersichtlich aufgelistet wurden. Um die einzelnen Schritte des Entwicklungsprozesses abzubilden, wurden die einzelnen Stadien *offen*, *wird ausgeführt*, *wartet auf Test*, *im Test* und *fertig* angelegt. Zu beginn jedes Sprints, befanden sich die einzelnen Aufgaben in dem Status *offen*. Dieser Status signalisierte, dass die Aufgabe zurzeit von keinem Projektgruppenmitglied bearbeitet wird. Sobald ein Gruppenmitglied anfing, eine Aufgabe abzuarbeiten, wurde die entsprechende Teilaufgabe vom Gruppenmitglied in den Status *wird Ausgeführt* geschoben. Wenn die Aufgabe vom Gruppenmitglied abgeschlossen wurde, wurde der Status der Aufgabe auf *wartet auf Test* geändert. Alle Aufgaben die sich im Status *wartet auf Test* befanden, wurden jeweils von anderen Projektgruppenmitgliedern nach unserer *Definition of Done* überprüft. Entsprach die Umsetzung der *Definition of Done* wurde die Aufgabe in den Status *fertig* geändert. Bei Mängeln oder fehlerhafter Umsetzung, wurde die Aufgabe in den Status *offen* zurückgesetzt.

8.1.8 Sprints

Das Projekt LSOP wurde mit Hilfe des Agilen Vorgehensmodell *Scrum* umgesetzt. In diesem wurden insgesamt 13 Sprints mit jeweils einer Länge von drei Wochen durchgeführt. Das Projektteam bestand aus insgesamt zwölf Studenten die in dieser Zeit mehr als 6552 Stunden in das Projekt investiert haben. Im nachfolgenden Abschnitt werden die einzelnen Umsetzungen der Sprints reflektiert. Hierbei werden die Sprintziele die im jeweiligen Sprint erfolgreich umgesetzt werden konnten mit ☑ und Ziele die innerhalb des Sprints nicht umgesetzt werden konnten mit □ gekennzeichnet.

8.1.8.1 Sprint LSOP-2014-01

Im ersten Sprint des Projektes ging es im Kern darum, eine Grundlage für das Projekt zu schaffen. Neben verschiedenen Evaluierungen im Client, Sensor und Auswertungsbereich wurden auch grundlegende Möglichkeiten geschaffen, die z.B. das Verschieben von Anfragen ermöglichten. Bis auf die Evaluierung verschiedener Sensoren und die Ermittlung der relevanten Auswertungen, konnten in diesem Sprint alle weiteren Ziele erreicht werden.

Übersicht der Sprintziele:

☑ Grundlegendes Verschieben von Anfragen ermöglichen

☑ Kommunikation zwischen P2P und Client evaluieren

☑ Evaluierung für Client Applications

☑ Grundlegende, einfache Muster erkennen

☐ Evaluierung der verschiedenen Sensoren auf Basis des Vortrags

☐ Relevante Auswertungen (Trainerbefragung) ermitteln

8.1.8.2 Sprint LSOP-2014-02

Bereits im zweiten Sprint wurde erfolgreich das Grundgerüst der Zwischensprache entwickelt und einzelne Beispiele hierzu dokumentiert. Darüber hinaus wurde die grundlegende Struktur der Entwickler-GUI erstellt und die gesamte Entwicklungsumgebung samt Bamboo, JIRA und Confluence installiert. Die Ermittlung der relevanten Auswertungen konnte auch im zweiten Sprint nicht abgeschlossen werden. Dennoch ergaben sich in diesem Bereich Fortschritte; so konnte ein Termin für ein Interview mit Viktor Skripnik, damals Trainer der U21 Mannschaft und nun Bundesligamannschaft, vereinbart werden.

Übersicht der Sprintziele:

☑ Zwischensprache entwickeln

☑ Beispiele für die Zwischensprache erstellen

☐ Parser für Zwischensprache erstellen

☑ Grundlegende Entwickler-GUI Strukturaufbau

☑ Entwicklungsumgebungen aufsetzen (Bamboo, JIRA, Confluence)

☐ Relevante Auswertungen (Trainerbefragung) ermitteln

8.1.8.3 Sprint LSOP-2014-03

Im dritten Sprint konnten bereits verschiedene Sensoranbieter miteinander verglichen werden. Hierbei stellte sich aber heraus, dass diese Systeme entweder sehr teuer waren oder nicht für den Sporteinsatz geeignet waren. Des Weiteren konnte die Entwickler-GUI erfolgreich mit Odysseus verbunden werden (SOAP-Schnittstelle). Hierdurch war es erstmals möglich, Odysseus-Befehle über die Entwickler-GUI zu einer Odysseus-Instanz zu senden. Darüber hinaus wurde der Parser für die Zwischensprache fertigt gestellt. Im Bereich Recovery konnte ein grobes Konzept erstellt werden, was in weiteren Sprints verfeinert wurde.

Übersicht der Sprintziele:

☑ Sensoranbieter vergleichen

☐ Fußballvereine befragen (noch keine Dokumentation)

☑ Herstellung der Kommunikation zwischen Apps und Odysseus (SOAP)

☑ Parser für Zwischensprache

☐ Anfragepläne für verschiedene Queries erstellen (zwei Anfragepläne fehlen noch)

☐ Recovery Konzept ausarbeiten (Grobkonzept steht, Feintuning fehlt noch)

8.1.8.4 Sprint LSOP-2014-04

Im vierten Sprint konnte mit Hilfe der Anzeige der Spieltminute gezeigt werden, dass die einzelnen Konzepte, die in den vergangenen Sprints erstellt wurden, funktionieren (Durchstich). Des Weiteren konnte durch das Interview mit Viktor Skripnik alle relevanten Auswertungen für die Sportart Fußball erfasst und dokumentiert werden. Eine weitere Befragung die mit dem Trainer Jörg-Uwe Klütz vom BV Cloppenburg durchgeführt wurde, deckt sich mit den Ergebnissen aus dem Interview mit Viktor Skripnik und müssen nur noch dokumentiert werden. Beim Task *Zustandsbehaftete-Operatoren verschieben* konnten bereits erste Teile umgesetzt werden, er konnte aber durch notwendige Abstimmungen mit Timo Michelsen und Marco Grawunder nicht fertig gestellt werden.

Übersicht der Sprintziele:

☑ Grundlegende Struktur der Anwender-App erstellen

☑ Evaluierung der verschiedenen Sensoren auf Basis des Vortrags

☑ Relevante Auswertungen ermitteln

☑ Dokumentation von Spielereignissen anhand des Videos

☑ Durchstich: Spielminute anzeigen

☐ Zustandsbehaftete-Operatoren verschieben

8.1.8.5 Sprint LSOP-2014-05

Im fünften Sprint konnte große Fortschritte im Bereich der Coach App erzielt werden. So wurden bereits Spielspiel, Spieler und Ball korrekt in der Coach App angezeigt. Um Konstanten zu definieren, die für die Auswertung benötigt werden, konnte erfolgreich ein DDC erstellt werden. Mit Hilfe des DDC können somit Konstanten wie z.B. Spielfeldabmessungen, Spielernamen und Torabmessungen gespeichert werden.

Übersicht der Sprintziele:

☐ Weitere Anfragepläne erstellen und bestehende optimieren

☐ Konzept zur Verschiebung von zustandsbehafteten Operatoren

☐ DDC erstellen

☑ Anzeige des Spielfelds inkl. Spielern und Ball in der Coach App

☐ Erstellung der Zwischenpräsentation

☑ Coach App Refactoring

☑ Load-Balancing bugfixen

8.1.8.6 Sprint LSOP-2014-06

Im sechsten Sprint konnten die vorhanden Anfragepläne korrigiert und optimiert werden. Um die fehlenden Sensoren zu kompisieren, konnte das Spiel Eat The Wistle (ETW) erfolgreich an das Odysseus System angebunden werden. Mit Hilfe von ETW können somit Sensoren simuliert werden. Des Weiteren konnte die Schnittstelle zur Coach App erweitert werden und ermöglicht somit den Zugriff auf das DDC von der Coach App aus.

Übersicht der Sprintziele:

☑ Anfragepläne Korrigieren und Optimieren

☑ Intermediate Schema nutzen

☑ X/Y tauschen

☑ Eat The Wistle einbinden

☐ Recovery Grundlagen in Odysseus erstellen

☑ Metadaten / DDC in den Apps verfügbar machen

8.1.8.7 Sprint LSOP-2014-07

Im siebten Sprint konnten die Vorbereitungen für die Zwischenpräsentation erfolgreich abgeschlossen werden. Des Weiteren konnten die ersten Anforderungen für die geplante Monitoring Application erfasst werden. Die Grundlegende Umsetzung des Recovery Konzeptes sowie die Verteilung von Anfragen auf mehrere Peers konnte nicht vollständig umgesetzt werden.

Übersicht der Sprintziele:

☑ Zwischenpräsentation vorbereiten (Abbranchen, Bugfixing, Präsentation, Video)

☐ Grundlegendes Recovery Konzept umsetzen

☐ Zustandsbehaftete Operatoren im Load-Balancing verschieben

☐ Verteilung von Anfragen auf mehrere Peers

☐ Erste Statistiken in der Coach App anzeigen

☐ Anforderungen an Monitoring-Application

8.1.8.8 Sprint LSOP-2014-08

Im achten Sprint wurde das grundlegende Recovery weiter angepasst. Es wurden verschiedene Allokatoren und entsprechende Schnittstellen für die Recovery Strategien erstellt. Im Bereich Coach App wurde die gesamte Anwendung erfolgreich einer Überarbeitung unterzogen. Hierbei wurden verschiedene Bestandteile wie Activities, Fragments und Models neu strukturiert.

Übersicht der Sprintziele:

- ☐ Grundlegendes Recovery abschließen
- ☐ Zustandsbehaftete Operatoren (Zustand kopieren)
- ☐ Load-Balancing um Moving-State Strategy erweitern
- ☑ Coach App Architektur überarbeiten
- ☑ Coach App Statistiken anzeigen
- ☐ Monitoring Application (GUI's erstellen)

8.1.8.9 Sprint LSOP-2014-09

Mit Ende des neunten Sprints wurde damit angefangen, dass Recovery zu optimieren. Da es sich hier um einen hoch komplexen Vorgang handelt, konnte dies bis zum Ende des Sprints nicht fertig gestellt werden. Im Gegensatz dazu konnte erfolgreich die REST-Schnittstelle erweitert werden. Durch diese Erweiterung ist es nun möglich, Anfragen, die von der Coach App gestellt werden, im P2P-Netzwerk nach eigenen Einstellungen zu verteilen.

Übersicht der Sprintziele:

- ☐ Recovery optimieren und robuster machen
- ☑ Verteilung von Anfragen ermöglichen
- ☐ Moving-State Strategy integrieren und robuster machen
- ☐ Zustandskopie von Operatoren ermöglichen
- ☐ Coach App verbessern und um weitere Statistiken erweitern
- ☑ REST Schnittstelle erweitern
- ☑ Eigene Sensordaten aufnehmen (WLAN)
- ☑ Konsole und Ping Map in Monitoring-Application verfügbar machen

8.1.8.10 Sprint LSOP-2014-10

Im zehnten Sprint konnte ein Bug der bei der Verschiebung von Zustandsbehafteten-Operatoren auftritt gefixt werden. Des Weiterhin konnte das Recovery abschließend verbessert werden, sodass die nun ordnungsgemäß funktioniert. Im Bereich der Coach App wurden einige Optimierungen sowie

Dokumentationen durchgeführt. Des Weiteren wurde die Unterstützung mehrerer Sportarten nun vollständig in die Coach App integriert.

Übersicht der Sprintziele:

- ☐ Queries optimieren
- ☑ Robustheit des Recovery verbessern
- ☐ Moving-State Strategy
- ☐ Fehlerfälle abfangen und beschreiben
- ☑ Bugfixing für Zustandsverschiebung (JoinTIPO)
- ☑ Coach App
- ☑ Mehrere Sportarten architektonisch unterstützen
- ☑ Optimierung und Dokumentation

8.1.8.11 Sprint LSOP-2015-01

Im zehnten Sprint wurde bereits die Sportunabhängigkeit in der Coach App vollständig umgesetzt. Durch diese Änderung wurden weitere Anpassungen im LSOP Service sowie in Odysseus notwendig, die erfolgreich durchgeführt werden konnten. Des weiteren wurde versucht Sensordaten über die erarbeitete WLAN-Technologie aufzuzeichnen. Hierbei konnten zwar Daten erfasst werden, aber die Auswertung sowie Dokumention des vorgehen konnte nicht mehr abgeschlossen werden.

Übersicht der Sprintziele:

- ☐ Active-Standby implementieren
- ☐ Vorhandene Anfragen optimieren und Fehler bereinigen
- ☐ Sensordaten mit der WLAN Technik aufnehmen
- ☑ Sportartenunabhängigkeit auf Odysseus-Seite und im LSOP Service
- ☐ Logging in der Monitoring-Application
- ☐ Sprintstatistik in Coach App anzeigen
- ☑ Zeichnen in der Coach App ermöglichen
- ☐ Spieler in der Coach App vergleichen können (keine neuen Anfragen)

8.1.8.12 Sprint LSOP-2015-02

Im elften Sprint wurde damit begonnen, das Active-Standby Konzept umzusetzen. Durch mehrere Probleme, die bei der Implementierung aufgetreten sind, konnte dieses Aufgabe nicht abgeschlossen werden. Ein Erfolg konnte bei der Optimierung der DDC Verteilung verzeichnet werden. Diese wurde erfolgreich optimiert und verursacht nun eine deutlich geringere Belastung für den jeweiligen Peer.

Übersicht der Sprintziele:

- ☐ Active-Standby
- ☑ Optimierung der DDC Verteilung
- ☑ Recovery Fehlerfälle behandeln
- ☐ Moving-State-Strategy im Load-Balancing abschließen
- ☐ Bugfixes

8.1.8.13 Sprint LSOP-2015-03

Im zwölften Sprint sollten eigentlich die Evaluationen durchgeführt werden. Zu Beginn des Sprints wurde aber festgestellt, dass es an mehreren Stellen massive Probleme gibt, die dazu führten, dass eine Evaluation nicht möglich ist. Somit wurde der zwölfte Sprint dafür genutzt, die Evaluation für den nächsten Sprint vorzubereiten. Des Weiteren konnte in anderen Bereichen bereits erfolgreich mit der Abschlussdokumentation angefangen werden.

Übersicht der Sprintziele:

- ☐ Durchführung der Evaluation (Latenz und Durchsatz)
- ☐ Weitere Bugfixes, sodass alle Evaluationen durchgeführt werden können
- ☑ Beginn der Abschlussdokumentation

8.1.8.14 Sprint LSOP-2015-Final

Im dreizehnten Sprint wurde die Evaluation in allen Bereichen abgeschlossen. Hierfür gab es für alle Bereiche Funktionstests sowie für die Anfragen, das Load-Balancing und das Recovery Leistungstests. Diese wurden ausgewertet und entsprechend dokumentiert. Ansonsten wurde die Projektdokumentation fortgeführt und abgeschlossen.

Übersicht der Sprintziele:

- ☑ Letzte Bugfixes
- ☑ Evaluation (Funktionstests)
- ☑ Evaluation (Latenz und Durchsatz)
- ☑ Abschlussdokumentation geschrieben
- ☑ Abschlussdokumentation Korrektur gelesen

8.2 Rollen

Zu Beginn der Projektgruppe wurden wichtige, zentrale Projektrollen festgelegt. Diese Projektrollen wurden während der gesamten Projektdauer beibehalten und stellten, abhängig von der Projektphase, einen unterschiedlichen Aufwand dar.

8.2.1 Projektmanagement

Thomas Prünie, Tobias Brandt

Das Projektmanagement stellte während der gesamten Projektphase eine wichtige Rolle dar, da die Verantwortlichen entsprechende Deadlines festgelegt haben. Darüber hinaus wurde der Gesamtfortschritt unabhängig von den Sprints überwacht, sodass Verzögerungen von Teilaufgaben möglichst schnell zurückgemeldet wurden und somit entsprechende Reaktionen durchgeführt werden konnten. Neben dem Meilensteinplan wurde auch das Projekttagebuch kontinuierlich gepflegt und bei wichtigen Ereignissen (beispielsweise die erste eigene Aufnahme von Sensordaten) entsprechende Einträge in diesem verfasst.

8.2.2 Scrum

Chris Tönjes-Deye, Pascal Schmedt

Bei der Durchführung dieser Projektrolle war es vor allem entscheidend, dass der Scrum-Prozess während des Projektes eingehalten wurde. Dies beinhaltete die Durchführung von Reviews, Retrospektive und Planning Meetings. Darüber hinaus war es ihre Aufgabe zu überwachen, dass das Product Backlog mit entsprechenden neuen Aufgaben kontinuierlich neu gefüllt wurde. Aber auch die Beseitigung von Hindernissen, die sich negativ auf die Gruppe und deren Produktivität auswirken, war eine zentrale Aufgabe. Dies waren zum Beispiel Probleme mit dem WLAN.

8.2.3 Konfiguration

Marc Preuschaft

Während der gesamten Dauer des Projektes war auch die Rolle des Konfigurationsmanagement von wichtiger Bedeutung. Hierbei war es vor allem entscheidend, dass sämtliche Server-Anwendungen, wie Jira, Confluence, Fisheye und Bamboo zuverlässig und korrekt funktionierten. Diese Rolle wurde vor allem wichtig, als die bestehenden Systeme im OFFIS auf eigene Server in der Abteilung Rechner- und Netzbetrieb Informatik (ARBI) migriert wurden, um wieder eine hohe Zuverlässigkeit der Systeme gewährleisten zu können.

8.2.4 Usability

Torben Pehlke, Marc Wilken

Die Projektrolle bezogen auf die Usability war vor allem im Bereich der Coach App entscheidend, da die Bedienung der App durch den Trainer möglichst intuitiv gestaltet werden sollte. In die Ge-

staltung der App sind auch die Ideen der Trainer, die im Abschnitt 8.3 genauer beschrieben werden, eingeflossen. Bei allen anderen entwickelten Systemen war die Berücksichtigung der Usability nicht von entscheidender Bedeutung.

8.2.5 Qualitätsmanagement

Thore Stratmann, Simon Küspert

Bei der Ausübung der Rolle des Qualitätsmanagements, wurde vor allem die Code-Qualität und die Entwicklung von Testfällen in den Vordergrund gestellt. Die Durchführung von Tests in Odysseus wurde hierbei eher weniger betrachtet, da es hier nahezu unmöglich ist gekapselte Unit-Tests zu schreiben. Auch bei der Entwicklung der Coach App stellte sich heraus, dass eine effektive Nutzung von Unit-Tests nicht möglich ist, da hierfür in Android eine entsprechende Umgebung in Form eines virtuellen Gerätes gestartet werden muss und somit die Ausführung der Tests eine zu große Dauer benötigt. Für eine kontinuierlich hohe Qualität der Software wurden alle Systeme in das Continious Integration System Bamboo eingebunden, sodass jederzeit sichtbar war, welche Systeme aktuell kompilierbar sind.

8.2.6 Sensorik

Michael Brand

Auch der Bereich der Sensorik stellte eine zentrale Aufgabe dieser Projektgruppe dar. Zu Beginn wurde ausschließlich mit den vorgegebenen Daten der DEBS Grand Challenge 2013 gearbeitet. Allerdings war es auch die Aufgabe der Projektgruppe, dass eigene Datenströme eines Spiels aufgezeichnet werden, sodass hierfür entsprechende Sensoren benötigt wurden. Zunächst wurden alle am Markt bekannten Systeme verglichen, allerdings stellte sich heraus, dass die Sensoren das Budget der Projektgruppe übersteigen. Aus diesem Grund wurden mit Hilfe von Smartphones und den darin enthaltenen GPS Sensoren eigene Systeme entwickelt und die Daten hiermit aufgezeichnet.

8.2.7 Dokumentation

Carsten Cordes

Vor allem zum Ende der Projektgruppe wurde diese Projektrolle immer wichtiger, da alle Inhalte koordiniert durch die gesamte Gruppe dokumentiert werden mussten. Aus diesem Grund wurden entsprechende Richtlinien entworfen, sodass die gesamte Dokumentation möglichst einheitlich ist. Darüber hinaus wurden in Jira Tasks erstellt, sodass die Dokumentation auch in kurzer Zeit möglichst gut koordiniert werden konnte. Zusätzlich war es die Aufgabe des Verantworlichen zu überwachen, dass die Dokumentationsstruktur im Wiki-System Confluence während der gesamten Projektdauer eingehalten wird.

8.2.8 Social Team Event Manager

Marc Wilken

Durch den Social Team Event Manager wurden während der gesamten Projektphase verschiedene Veranstaltungen organisiert. Dazu gehörten unter anderem Grillen, BBQ-Donut in Wilhelmshaven, Kramermarkt, Weihnachtsmarkt und das projektgruppenübergreifende Boule-Tunier. Durch diese Events wurde einerseits der Teamgeist gefördert und andererseits der Spaßfaktor mit in die Projektgruppe eingebracht.

8.3 Trainerinterviews

Um einen Einblick in die für einen Trainer an der Seitenlinie eines Fußballspiels wichtigen Aspekte zu bekommen, wurden Trainerinterviews durchgeführt. Nach einer Einführung und Beschreibung der Projektgruppe wurden verschiedene Fragen aus speziellen Fragekategorien gestellt. Diese werden im Folgenden kurz beschrieben.

8.3.1 Zielsetzung

Die Zielsetzung dieses Interviews war es, erstens durch offene Fragen die Aspekte, die nicht trivial für außenstehende zu sammeln sind, möglichst vollständig zu erfassen. Zweitens sollten, durch eine reine Datenabfrage mit Aspekten bestehender Systeme, mögliche Defizite am Markt aufgedeckt werden, die dann in der entwickelten Applikation umgesetzt werden können. Anschließend wurden bestehende Umsetzungen diskutiert, um ein Meinungsbild erfassen zu können und mögliche Schwachstellen und Vorteile bestehender Systeme auswertbar zu machen.

Um die Akzeptanz von Technologie im Bereich Livesportanalyse zu erfassen, wurden verschiedene Fragen gestellt, die das Verhalten des Trainers im Spiel- und Trainingskontext darstellen sollten. Abschließend wurde eine Frage zur Einschätzung über die Attraktivität eines Einsatzes des erarbeiteten Systems und die Unterstützung von Trainerentscheidungen während des Spiels gestellt.

8.3.2 Fragebogen

Der Fragebogen, der zur Unterstützung der Interviews genutzt wurde, wird hier aufgelistet. Um die Intention einiger Fragen zu erläutern, sind in Klammern Kommentare zu der Frage angegeben.

8.3.2.1 Offene Fragen

Innerhalb dieses Frageblocks werden initiale Fragen gestellt, um die Vorkenntnisse des Trainers einschätzen zu können und in das Thema einzuleiten. Um die Perspektive des Trainers nicht einzuschränken, werden keine Beispiele von möglichen spielunterstützenden Technologien gezeigt.

- Kennen Sie bereits Technologien, die Trainer bei ihren Entscheidungen unterstützen können? (Initiale Wissensabfrage, um Vorkenntnisse und Erfahrungen zu erfragen)
  - Falls ja,
    * Welche Technologien kennen Sie und haben Sie diese bereits eingesetzt?
    * Hat diese Technologie Sie bei Trainerentscheidungen unterstützt?
    * Welche Vor- und Nachteile hat diese Technologie?

- Wie möchten Sie am Spielfeldrand unterstützt werden? (Offene Anfangsfrage ohne große Vorkenntnisse. Vielleicht können hier relevante Gebiete erhoben werden, an die vorher nicht gedacht wurde. Außerdem wird der Trainer hier Bereiche nennen, die ihm wichtig sind und in denen er ggf. Defizite erkennt.)

8.3.2.2 Datenabfrage

Dieser Bereich des Fragebogens soll darüber Auskunft geben, an welchen konkreten Daten die Trainer interessiert sind. Dazu werden unterschiedliche Statistiken genannt, zu denen die Trainer Einschätzungen geben können.

- Gibt es Daten, die Sie dringend benötigen, die aber aus Ihrer Perspektive nicht erfasst werden können?
- Welche Daten könnten Ihnen bei strategischen Entscheidungen helfen?
- Könnte Ihnen dies helfen? (Eine Heatmap für Freiräume wird gezeigt.)
- Spielerdaten
  - Diese Daten sollen durchgegangen werden und zusammen mit dem Trainer auf Vollständigkeit und Sinnhaftigkeit im Spielkontext geprüft werden.
  - Heatmaps -> Aufenthaltsorte der Spieler
  - Welche Erkenntnisse können aus Laufwegen gewonnen werden?
  - Echtzeitdaten -> freie Flächen anzeigen
  - Analysedaten Spielerprofil
    - Welche Spieler stehen frei (Freiraum/Zeit frei)
    - Zeit mit dem Ball
    - Zeit mit dem Ball/Pässe
    - Geschwindigkeit
    - Beschleunigung
    - Schusshärte (-> Geschwindigkeit des Balls?)
    - Angekommene Pässe/Pässe insgesamt
    - Laufstrecke
    - Ballberührungen
  - Analysedaten mehrere Spieler
    - Abwehrketten
    - Abstand zwischen Ketten, Breite der Ketten(Durchlässigkeit)
    - Positionierung bei Standards(Abweichungen), Manndeckung
    - mögliche Passstationen
    - optimale Deckungs-/Laufwege
- Trainingsdaten
  - Vergleich von Laufwegen/Geschwindigkeiten von Spieler

  - Vergleich von „Geisterspielern“ mit dem aktuellen Spieler (oder vergangene Trainingseinheiten)
  - Vergleich mit optimalen Wegen möglich
- Rohdaten oder aufbereitete Daten

8.3.2.3 Verhaltensfragen

- Würden Sie die Technologie einsetzen oder der Co-Trainer?
- Könnten Sie sich verstellen eine Technologie zur Unterstützung ihrer Trainer-Entscheidungen zu verwenden?
- Wie realistisch schätzen Sie die Möglichkeit des Technologieeinsatzes in Spielsituationen/Trainingssituationen ein? (Trainings- oder Spielsituation?)

8.3.2.4 Einstellungs- und Meinungsfragen

- Denken Sie, Spieler könnten anhand ihrer Geschwindigkeit/Beschleunigung/Laufstrecke etc. bewertet werden und auf dieser Grundlage Empfehlungen, z.B. für eine Auswechslung gegeben werden?
- Schwellwerte für Empfehlungen
- Diskussion von beispielhaften Umsetzungen (Anregungen/Ideen)

8.3.2.5 Abschließende Frage

- Denken Sie, nachdem Sie einiges über den Einsatz unseres Systems erfahren haben, dass Ihnen dieses System bei Trainer-Entscheidungen helfen kann?

8.3.3 Ergebnisse

Befragt wurden Viktor Skripnik, Trainer bei Werder Bremen, damals Trainer bei Werder Bremen II und Jörg-Uwe Klütz, Trainer bei BV Cloppenburg. Beide Mannschaften spielten zur Zeit der Befragung in der Regionalliga Nord.

8.3.3.1 Allgemeines zu den Trainern

Beide Trainer sind recht unterschiedlich in ihrer Herangehensweise.

Skripnik sieht sich eher als Menschenführer, der großen Wert auf die mentale Einstellung seiner Spieler legt und für den Statistiken eher zweitrangig sind. Im Interview wurde dies dadurch deutlich, dass er bei fast allen Fragen immer darauf zurückkam, wie wichtig die Motivation der Spieler sei.

Klütz hingegen ging mehr auf taktische Aspekte ein und kann sich besser mit Statistiken anfreunden, auch wenn er durch den kleinen Etat vom Verein kaum Gelegenheit hat, diese zu nutzen. Doch auch er hat betont, dass bei allem auch immer die Einstellung der Spieler stimmen muss.

8.3.3.2 Protokoll

Bekannte Technologien

Skripnik sind unterschiedliche Technologien bekannt. So wird bei ihm die Videoanalyse eingesetzt, auch bereits in der Halbzeitpause. Außerdem nutzt der Verein GPS Sender an allen Spielern in Freundschaftsspielen oder manchmal auch in Regionalliga-Spielen mit Erlaubnis vom Schiedsrichter. Dies ist nützlich, um Trainingsanteile von Spielern zu ermitteln, da mit den Daten Stärken und Schwächen aufgedeckt werden können. Im Hinblick auf mögliche Statistiken erhält der Verein Anregungen vom DFB.

Klütz hat weniger Zugriff auf solche Technologien. Selten wird die Videoanalyse eingesetzt. Dies vor allem, um Erklärungen zu verdeutlichen und Eindrücke, die während des Spiels fehlerhaft aufgenommen wurden, zu erkennen. Außerdem hat sich der Trainer privat eine GPS-Uhr gekauft, um die Laufleistung einzelner Spieler tracken zu können.

Spielerdaten

Skripnik stimmt der Sinnhaftigkeit der vorgestellten Spielerdaten aus dem Fragebogen grundsätzlich zu. Welche dieser Daten jedoch wichtig sind, hinge von der Präferenz des Cheftrainers ab. Laut dem Trainer seien bei Laufdaten die Anforderungen für die Spieler zudem unterschiedlich: Stürmer müssten eher viele Sprints anziehen, während Außenverteidiger mehr lange Läufe absolvieren müssten. Es sei vor allem wichtig, dass die Spieler intelligente Laufwege gehen und nicht, dass sie viel laufen. Die Heatmap sei nützlich, um Verbesserungspotential im Stellungsspiel sehen zu können. Interessant seien zudem die Anzahl der Angriffe von links, rechts und der Mitte. Bei Pässen könne man auf eine erfolgreiche Passquote oder mit mehr Risiko spielen. Hierbei brachte Skripnik ein, dass der Trainer der ukrainischen Nationalmannschaft ein Bewertungssystem nutzte. So gab es z.B. für Pässe nach hinten 0,3 Punkte, für Querpässe 0,5 und für Pässe nach vorne unter Bedrängnis 3 Punkte, sofern diese auch ankamen. Daraus ergibt sich ein Score, der Aufschluss darüber gibt, ob ein Spieler die Tendenz hat, Sicherheits- oder Risikopässe zu spielen. Dabei ist es jedoch sehr wichtig, den Kontext zu anderen Spielern zu beachten, also zu beobachten, wie sich die Spieler zueinander bewegen. Zeigt man dem Spieler zum Beispiel seine Pass-Statistik und sagt ihm, dass er zu wenig Pässe gespielt hat, fängt der Spieler im nächsten Spiel an viel quer- oder zurückzupassen, nur um die Statistik besser aussehen zu lassen.

Klütz findet die Vogelperspektive sehr sinnvoll. Darin seien Stellungsfehler gut zu erkennen, deshalb ginge er auch selbst in Freundschaftsspielen manchmal auf die Tribüne, um zu sehen, warum bestimmte Fehler passieren und um Laufwege zu erkennen. Zudem kann man von dort besser beobachten, wo ein Spieler leicht angespielt werden kann. Bei Angriffen seiner Mannschaft achtet er dabei zudem eher auf das Stellungsspiel seiner Abwehr, um bereit zu sein, einen Konter abzufangen. Außerdem sehe man aus der Vogelperspektive die Positionierung in Standardsituationen besser. Die Heatmap hingegen lenke eventuell zu schnell ab und sei deshalb eher etwas für den Co-Trainer. Jedoch sei sie trotzdem sinnvoll, um einige Verhaltensweisen der Spieler besser sehen zu können. So sei es zum Beispiel möglich, damit zu sehen, ob der Außenverteidiger die ganze Zeit nur seine Linie hoch und runter läuft oder gegebenenfalls auch darauf achtet, ob der gegnerische Stürmer auf die Mitte oder die andere Seite ausweicht und sich der Abwehrspieler dann etwas an ihm orientiert. Freiräume für die Spieler anzuzeigen ginge hingegen vermutlich etwas zu weit.

Während der Saison-Vorbereitung seien Werte wie Geschwindigkeit, Beschleunigung, Sprungkraft, Ausdauer und Schusshärte wichtig. Das von Skripnik angesprochene Passbewertungssystem der ukrainischen Nationalmannschaft hält Klütz nicht für sinnvoll, man solle eher angekommene Pässe und Fehlpässe zu Rate ziehen. Diese könnten auch nach den drei Ketten (Sturm, Mittelfeld und Abwehr) aufgeteilt werden. Für diese Ketten sei es zudem sehr sinnvoll, die Abstände zueinander (von Kette zu Kette), die Breite und die Durchlässigkeit zu prüfen. Der Abstand der Linie sollte immer recht gleich sein, sodass der Zwischenlinienraum nicht zu groß wird. Dies sei der wichtigste Bereich im Spiel. Der Abstand der Abwehr zum Torwart sei wichtig für die Bälle hinter der Abwehr.

Die Ballkontakte seien interessant, um damit im Zusammenhang zu schauen, über welche Seite Angriffe gespielt wurden. Sinnvoll seien zudem auch Ballkontakte pro Pass oder Zeit am Ball pro Pass. Dies zeige, ob der Spieler schon vorher weiß, was er mit dem Ball vorhat oder ob er diesen zunächst stoppen muss, um zu schauen, wohin als nächstes gespielt werden muss. Der Deckungsschatten sei ebenfalls sinnvoll und in diesem Zusammenhang auch, ob der Spieler im Schatten bereit ist, sich freizulaufen oder ob er sich lieber hinter dem Gegner verstecke.

Außerdem erwähnte Klütz, dass er keinen Unterschied zwischen Trainings- oder Spieldaten mache, da Spielsituationen im Training durchgespielt würden.

Verhaltensfragen

Skripnik ist der Meinung, dass der Co-Trainer die Technologie einsetzen sollte. Dieser muss dabei die Spielphilosophie des Trainers vertreten und wissen, was diesem wichtig ist. Der Co-Trainer könne die Ergebnisse nochmals filtern und dem Trainer die wichtigen Erkenntnisse mitteilen. Bei der Frage, ob er die Technologie zur Unterstützung einsetzen würde, war er zwiegespalten. Er selbst würde sie nicht einsetzen, dies lieber dem Co-Trainer oder Experten überlassen. Er setze Statistiken lieber nach dem Spiel zur Analyse ein als währenddessen.

Klütz würde die Entscheidungen nicht von der Technologie abhängig machen, aber unterstützend zu Rate ziehen. Während des Spiels sollte jedoch nur der Co-Trainer oder eventuell ein Dritter, der sich ausschließlich darum kümmert, die Daten bekommen. Er würde solche Technologien einsetzen, gerade in Freundschaftsspielen. Veränderungen an der Taktik, Aufstellung, etc. könnten in der Halbzeit wesentlich besser vorgenommen werden als während des Spiels.

8.3.4 Fazit

Generell ist festzuhalten, dass es viele verschiedene Trainertypen gibt, die verschiedene Schwerpunkte haben, wodurch die Coach App anpassbar an die Wünsche des Trainers sein sollte. In den Interviews wurden die vorgestellten Statistiken bestätigt, mit der Anmerkung das diese Statistiken immer im Kontext analysiert werden müssen. Interessant ist der Ansatz die Geschwindigkeit des Balles zu verfolgen, um zu sehen ob es viele Standzeiten gibt und wie die Anzahl der Angriffe über links, rechts oder durch die Mitte aussieht. Viel Wert wird ebenfalls auf das Zusammenspiel der Mannschaftsteile sowie die Positionierung bei Standards gelegt. Außerdem sind sich beide einig, dass sie das Gerät selbst nicht im Blick haben können um nicht zu sehr vom Spielfeld abgelenkt zu werden. Dies hat zur Folge, dass die Coach App durchaus ausführlichere Statistiken bereitstellen könnte, da sich eine Person speziell damit beschäftigt und nicht alles auf einen Blick zu erkennen sein muss.

8.4 Projekttagebuch

Um das Voranschreiten des Projektes zu dokumentieren, wurden in unregelmäßigen Abständen Einträge für ein Projekttagebuch erstellt. Das Verfassen von Tagebucheinträgen ist eine der Aufgaben des Projektmanagements. Im Folgenden werden die Einträge ohne die angehängten Bilder wiedergegeben.

8.4.1 Projekt gestartet

16.05.2014

Die erste offizielle Sitzung nach der Seminarphase startete das Projekt. Von Timo wurden die groben Pläne und Vorstellungen zur Architektur sowie die Ziele des Projektes vorgestellt. Dabei wurde hervorgehoben, dass das Team viel Spielraum bei Entscheidungen hat und nur der grobe Rahmen vorgegeben ist. Es wurden zudem organisatorische Dinge geklärt, wozu unter anderem die Rollenverteilung gehört.

Um das Projekt ins Rollen zu bringen, wurde nach der Sitzung überlegt, welche Teile besonders wichtig sind. Dabei stellte sich heraus, dass die erste Hauptbaustelle der P2P-Bereich ist. Die Formulierung der Anfragen wird zwar ebenfalls aufwendig, jedoch wurde es als sinnvoller erachtet, erstmal im P2P-Bereich zu beginnen. Auch die GUI soll erst später begonnen werden, wenn man beginnt, die Anfragen zu formulieren und somit feststellen kann, was überhaupt visualisiert werden muss.

Im P2P-Bereich ist es zunächst von Nöten, dass wir in Odysseus Fachwissen erlangen, um dort solche Aufgaben angehen zu können. Wichtige Aufgaben werden das Load-Balancing sowie das Recovery. Beim Load-Balancing wird die Möglichkeit, Operatoren von Peer zu Peer verschieben zu können, ein zentraler Bestandteil der Aufgaben. Es wurde zudem darüber nachgedacht, ob es sinnvoll ist, den Datenstrom des Spiels zusätzlich in einer Datenbank abzuspeichern. Somit es möglich, Anfragen, die der Benutzer (i.d.R. der Trainer) erst später stellt, auch noch gut beantworten zu können. Hier muss darüber nachgedacht werden, was zwischengespeichert werden und was eventuell wie aggregiert werden kann.

In der nächsten Woche soll weiter über die Möglichen Aufgaben und Ansatzpunkte nachgedacht werden. Unten ist ein Bild der Aufzeichnungen des Brainstormings zu sehen.

8.4.2 Erster Sprint gestartet

03.06.2014

Am 30.05.2014 wurde der erste Sprint des Projekts gestartet. Die Länge des ersten Sprints wurde auf zwei Wochen (und damit eine Woche kürzer als für die folgenden Sprints geplant) festgelegt, um mögliche Schätzungenauigkeiten zu Beginn des Projektes zu vermindern. Das Projekt beginnt mit Aufgaben in unterschiedlichen Teilbereichen, den Epics, die vorher in der Planung der Aufgaben festgelegt wurden. So wurde zu Beginn sowohl die Entwickler-GUI, das Load-Balancing, die Sensorik, die Client-P2P-Kommunikation und das Data Mining mit ersten Aufgaben in Angriff genommen. Durch die Streuung in verschiedene Bereiche ist es möglich, Aufgaben für alle Projektmitglieder zu haben, ohne dass zu viele gegenseitige Abhängigkeiten entstehen und ohne dass man sich (z.B. im Code) zu sehr in die Quere kommt. Das Epic für das Recovery wurde noch nicht begonnen, da bereits

so genug Arbeit vorhanden war und wir uns Synergien mit dem Load-Balancing erhoffen. Die grobe Zeitplanung wurde im Projektplan festgehalten und vorgestellt und wir hier in einem der nächsten Einträge beschrieben.

8.4.3 Roadmap und Architekturplan

23.06.2014

Das Projekt läuft, der erste Sprint wurde erfolgreich beendet und der zweite Sprint gestartet. Da einige Tasks, bei denen auf Antworten von Trainern oder Unternehmen gewartet werden muss, nicht beendet werden konnten, wurde ein entsprechender neuer Tasktyp erstellt. Somit muss der Sprint nicht scheitern, wenn auf nicht beeinflussbare Sachen gewartet werden muss.

Während des laufenden Sprints sind einige technische Dinge aufgefallen, die wir vorher nur teilweise beachtet oder als nicht so gravierend eingeschätzt hatten. So wird der CommunicationLayer der Architektur nun weniger Aufgaben übernehmen, da die Arbeit mit Simple Object Access Protocol (SOAP) bzw. die Netzwerkkommunikation allgemein in Android und Standard-Java Applikationen zu unterschiedlich ist.

Um den Plan der Projektgruppe übersichtlicher darzustellen als in einer Projektdatei, habe ich zudem eine Roadmap angelegt. Sie zeigt die Planung, wann welche Ziele erreicht werden sollen. Der Plan muss noch mit der Gruppe abgesprochen werden und könnte z.B. im Projektraum aufgehängt werden, um den Blick auf das „Große Ganze“ nicht zu verlieren.

Als nächsten Meilenstein habe ich vorerst für Anfang August angegeben, dass erste Ergebnisse von Anfragen auf der App zu sehen sind. Das beinhaltet, dass die App mit dem Netzwerk kommunizieren kann, eine Anfrage schicken kann und Ergebnisse zurück bekommt. Wie diese angezeigt werden, spielt dabei bei diesem Meilenstein noch keine Rolle – die ersten Visualisierungen und Aufbereitungen sind erst für Mitte November vorgesehen.

Das Load-Balancing ist in den Plan Anfang Oktober als „fertig“ angegeben, was bedeuten soll, dass das Netzwerk selbstständig, nach einer sinnvollen Strategie Load-Balancing betreibt, wie es im Ziel der Projektgruppe vorgesehen ist.

Die Fortschritte in der Entwickler-GUI sind absichtlich nicht mit aufgenommen, da diese nur für interne Zwecke dienen, jedoch nicht das eigentliche Ziel des Projektes darstellen. Die Roadmap dient also auch dazu, den Fokus auf das Wesentliche zu behalten.

8.4.4 Zwischenpräsentation:Roadmap angepasst

08.07.2014

Die vorgestellte Roadmap wurde ein wenig angepasst. Die Zeit zwischen den letzten Tests (rot) mit den Sensoren und dem Projektende wurde leicht um zwei Wochen vergrößert, um mögliche Korrekturen beim Zusammenspiel mit allen Komponenten eher zu ermöglichen. Außerdem wurde die Zwischenpräsentation, die am 23.09.2014 stattfindet, eingetragen.

8.4.5 Projektgruppen-Boule-Turnier

24.07.2014

Am Mittwoch fand das Boule-Turnier der Projektgruppen statt. Unsere Projektgruppe (PG) war mit zwei Mannschaften angetreten, das Ziel fest im Blick: Wir holen den Pokal. Nach hochprofessionellen Spielen und einem atemberaubend spannenden Finale hat Team Klickers 1 gewonnen und den Pokal für die Abteilung geholt. Team Klickers 2 wurde 5. Bei insgesamt 13 Mannschaften ein sehr gutes Ergebnis. Eine gute Gruppe lässt sich auch an dieser starken Mannschaftsleistung erkennen - allein die Tatsache, dass alle (bis auf krankheitsbedingt Thore) dabei waren, zeigt, wie motiviert unsere PG ist.

8.4.6 Durchstich geglückt

26.08.2014

Ein großes Ziel des letzten Sprints und der Bemühungen war es, einen Durchstich zu schaffen. Durch die Größe des Projekts und den vielen Baustellen, an denen gleichzeitig gearbeitet wird, war es schwierig zu sagen, ob überhaupt alle in die gleiche Richtung arbeiten und ob die unterschiedlichen Lösungen gut miteinander funktionieren würden. Deshalb war eines der Ziele des letzten (vierten) Sprints, dass ein Durchstich gemacht wird. Anhand einer einfachen Statistik sollte gezeigt werden, dass Odysseus P2P, die Anfragen, SportsQL, REST, die Sockets und die Apps (also Android und die Developer GUI) zusammenarbeiten können.

Als einfache „Statistik“ wurde dazu die Spielminute gewählt. Um diese auf der App anzuzeigen benötigt es alle Teilbereiche, bei denen wir am Entwickeln sind. (Lediglich das ebenfalls in Arbeit befindliche Load-Balancing wird eben nur berücksichtigt, wenn es benötigt wird. Kann aber auch manuell angestoßen werden, um es zu testen.) Und siehe da: Es funktioniert, die Teilbereiche funktionieren auch als großes Ganzes. Auf dieser Basis können wir also weiter arbeiten und weitere Statistiken hinzufügen.

Und auch in anderen Bereichen gibt es sichtbare Fortschritte. Das Load-Balancing funktioniert in einer ersten Version mit einfacher Strategie und einfachem Allokator, sogar eine Fehlerbehandlung für evtl. wegfallende Peers während des Vorgangs ist bereits implementiert. Das Recovery kommt ebenfalls voran, die Implementierung steht jedoch noch aus. Dafür wurde bereits ein umfangreiches Konzept erarbeitet. Die Coach App bekommt in diesem Sprint (der 5.) eine bessere Architektur/ Struktur und die Anzeige des Spielfeldes steht kurz bevor.

An „Veröffentlichungen“ wird ebenfalls weiter gearbeitet. So arbeiten weiterhin vier Mitglieder der Gruppe an einem Paper für eine Konferenz. Da der Einreichungstermin für die zuerst angestrebte Konferenz jedoch früher war als wir dachten (wir hatten uns um einen Monat versehen), ist unser Ziel nun die Association for Computing Machinery (ACM)-Symposium on Applied Computing (SAC) 2015. Des Weiteren ist ein Paper in Arbeit, dass als Demo-Paper bei der Fachtagung „Datenbanksysteme für Business, Technologie und Web (BTW)“2015 eingereicht werden soll. Und in Kürze steht auch die Zwischenpräsentation an, bei der unser aktueller Stand präsentiert werden soll (23.09.2014).

Die Sensoren bereiten uns derzeit jedoch Sorgen. Da es keinen Anbieter gibt, der für uns im bezahlbaren Rahmen Sensoren für Sport anbietet, werden wir vorerst bei der Verwendung unseren Datensatzes aus der ACM DEBS Grand Challenge 2013 bleiben. Eine mögliche Alternative ist es, „Stand-

ortdaten“ aus einem Open-Source-Fußballspiel zu extrahieren, sodass man ein auf dem Computer gespieltes Spiel analysieren könnte.

8.4.7 Erstes Video

03.09.2014

Seit dem heutigen Tag können wir die Spieler, den Ball sowie die Spielzeit in unserer Android-APP darstellen.

8.4.8 Fortschritt in der Coach App

26.09.2014

Seit dem letzten Eintrag hat sich einiges in der Coach App getan.

Neben einem komplett neuen Design für den Auswahlbildschirm ist auch die Darstellung des Spiels angepasst worden. Die Spieler der einzelnen Mannschaften werden nun entsprechend ihrer „Vereinsfarben“ auf dem Spielfeld angezeigt und auch der Schiedsrichter ist jetzt eindeutig von den Spielern zu unterscheiden. Alle Elemente werden mittlerweile auch als Kreise dargestellt und haben eine Nummer, sodass jeder Spieler besser zuzuordnen ist. Damit die Farben nicht zu langweilig werden, obliegt es dem Nutzer der Applikation, die Farben für einzelne Elemente zu ändern. Ein erstes App Icon ist auch gefunden, nachdem in einer der vergangenen Sitzungen festgelegt wurde, dass das Projekt auf den Namen „Herakles“ getauft werden soll. „Herakles“ deshalb, da er laut griechischer Mythologie der Gott der Athletik und allgemein des Sports ist. Das ist doch mal ein passender Name.

Davon abgesehen wurde weiterhin kräftig an der App gearbeitet. So wurde ebenfalls umgesetzt, dass die Daten der einzelnen Spieler auf der damals noch leeren „Statistik“ -Seite zu sehen sind.

Ein weiteres Highlight ist momentan die Selektion von Spieler. Durch Anhaken eines Spielers in der Statistik-Ansicht oder durch Berührung auf dem Spielfeld wird dieser farblich von den anderen Spielern abgehoben. Dadurch ist es möglich, einen Spieler genauer zu beobachten und ihn nicht aus den Augen zu verlieren.

8.4.9 Zwischenpräsentation

23.10.2014

Sie war ein großes Zwischenziel in unserem Projekt: Am Dienstag, den 21.10.2014 haben wir unsere Zwischenpräsentation gehalten. Vor Mitarbeitern der Abteilung Informationssysteme und dem OFFIS aus dem Bereich Gesundheit haben Carsten Cordes, Marc Preuschaft und Tobias Brandt unseren Projektstand vorgestellt. Es wurden unsere Konzepte und Implementierungsansätze erklärt und ein Ausblick auf zukünftige Pläne gegeben. Für die Live-Demo wurde das Tablet auf einem weiteren Beamer angezeigt und die bisherige Oberfläche der App gezeigt. Dazu wurde live ein Eat-The-Whistle-Spiel analysiert, das auf dem Hauptbeamer zu sehen war. Auf dem Tablet wurden dann die Spielübersicht sowie die Spielzeit angezeigt.

8.4.10 Serverausfall und Umzug

05.11.2014

Nachdem wir nun physikalisch zurück ins OFFIS gezogen sind, nachdem der Flur renoviert wurde, ziehen wir virtuell zurück in die Uni. Zumindest unsere Server. Nach einem Ausfall der Server im OFFIS, bei dem auch Daten verloren gingen (das letzte Backup war immerhin noch von Mittwoch Nacht letzter Woche, sodass nicht allzu viel verloren ging), entschlossen wir uns kurzerhand dazu, unsere Infrastruktur in die zuverlässigen Hände von Jörg und seinem Server „Duemmer “ zu geben. Marc P. und Chris vollzogen den Umzug und schafften es, die letzten Daten wieder einzuspielen, den Bamboo zum bauen zu überreden und die Nutzer in das neue System zu migrieren. Nun hoffen wir hier auf stabilere Zeiten.

8.4.11 Fleißig, fleißig

13.11.2014

Nachdem eine detaillierte Recherche nach Sensoren durchgeführt wurde, wurde schnell klar, dass wir nicht über das notwendige Budget verfügen, um uns solche Systeme zuzulegen.
Als Ersatz wurde in dieser Hinsicht die GPS-Funktion von Smartphones angesprochen. Zwar sind diese zum Teil sehr ungenau, aber lieber ungenaue Sensoren anstatt gar keine Sensoren. Hierfür wurde dann eine Applikation geschrieben, die die GPS-Daten des Handys an einen Server sendet, der als Quelle in Odysseus eingebunden werden kann.
Auch der Server wurde eigens für diesen Zweck geschrieben und mit einer GUI versorgt.
Wie sollte allerdings der Ball getrackt werden? Hierfür wurde nach langer Überlegung ein Schaumstoffball genutzt, aufgeschnitten und mit einem Smartphone ausgestattet.

8.4.12 Basketballspiel aufgezeichnet

02.12.2014

Funktioniert unser System wirklich? Auch, wenn man die Daten aus einem echten Spiel nimmt? Um das zu testen, haben wir uns auf dem Sportplatz in Wechloy getroffen und mit dem nötigen technischen Equipment die Meisterschaft im GPS-Ball ausgefochten. Mit viel Einsatz wurde um die Körbe gekämpft und trotz eisigen Temperaturen kamen die Spieler gut ins Schwitzen.

Zu den technischen Details: Jeder Spieler hatte ein Android-Gerät in der Tasche, auf dem unsere GPSender-App lief. Diese war mit dem lokal aufgebauten WLAN verbunden, genau wie das Notebook von Marc, auf dem das Empfangsprogramm lief. Damit wurden die Positionsdaten von jedem Gerät aufgezeichnet und in einer CSV-Datei gespeichert. Die Feldabmessungen wurden vorher ebenfalls mit der GPSender-App aufgenommen. Der Ball war ebenfalls mit einem Smartphone ausgestattet. Es ging dabei vor allem um die Frage, ob die GPS Daten genau genug sind, um für unsere Analysen geeignet zu sein. Und eine kurze erste Analyse nach dem Spiel zeigte: Die Daten können sich durchaus sehen lassen. Es ist möglich, das Spiel auf dem Tablet mit Herakles zu analysieren. Wie genau die Daten sind, müssen wir noch weiter untersuchen. Trotzdem kann die Aufzeichnung schon jetzt als kleiner Erfolg gewertet werden. Die reale Nutzbarkeit rückt näher!

8.4.13 Baskets Oldenburg

18.12.2014

Als Weihnachtsgeschenk wurde die PG von Prof. Appelrath zu einem Basketballspiel der Oldenburger Baskets eingeladen, die gegen das Team von Virtus Rom im Eurocup antreten mussten. Das Spiel fand am Mittwoch, 17.12.2014 statt. Auch wenn zu diesem Zeitpunkt bereits klar war, dass die Baskets ausgeschieden sind, wollten sie sich dennoch mit einem guten Spiel aus dem Eurocup verabschieden.

Spannend war das Spiel von Anfang an. Nach dem Herantasten an den Gegner wurde den Baskets relativ schnell klar, dass hier doch noch mehr drin ist, als vielleicht anfangs angenommen wurde. Auch wenn der Gegner zwischenzeitlich dem Unentschieden nahe war, führten die Baskets die gesamte Spielzeit über verdientermaßen und gewannen das Spiel klar mit 80:64. Für Verwirrung sorgte bei einigen Leuten aber die Anzeigetafel. Eine Person (Namen dürfen hier nicht erwähnt werden) war sich nicht so ganz sicher, was denn die roten Zahlen bedeuten. Seiner Ansicht nach konnte man hier die Punkte ablesen, die ein Spieler durch das Werfen von Körben erzielt hat. Bei genauerer Betrachtung musste er dann allerdings feststellen, dass die Summe wohl nicht dem Spielstand entspricht. Letztendlich konnte aber auch dieses Problem gelöst werden: es handelte sich um die Rückennummern.

Abschließend kann man sagen, dass der Abend mal wieder hervorragend war und das auch wir zum Sieg der Mannschaft mit unseren Klatschpappen beitragen konnten.

8.4.14 Fail better

15.01.2015

Heute haben wir die WLAN-Ortung im größeren Umfang getestet - und leider feststellen müssen, dass es nicht wie gewünscht funktioniert. Trotz des vorher von Marc P. gut geplanten Aufbaus mit mehreren WLAN-Access-Points, speziellen App-Anpassungen (GPSender) und präpariertem Ball konnten wir keine vernünftigen Ortungspunkte aus den Daten berechnen.

Ziel war es, in der Sporthalle in Ofen die ein Basketballspiel durchzuführen und die Positionen der Spieler und des Balls durch WLAN-Ortung mit Smartphones festzustellen. Im kleinen Test vorher hat dies gut funktioniert, in unserem nun größeren Testaufbau hat es jedoch nicht geklappt. Die Punkte der Spieler, die zwar in der Coach App zu sehen waren, sprangen wild hin und her, auch wenn der Spieler auf einem Punkt stand. Auch nach einer Verkleinerung des Spielfeldes wurde es nicht besser, sodass wir den Test leider erfolglos beenden mussten. So gilt weiterhin, dass die Positionsermittlung mit Smartphones nur auf Basis von GPS-Koordinaten gut funktioniert.
Trotzdem hatten wir bei ein paar (unaufgezeichneten) Basketballspielen viel Spaß. Von diesem kleinen Rückschlag lassen wir uns nicht vom Kurs abbringen.

Ever tried. Ever failed. No matter. Try again. Fail again. Fail better. (Samuel Beckett)

8.4.15 Feature Freeze

29.01.2015

Heute ist es soweit, die PG neigt sich dem Ende zu und in der kalten Jahreszeit haben wir nun unseren Feature Freeze. Ab dem heute beginnenden Sprint werden noch Bugs gefixt, die Dokumentation geschrieben und die Präsentation(en) vorbereitet. Dazu gehört die Endpräsentation, jedoch auch noch die Präsentation bei der Data Streams and Event Processing (DSEP), einem Workshop bei der BTW.

Die einzelnen Teile von Herakles befinden sich nun auf einem nutzbaren Niveau. Die App hat die grundlegenden Funktionalitäten und noch ein paar Features, die darüber hinaus gehen (z.B. das Vergleichen von Spielern oder das Zeichnen auf der Spieloberfläche). Die Anfragen wurden nochmal überarbeitet und liefern nun zuverlässigere Ergebnisse. Das Load-Balancing wurde robuster gemacht, die „Moving-State Strategie“, bei der die Zustände übertragen werden, funktioniert weitestgehend, kann aber auch noch verbessert werden (z.B. fehlt noch das Senden größerer Zustände). Im Recovery funktioniert die Standard-Strategie nun zuverlässig und das Active-Standby kommt ganz gut voran, hat aber eventuell noch einen Fehler.

Somit haben wir nun einen frostigen Feature Freeze ...

8.4.16 Hack Night

01.02.2015

Am Samstag trafen sich mehrere wackere Helden der Projektgruppe zur gemeinsamen Hack Night. Zunächst war nicht ganz sicher, ob wir die Hack Night überhaupt durchführen können, da die Universität doch nur bis 18:00 geöffnet war. Aber dank des freundlichen Personals am Service Point gelang es uns trotzdem, die Universität zu stürmen. Schnell war die ARBI häuslich eingerichtet. Fortschritt konnte an verschiedenen Stellen auch verzeichnet werden. Insgesamt ein gelungener Abend!

8.4.17 Schülerinformationstag

10.03.2015

Am 06.03 fand der Schülerinformationstag „Informatik - phänomenal“ statt. An diesem Tag waren auch wir mit einem kleinen Stand vertreten und präsentierten Schülern unsere Ergebnisse. Ein interessanter Tag, an dem wir unser System wieder einmal testen konnten. Hin und wieder traten zwar ein paar Fehler auf, insgesamt waren aber alle zufrieden.

8.4.18 Evaluationsphase in voller Fahrt

16.03.2015

In der letzten Woche wurde mit der Evaluation der umgesetzten Funktionen begonnen. Dabei war es zunächst der Plan, die Anfragen auf den Raspberry PI's auszuführen. Aufgrund größerer Probleme wurde Mitte der Woche entschieden, dass die Evaluation nun auf den Notebooks der Projektmitglieder durchgeführt wird. Diese bringen vor allem die Heterogenität der Peers mit in die Auswertung ein.

Aktuell werden die umgesetzten Funktionen auf Herz und Nieren getestet. Dabei werden vor allem die Funktionen innerhalb von Odysseus (Anfragen, Recovery und Load-Balancing) und die der Client Systeme (Monitoring-Application und Coach App) getestet. Insbesondere die Performance-

Auswertungen der Anfragen stellen eine große Herausforderung dar, da eine Vielzahl von äußerlichen Einflüssen (WLAN, heterogene Systeme) beachtet werden müssen. Eine große Bereicherung für die Auswertung der verteilten Anfragen ist die Monitoring-Application, da hierüber eine Darstellung der verteilten Anfragen möglich ist. Trotz aller Probleme sind nun nahezu alle Anfragen unter verschiedene Bedingungen ausgeführt worden und aus den gesammelten Daten werden nun entsprechende Diagramme und Boxplots erzeugt.

Generell sind aktuell nahezu alle Probleme aus dem Recovery und dem Load-Balancing beseitigt, sodass auch hier die Evaluation durchgeführt werden kann.

8.4.19 CeBIT 2015

24.03.2015

Die digitale Welt hat sich wieder in Hannover zur CeBit getroffen und eine kleine Delegation unserer Projektgruppe war ebenfalls dort, um nach den neusten Trends zu schauen. Dort haben wir schnell gemerkt, dass unser Thema sehr aktuell ist. Gleich zwei Aussteller haben Lösungen zur Sportanalyse vorgestellt, die unserem System Herakles gar nicht so unähnlich sind. Beide Systeme basieren auf SAP Hana. Jedoch bieten sie keine Möglichkeit zur Live-Analyse, wie es Herakles ermöglicht. Ein System basiert auf Videodaten, das zweite, Entwickelt von Hasso-Plattner-Institut, auf Sensordaten. Es war sehr interessant zu sehen, wie andere Entwickler an diese Aufgabe herangegangen sind.

8.5 Projektplanung

Die Projektplanung wurde vom Projektmanagement durchgeführt. Dabei wurden unterschiedliche Tools eingesetzt, unter anderem *Microsoft Project 2013*. Hiermit wurden Ressourcen und wesentliche Ziele definiert. Die Planung mit diesem Tool wurde allerdings nicht weiterverfolgt, da die Projektgruppe Scrum (siehe Abschnitt 8.1) nutzt und sich dadurch die Planung durch das Projektmanagement auf einzelne Meilensteine beschränkt. Folgende Meilensteine wurden definiert:

01.04.2014 Projektstart

30.06.2014 Projektarchitektur aufgesetzt

01.08.2014 Coach App zeigt erste Ergebnisse

01.10.2014 Load-Balancing funktioniert

21.10.2014 Zwischenpräsentation

15.11.2014 App visualisiert erste Statistiken

01.12.2014 Tests mit Sensoren

15.01.2015 Recovery funktioniert

15.01.2015 Sensordaten werden von P2P Netzwerk verarbeitet

01.02.2015 Tests Zusammenspiel Sensoren, P2P, App

01.03.2015 Coach App ist fertig und an Nutzerwünsche angepasst

31.03.2015 Projektende

Diese Meilensteine sind ebenfalls in einer Roadmap (Abbildung 8.1) dargestellt worden, um das Projekt insgesamt überschaubar darzustellen.

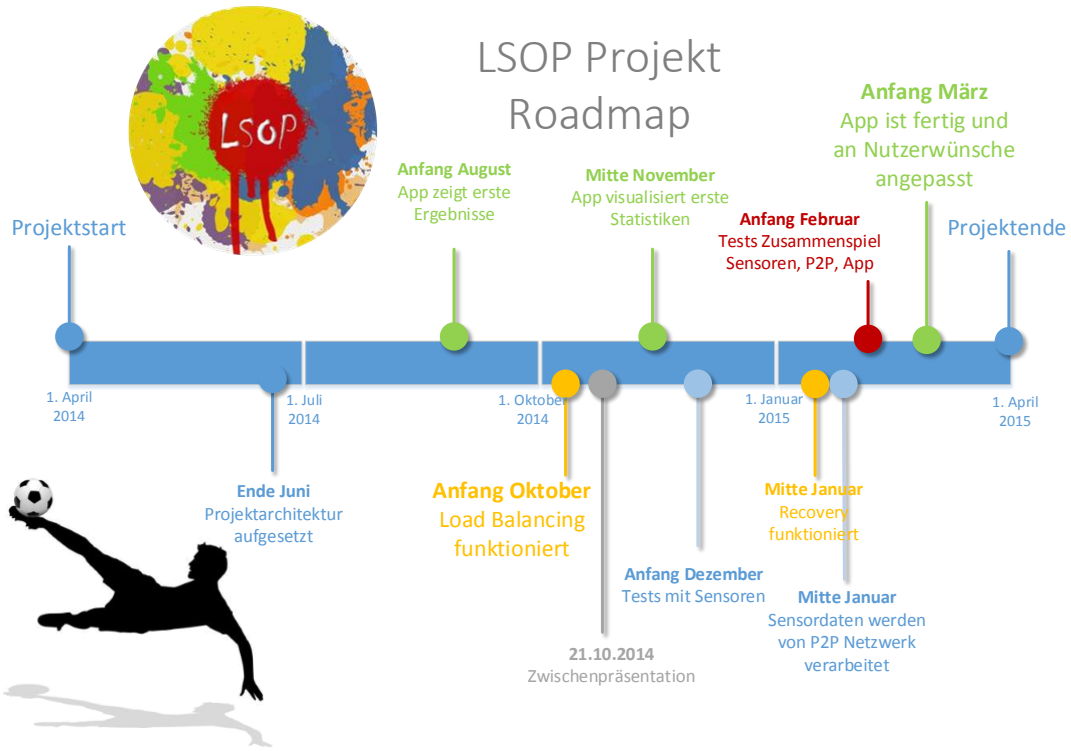


Abbildung 8.1: Roadmap LSOP

8.6 Kooperation

Zu Beginn der Projektgruppe wurde über verschiedene Kooperationspartner nachgedacht und mit diesen auch Gespräche geführt. Dabei handelte es sich einerseits um den Basketballverein *EWE Baskets Oldenburg* und andererseits um den Fußballverein *SV Werder Bremen*. Der Trainer der EWE Baskets wurde bereits vor Beginn der Projektgruppe vom Product Owner vorgeschlagen, sodass zunächst versucht wurde, diesen als Projektpartner zu gewinnen. Nachdem die Gespräche allerdings zu keinem Ziel führten, ist als weiterer möglicher Partner der SV Werder Bremen vorgeschlagen worden. Grund hierfür war das Interesse des Trainers, das während eines Interviews mit der Projektgruppe geweckt wurde. Allerdings ist auch diese Kooperation nicht zustande gekommen, da keine brauchbaren Sensoren auf Seiten des Vereins und der Projektgruppe verfügbar waren. Somit kann letztendlich gesagt werden, dass für das Projekt kein Kooperationspartner vorhanden war.

8.7 Zusammenfassung

Das Vorgehen im Projekt wurde maßgeblich durch den Einsatz von Scrum als agiles Vorgehensmodell geprägt. Das agile Vorgehen und das Arbeiten in Sprints bietet den Vorteil, dass schnell auf Veränderungen bei den Anforderungen reagiert werden kann. Für eine effektive Aufwandsschätzung wurde Magic Estimation genutzt, das im Vergleich zum Planning Poker deutlich weniger zeitlichen Aufwand in Anspruch nahm. Um vorhandene Probleme während des Projektes zu beseitigen, wurden verschiedene Retrospektive-Techniken eingesetzt. Für eine bessere Unterstützung des Scrum-Vorgangs wurde das Tool Jira Agile eingesetzt, sodass der Fortschritt des Sprints jederzeit sichtbar ist.

Durch das Projektmanagement ist während des gesamten Projektes ein Projekttagebuch geführt worden, in dem wichtige Ereignisse, wie beispielsweise der erste Durchstich (Erste Zusammenarbeit aller Teilkomponenten), beschrieben wurden. Darüber hinaus ist ein Meilensteinplan in Form einer Roadmap erstellt worden, an der jederzeit ersichtlich war ob der aktuelle Zeitplan eingehalten wird. Für eine effektive Verteilung der Aufgaben sind zu Beginn der Projektgruppe entsprechende Projektrollen festgelegt worden. Der Vorteil durch die Rollenverteilung lag darin, dass Aufgaben einfach delegiert werden konnten und es für alle Bereiche einen Verantwortlichen und gleichzeitig Ansprechpartner gab.

Die Anforderungen an eine Tablet-Anwendung, auf der die Ergebnisse der Sportanalysen angezeigt werden, wurden durch Interviews bei Fußball-Vereinen ermittelt. Befragt wurden Viktor Skripnik, Trainer bei Werder Bremen (damals Trainer bei Werder Bremen II) und Jörg-Uwe Klütz, Trainer beim BV Cloppenburg. Die Befragung wurde vor der Durchführung zunächst vorbereitet. Auch eine Kooperation mit Sportvereinen, unter anderem mit den EWE Baskets aus Oldenburg, waren zu Beginn der Projektgruppe geplant, sind allerdings bis zum Ende nicht zu Stande gekommen. Einer der zentralen Gründe hierfür ist, dass es keine entsprechenden Sensoren gibt, die für die Aufnahme der Positionen verwendet werden können. Dabei stellt vor allem die Integration von Sensoren in Basketbälle ein Problem dar.

9 IT Infrastruktur und eingesetzte Tools

Im Rahmen der Projektgruppe werden eine Vielzahl von Programmen und Server-Anwendungen eingesetzt, damit eine effiziente Zusammenarbeit möglich ist. Darüber hinaus werden aufgrund der großen Heterogenität der eingesetzten Technologien verschiedene Entwicklungsumgebungen benötigt. Für die Dokumentation der Ergebnisse und auch während der Projektdurchführung werden darüber hinaus weitere Software-Lösungen benötigt. In diesem Kapitel werden diese Tools genauer erläutert und die Gründe für ihren Einsatz beschrieben. Zudem wird darauf eingegangen warum Alternativen nicht eingesetzt werden.

9.1 Server Tools

Um die Zusammenarbeit der Gruppe zu ermöglichen, werden einige Tools eingesetzt, die Dienste anbieten, mit denen die Entwicklung ermöglicht oder erleichtert wird. Die Anwendungen wurden zu Beginn des Projekts auf einem Server der Abteilung Informationssysteme im OFFIS installiert. Aufgrund von Serverproblemen wie Ausfällen mit Datenverlust wurden die Installationen etwa zur Mitte des Projekts auf den Server „duemmer“ der ARBI migriert. Im folgenden sind die eingesetzten Werkzeuge und der Grund für ihren Einsatz näher beschrieben.

9.1.1 Subversion (SVN)

Als Versionsverwaltung wird SVN eingesetzt. Es wurde ausgewählt, da der Quellcode von Odysseus in einem SVN verwaltet wird und es als Versionskontrolle für die Weiterentwicklung von Odysseus P2P somit eingesetzt werden musste. Zusätzlich wurde ein SVN auf den Servern der ARBI eingerichtet, in dem der Quellcode der selbst entwickelten Anwendungen wie der Coach App sowie zusätzliche Dokumente wie Paper und Präsentationen verwaltet wurden.

9.1.2 Jira

Um die Zusammenarbeit besser koordinieren zu können, wird das Projektverfolgungstool Jira des Herstellers Atlassian eingesetzt. Es ermöglicht die Verwaltung von Aufgaben und User Stories, die durch den Scrum-Prozess entstehen. Die Projektmitglieder können sich einzelne Aufgaben (engl. Tasks) zuweisen, um diese zu bearbeiten. Zur besseren Kommunikation zwischen den Teammitgliedern können diese unter anderem auch kommentiert werden. Für jede Aufgabe gibt es einen Workflow, der durchlaufen werden muss. Jira wird ähnlich wie andere Werkzeuge von Atlassian über eine Weboberfläche im Browser bedient. Im Abschnitt 8.1.7 wird näher darauf eingegangen wie Jira die Projektgruppe im Scrum-Prozess unterstützt hat.

9.1.3 Confluence

Confluence ist ein Wiki-Werkzeug, das ebenfalls vom Hersteller Atlassian stammt und daher mit Jira stark verknüpft ist. Es dient vor allem dazu, dass die Mitglieder der Projektgruppe ihr Wissen schneller und einfacher austauschen können. In dem Wiki werden wichtige Informationen, Proto-

kolle, Erklärungen zu häufigen Fragen sowie Dokumentationen zu Konzepten und Implementierungen gesammelt und verwaltet. Durch die enge Verzahnung mit dem Projektverfolgungstool Jira können Seiten mit Aufgaben verknüpft werden, sodass zum Beispiel in einer Dokumentation auf einen Implementierungs-Task verwiesen werden kann. Des Weiteren werden die Stundenzettel der Projektmitglieder, auf denen die geleistete Arbeit mit der Anzahl der aufgebrachten Stunden vermerkt wird, in Confluence gepflegt. Wie die anderen Werkzeuge von Atlassian wird Confluence über eine Weboberfläche im Browser bedient. Eine Beispielseite ist in Abbildung 9.1 zu sehen. Hier wird ein Teil des Load-Balancings dokumentiert. Oben ist die Verknüpfung mit einem Task aus Jira zu erkennen. Jedes Gruppenmitglied kann jeden Artikel bearbeiten und somit Fehler beheben oder den Artikel erweitern und aktuell halten.

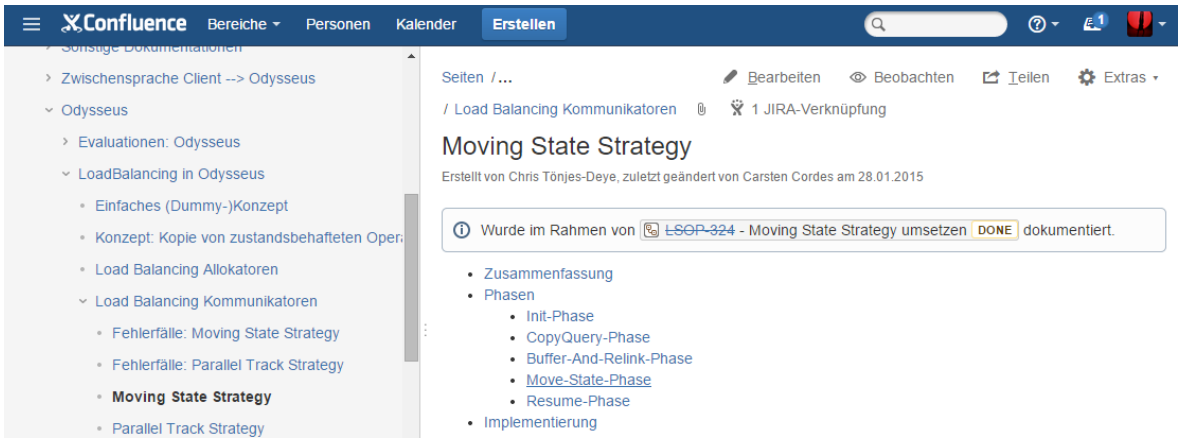


Abbildung 9.1: Beispielartikel aus Confluence, der ein Teil des Load-Balancings dokumentiert.

9.1.4 Bamboo

Bamboo ist die dritte eingesetzte Software aus dem Paket von Atlassian. Es ist ein Build-Server, der automatisch die von der Projektgruppe erstellten Softwareprodukte erzeugt. Dies hilft dabei, Fehler frühzeitig zu erkennen und zu vermeiden. Die automatisch gebauten Projekte sind folgende:

- Coach App
- GPSender App
- Monitoring Application
- LSOP Service
- Developer App (ehemals)

Bamboo bietet zudem die Möglichkeit, Projekte automatisch zu verteilen. Diese Funktion wird genutzt, um die fertig gebauten LSOP Serives automatisch in das JFrog Artifactory (siehe Abschnitt 9.1.5) zu laden, damit es von dem eingesetzt Build-Tool Gradle in Android Studio geladen und verwendet werden kann.

9.1.5 JFrog Artifactory

JFrog Artifactory ist ein Maven-Repository. Es bietet die Möglichkeit, eigene Projekte, bei der Projektgruppe für gewöhnlich JAR-Dateien, über Build-Tools zu beziehen. Da in der Android-Entwicklung standardmäßig Gradle, welches Maven-Repositories nutzt, als Build-Tool zum Einsatz kommt, können somit selbst geschriebene Bibliotheken über das Build-Tool eingebunden werden. In dem Fall der Projektgruppe ist dies der LSOP Service, der von Bamboo erzeugt und von dort in JFrog Artifactory geladen wird. Anschließend kann Gradle beim Kompilieren der Coach App auf die neueste Version des LSOP Services zugreifen. JFrog Artifactory wird über eine Weboberfläche verwaltet.

9.1.6 FishEye

Atlassian bietet eine weitere Software an, die mit den bereits beschriebenen Werkzeugen für Entwickler verknüpft ist. FishEye stellt Statistiken über die eingebundenen SVN-Repositories bereit. Durch die enge Verzahnung mit Jira bietet FishEye den Vorteil, dass SVN-Commits, welche das Kürzel eines Tasks in dem Commit-Kommentar beinhalten, automatisch mit dem jeweiligen Task verknüpft werden. Somit sind die Commits zu einem Task in Jira direkt zu sehen. FishEye wird ebenfalls über eine Weboberfläche im Browser bedient.

9.1.7 SonarQube

SonarQube wurde kurzzeitig eingesetzt, um Code-Metriken zu erstellen. Die Software, die über eine Weboberfläche bedient werden kann, erstellt aus einem SVN-Repository automatisch Code-Metriken wie etwa die Testabdeckung. Da das Tool jedoch kaum genutzt wurde, hat die Projektgruppe das SonarQube auf dem Server deinstalliert und nicht weiter verwendet.

9.2 Entwicklungsumgebungen

Damit die Entwicklung der Softwarekomponenten innerhalb der Projektgruppe ohne Inkompatibilitäten abläuft, müssen alle Mitglieder die gleichen Entwicklungsumgebungen benutzen. Diese wurden zu Beginn des Projektes evaluiert, sodass anschließend benötigte Umgebungen bestimmt werden konnten. Dieser Abschnitt dient dazu, einen Überblick über die verwendeten Entwicklungsumgebungen zu geben.

9.2.1 Eclipse

Die Projektgruppe verwendet für die Programmierung von Odysseus P2P Komponenten Eclipse. Der Grund dafür ist, dass die bisherige Entwicklung von Odysseus in Eclipse mit dem OSGi-Framework stattgefunden hat und die Projekte aus Eclipse nicht mit anderen Entwicklungsumgebungen ohne weiteres kompatibel sind.

9.2.2 Android Studio

Für die Entwicklung der Coach App hat sich die Projektgruppe für Android Studio entschieden. Es ist die offizielle Entwicklungsumgebung für die Android-Programmierung, wird von Google selbst weiterentwickelt und basiert auf IntelliJ. Im Vergleich zu Eclipse mit installiertem Android Developer Tools Plugin bietet Android Studio einige Vorteile, da es auf Android spezialisiert ist und so enger mit dem Framework verzahnt ist. Zum Beispiel vereinfacht der eingebaute GUI-Designer die Erstellung von Grafischen Oberflächen. Eng mit der Entscheidung für Android-Studio verknüpft, ist die Nutzung von Gradle als Build-Tool, das hier standardmäßig verwendet wird. Unabhängig von der IDE ist Gradle das Build-Tool der Wahl, da Gradle verglichen mit den Alternativen am schnellsten beim Bauen von Projekten ist und es inkrementelles Bauen unterstützt. Somit muss nicht immer das komplette Projekt neugebaut werden, sondern nur die Teile, die verändert wurden.

9.2.3 IntelliJ

Für die Entwicklung der Developer App und des LSOP Services wurde IntelliJ benutzt. Die Wahl für IntelliJ fiel im Zusammenhang mit der Entscheidung für JavaFX als GUI-Framework, da die Integration hier am besten ist. JavaFX trennt, wie auch Android, die Definition der Oberfläche vom Java Code. Dadurch wird das Befolgen des MVC-Patterns vereinfacht. Weiterhin enthält es die Möglichkeit für das Styling CSS zu verwenden und eigene Charts. Weitere Gründe für IntelliJ sind die sehr gute Gradle Unterstützung und Autovervollständigung im Code-Editor.

9.3 Dokumentationstools

In diesem Abschnitt werden die Anwendungen genauer erläutert, die im Bereich der Dokumentation eingesetzt wurden. Dazu gehören Wiki-Systeme, aber auch Modellierungswerkzeuge. Durch die kontinuierliche Dokumentation der Ergebnisse und des Projektverlaufes war es ohne Aufwand möglich, das erlangte Wissen einer Person auf die gesamte Gruppe zu übertragen.

9.3.1 Confluence

Wie bereits im Abschnitt 9.1 beschrieben, stellt Confluence das zentrale Werkzeug zur Dokumentation während der Projektdurchführung dar. Da Informationen sehr gut gegliedert werden können und einfach zugreifbar sind, bietet Confluence wesentliche Vorteile im Vergleich zu LaTeX da die Informationen während der Durchführung des Projektes häufigen Änderungen unterworfen sind. Darüber hinaus werden in Confluence auch Protokolle, Stundenzettel und andere organisatorische Informationen erfasst, die nicht in die Abschlussdokumentation aufgenommen werden.

9.3.2 LaTeX und TeXMaker

Für die Erstellung des Abschlussdokumentes wurde das Softwarepaket LaTeX eingesetzt, das die Benutzung des Textsatzsystems TeX vereinfacht. Entscheidend für den Einsatz ist zum einen die Plattformunabhängkeit des Systems gewesen und zum anderen die Aufteilung des Dokumentes in einzelne Dateien. Letztes hat den entscheidenden Vorteil, dass viele Personen gleichzeitig an dem Dokument

arbeiten können und es dabei zu möglichst wenig Merge-Konflikten kommt. Als Alternative zu der Verwendung von LaTeX sei auch noch die Office-Suite von Microsoft und speziell das Textverarbeitungsprogramm Microsoft Word genannt, das durch die Universität kostenlos zur Verfügung gestellt wird. Allerdings ist eine Zusammenarbeit an einem Dokument nur über einen Share-Point Server möglich und darüber hinaus ist insbesondere das Speichern des Dokuments mit vielen gleichzeitigen Nutzern sehr langsam. Ein weiterer Grund gegen die Verwendung von Microsoft Office ist, dass Microsoft Word ausschließlich auf Windows und Mac OS X Systemen lauffähig ist, sodass dies endgültig gegen den Einsatz der Software sprach.

9.3.3 Visual Paradigm

Das UML-Modellierungswerkzeug Visual Paradigm in der Version 11.0 wird für die Erstellung sämtlicher UML-Diagramme genutzt. Die Entscheidung für Visual Paradigm wurde zu Beginn gefällt und durch die Plattformunabhängigkeit, kostenfreie Lizenz und im Vergleich zu Rational Software Architect (RSA) guten Bedienbarkeit begründet. In Visual Paradigm wurden Klassen-, Sequenz- und Zustandsdiagramme erstellt. Da alle Projektmitglieder mit der identischen Version arbeiten, ist ein Austausch über das SVN problemlos möglich, solange es nicht zu Merge-Konflikten kommt. Diese sind bei dem verwendeten Format nur umständlich zu lösen. Damit die Wahrscheinlichkeit der beschriebenen Merge-Konflikte reduziert und die Übersichtlichkeit erhöht wird, wurden für die verschiedenen Bereiche bzw. Systeme einzelne Visual Paradigm Projekte angelegt.

9.3.4 Draw.IO

Mit dem Confluence Plugin Draw.IO wurden sämtliche konzeptuellen Grafiken erzeugt. Das Plugin bietet eine Reihe von Vorteilen gegenüber anderen Software-Lösungen in diesem Bereich:

- plattformunabhängig
- kostenlose Nutzung möglich
- direkte Integration in Confluence und Erstellung der Diagramme direkt im Browser
- Export der Diagramme in verschiedenen Formaten (u.a. PDF für die Abschlussdokumentation)

Vor allem die Plattformunabhängigkeit war entscheidend, da verschiedenste Betriebssysteme innerhalb der Projektgruppe im Einsatz sind und Draw.IO eine Möglichkeit bietet die Grafiken im Browser anzupassen. Bei der Entscheidung standen noch weitere Werkzeuge zur Auswahl wie Microsoft Visio (nur Windows), OmniGraffle (nur Mac OS X) sowie Inkscape.

9.3.5 Balsamiq

Für die Benutzeroberflächen der Coach App wurden entsprechende Wireframes mit dem Werkzeug Balsamiq Mockups erstellt. Dieses Werkzeug ist neben Axure Marktführer und es lassen sich mit wenig Aufwand entsprechende Wireframes erzeugen. Auch eine Verbindung der Mockups und somit die Simulation von Anwenderinteraktionen ist mit diesem Werkzeug möglich. Balsamiq bietet darüber

hinaus vorgefertigte Vorlagen für die Erstellung von Wireframes für Android Anwendungen. Dies stellt einen wesentlichen Vorteil dar, da es auf diese Weise schnell möglich war, die Wireframes für die Coach App zu erstellen.

9.4 Zusammenfassung

Während der Projektdurchführung werden verschiedene Serveranwendungen eingesetzt um in erster Linie eine effiziente Zusammenarbeit zu ermöglichen. Hierzu gehört die Versionsverwaltungssystem SVN, die zum einen für die Arbeit an Odysseus benötigt wird und zum anderen für die Entwicklung der Client-Anwendungen. Darüber hinaus wird Jira als Projektverfolgungstool eingesetzt und Confluence als Wiki-System für die Dokumentation während der Projektdurchführung. Ergänzt werden diese beiden Anwendungen durch die optimal integrierten Tools Fisheye, Bamboo und SonarQube, die für die Qualitätsüberwachung genutzt wurden. Artifactory wird für eine dynamische Bereitstellung von Java Artefakten benötigt.

Im Bereich der Entwicklungsumgebungen werden Eclipse für die Entwicklung von Odysseus P2P genutzt. Da es sich bei Odysseus um ein auf OSGi-basierendes Framework handelt und die Entwicklung aktuell immer in Eclipse durchgeführt wird, gab es für diesen Bereich keine Alternative. Für die Entwicklung der Coach App wurde Android Studio genutzt, da diese aktuell von Google selber für die Entwicklung von Android Anwendungen empfohlen wird und eine gute Gradle Unterstützung bietet. Da die Developer Application auf dem Framework JavaFX basiert, wurde für die Entwicklung IntelliJ Idea festgelegt, da die Unterstützung für dieses Framework hier am besten ist. Der LSOP Service wurde ebenfalls in IntelliJ Idea entwickelt, da die Gradle-Unterstützung besser ist als bei Eclipse.

Für die Dokumentation der Projektergebnisse wird während des Projektes Confluence genutzt, da hier schnelle Änderungen möglich und für alle Projektmitglieder direkt sichtbar sind. Die Abschlussdokumentation wird in LaTeX erstellt. Für die Modellierung von UML-Diagrammen wird Visual Paradigm und für Mockups wird Balsamiq genutzt. Mit dem Confluence Plugin Draw.IO können plattformübergreifend Grafiken beispielsweise für Konzepte erzeugt werden.

10 Installationshandbuch

Im Folgenden sollen die einzelnen Vorgänge zur Installation von Herakles näher beschrieben werden. Wichtig hierbei sind insbesondere die Einstellungen der Sensoren und des P2P-Netzwerks. Diese Einstellungen sind maßgeblich dafür verantwortlich, ob die Statistiken korrekt berechnet werden können. Des Weiteren wird erläutert, wie die Coach App eingerichtet werden kann, um die Statistiken zu betrachten. Um das P2P-Netzwerk überwachen zu können, ist auch die Installation der Monitoring App beschrieben. Alle Anwendungen sind in der Abgabe der Projektgruppe enthalten.

10.1 Sensoraufbau

Der Aufbau des Sensorsystems teilt sich in zwei Bereiche. *Sensoren* senden ihre Positionsdaten zu einem *Server*, der als Quelle in Odysseus definiert wird. Folgend werden die Teilbereiche näher erläutert.

10.1.1 GPServer

Zur Nutzung von GPServer muss sich die Anwendung zusammen mit Odysseus auf einem Rechner befinden. Der Server muss als Datenquelle in Odysseus eingebunden werden. Das vollständige Script zum Einbinden des Servers ist dabei in Listing A.1 abgebildet. Sobald der Server gestartet wird, können Sensoren ihre Positionen zu diesem Server senden. Die Abbildung 10.1 zeigt die grafische Oberfläche von GPServer. In der Liste wird jeder Sensor aufgelistet, der sich mit dem Server verbunden hat. Die Zeit wird immer dann aktualisiert, sobald neue Werte ankommen.

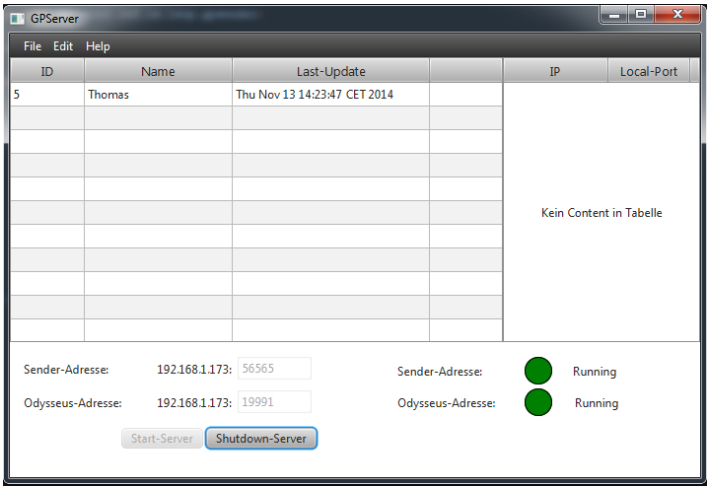


Abbildung 10.1: GPServer GUI

10.1.2 GPSender App

Mittels der GPSender App können diverse Einstellungen vorgenommen werden (siehe Abbildung 10.2). Zunächst muss die IP des Servers in den Einstellungen hinterlegt werden. Zudem muss hier der im DDC hinterlegte Name sowie die Identifikationsnummer für den jeweiligen Spieler eingetragen werden, damit der Server die ankommenden Daten zuordnen kann. Daneben besitzt die App die Möglichkeit, verschiedene Verfahren zur Positionsbestimmung zu nutzen (WIFI, GPS, GK).

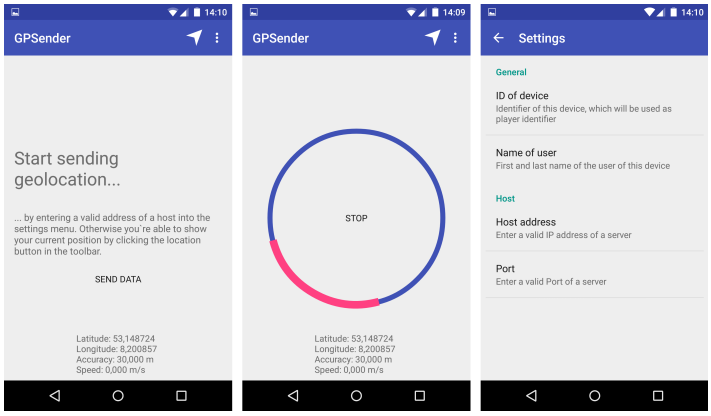


Abbildung 10.2: GPSender Hauptansicht und Einstellungsmenü

Zur Bestimmung der Spielfeldgröße kann unterstützend ein Assistent in der GPSender App genutzt werden, der über den Ortungsbutton in der Hauptansicht ausgewählt und gestartet werden kann (siehe Abbildung 10.3). Hierfür muss allerdings das GPS- bzw. GK- Tracking ausgewählt sein. Der Setup-Assistent führt den Nutzer dann durch die verschiedenen Einstellungen und schließt mit der Möglichkeit ab, eine Email mit den ermittelten Abmessungen zu verschicken. Die Werte müssen dann nur noch im DDC hinterlegt werden. Wo genau das DDC zu finden ist und wie dort Werte ergänzt werden können, wird im Abschnitt 10.2 näher beschrieben.

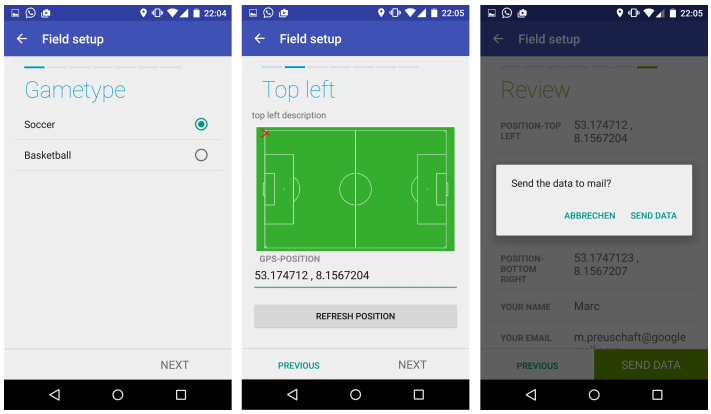


Abbildung 10.3: Teilansichten des Assistenten

Abschließend kann das Tracking nun über den „Start-Button“ in der Hauptansicht gestartet werden. Eine Anzeige am unteren Rand der Ansicht zeigt daraufhin die aktuellen Positionsdaten an.

10.2 Odysseus P2P

In diesem Abschnitt wird beschrieben, wie mit Odysseus P2P-Instanzen ein Netzwerk aufgebaut werden kann, welches in der Lage ist, gemeinsam die Sensor-Datenströme zu verarbeiten. Die einfachste Variante mit nur einem Peer und einer Instanz wird zuerst erklärt, um aufbauend auf dieser Basis die Benutzung von mehreren Peers in einem Netzwerk zu erläutern.

10.2.1 Einzelner Peer

Die einfachste Form, um Odysseus P2P für Herakles zu nutzen, ist ein einzelner Peer. Auf einem Computer mit einem der unterstützten Betriebssysteme (Windows, Linux und Mac OS) muss dazu Odysseus aus dem SVN ausgecheckt werden. Wie dies funktioniert, ist in dem Wiki von Odysseus beschrieben[1]. Wenn die Version genutzt werden soll, die für Herakles zum Ende des Projektes genutzt wurde, ist die Revision 23.850 geeignet.

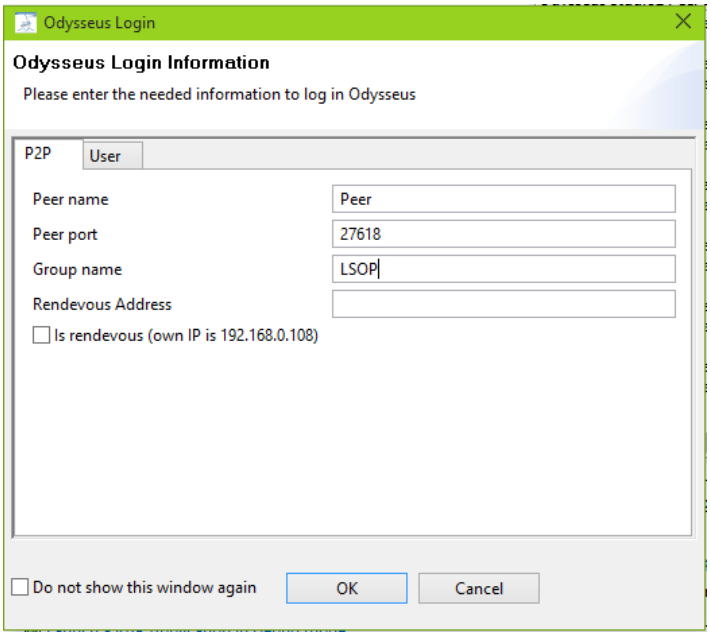


Abbildung 10.4: Peer-Einstellungen beim Starten von Odysseus P2P

Zum Starten von Odysseus muss für Herakles das Produkt „Odysseus Studio2 Peer LSOP.product“ verwendet werden. Nach dem Starten muss zunächst in einem neuen Fenster ein Workspace für Odysseus ausgewählt werden. Dies kann ein einfacher Ordner sein, der für diesen Zweck angelegt wurde.

[1] `http://wiki.odysseus.informatik.uni-oldenburg.de/display/ODYSSEUS/Development+with+Odysseus`, zuletzt abgerufen am 23.03.2015

Im nächsten Fenster werden Einstellungen für das P2P-Netzwerk festgelegt. Wie in Abbildung 10.4 zu sehen ist, kann der Name des Peers frei gewählt werden. Auch die Peergruppe ist in dem Fall eines einzelnen Peers nicht entscheidend. Die Zugangsdaten von Odysseus sind fest vorgegeben. Als Benutzername kann „System“ mit dem Passwort „manager“ verwendet werden.

Einfügen der Quelle

Um Sensordaten analysieren zu können, müssen diese als Datenstrom in Odysseus einfließen. Dazu wird zunächst eine Quellendefinition benötigt. Möglich ist z.B. die Verwendung der Daten der DEBS Grand Challenge von 2013 über eine CSV-Datei. Dazu kann die Quellendefinition von Listing A.2 verwendet werden. In der angegebenen Definition muss der Pfad zur CSV-Datei noch korrekt gesetzt werden. Eine solche Anfrage kann ausgeführt werden, indem ein neues Odysseus-Projekt erstellt wird und in dieser eine neue Odysseus Script Datei angelegt wird. In diese Datei muss der Inhalt aus dem Listing eingefügt werden.

Wenn dies der erste Start von Odysseus war, muss das DDC noch eingefügt werden, damit einige Konstanten richtig gesetzt werden. Für die DEBS-Quelle ist das DDC beispielhaft in Listing A.3 angegeben. Für das Einfügen des DDC muss Odysseus zunächst beendet werden. Erst danach sollte die DDC-Datei angepasst werden, welche sich in dem Home-Verzeichnis von Odysseus befindet.

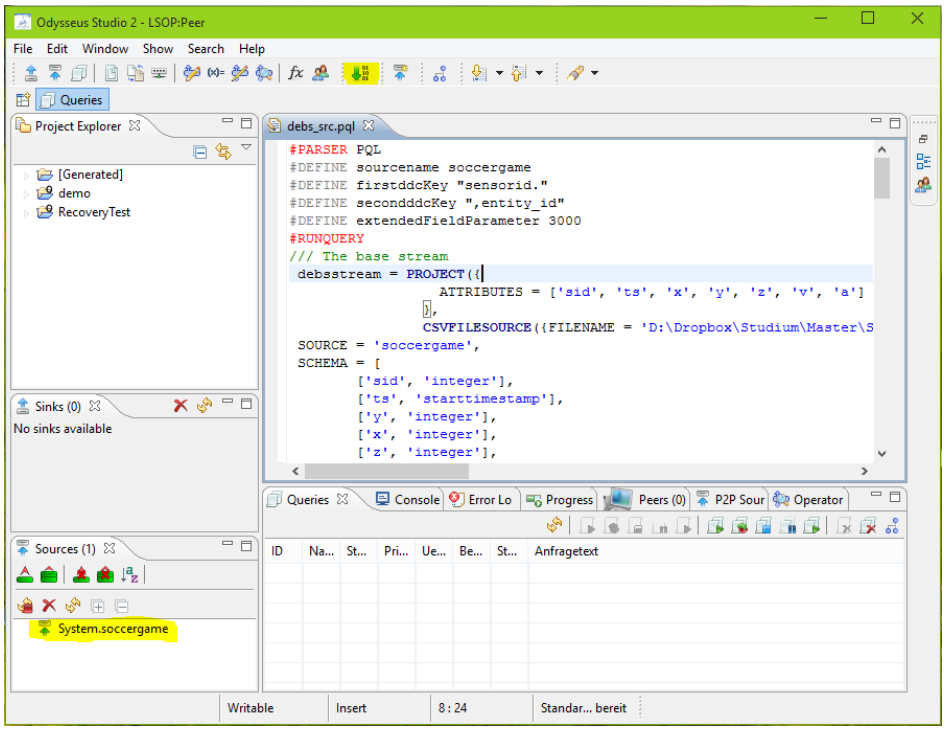


Abbildung 10.5: Odysseus Studio mit installierter Quelle.

Ist das DDC korrekt eingefügt und Odysseus wieder gestartet, kann die Quelle erstellt werden. Dazu wird in Odysseus Studio auf den kleinen grünen Pfeil geklickt, wie in Abbildung 10.5 zu sehen ist. Auf der Abbildung ist die Schaltfläche zum Installieren einer Anfrage gelb hinterlegt. Unten links ist, ebenfalls gelb hinterlegt, die nun installierte Quelle zu sehen.

Odysseus ist nun bereit, Anfragen zu verarbeiten, indem z.B. die Coach App gestartet wird. Alternativ können auch Sportanfragen direkt in Odysseus Studio gestartet werden. Dies funktioniert z.B. mit der einfachen Anfrage für die Analysezeit, die in Listing 10.1 angegeben ist.

```
#PARSER SportsQL

#RUNQUERY
{
    "statisticType": "global",
    "gameType": "soccer",
    "name": "gameTime"
}
```

Listing 10.1: Anfrage für die Analysezeit

10.2.2 Mehrere Peers

Mit dem oben beschriebenen Vorgehen konnte ein Peer für Odysseus eingerichtet werden. Auf diese Weise lassen sich auch mehrere Peers starten und miteinander verbinden. Das obige Vorgehen kann für weitere Peers auf dem selben Rechner durchgeführt werden. Jeder Peer benötigt dafür einen eigenen Workspace. Eine erneute Einrichtung des DDC ist nicht notwendig. Damit die Peers sich im Netzwerk finden, muss beim Start jedes Peers die selbe Peergruppe angegeben werden. Dies geschieht in dem Fenster, das bereits in Abbildung 10.4 zu sehen war.

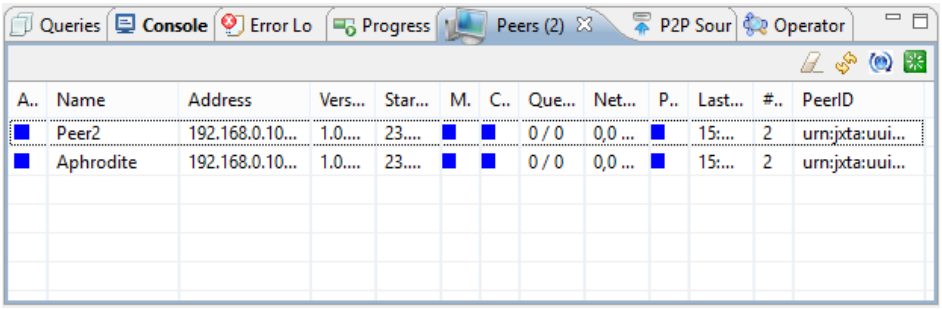


Abbildung 10.6: Liste bekannter Peers in Odysseus Studio.

Werden weitere Peers in der selben Gruppe gestartet, finden diese sich nach kurzer Zeit und zeigen dies an, wie in Abbildung 10.6 zu sehen ist. Wird nun eine Anfrage ausgeführt, kann diese verteilt werden, wenn die richtigen Parameter angegeben werden. Ein Beispiel hierfür ist eine verteile Anfrage für die Analysezeit, die in Listing 10.2 angegeben ist. Diese Anfrage wird mit diesen Einstellungen auf zwei Peers verteilt.

10.2.3 In einem Netzwerk

Mit dem im vorherigen Abschnitt angegebenen Vorgehen lässt sich Odysseus P2P auch in einem Netzwerk nutzen. Wichtig dafür ist, dass sich alle Peers im selben Netzwerk befinden, damit diese sich

```
#PARSER SportsQL
#CONFIG DISTRIBUTE true

#PEER_PARTITION OPERATORSETCLOUD
#PEER_ALLOCATE ROUNDROBIN
#PEER_POSTPROCESSOR FORCELOCALSOURCES
#PEER_POSTPROCESSOR MERGE

#ADDQUERY
{
    "statisticType": "global",
    "gameType": "soccer",
    "name": "gameTime"
}
```

Listing 10.2: Anfrage für die Analysezeit

finden. Auch darf das Versenden von UDP-Paketen im Netzwerk nicht unterbunden sein. Ansonsten ist das Vorgehen analog zu der Einrichtung mit mehreren Peers auf einem einzelnen Rechner.

Bei neueren Rechnern der Marke Apple kann es zu Problemen kommen, sodass sich die Peers nicht finden. Auch bei lokaler Ausführung mit mehreren Peers kann dies auftreten. Dies liegt an unsichtbaren Netzwerkadaptern, die für spezielle Apple-Funktionen genutzt werden (z.B. AirDrop). Der Adapter heißt z.B. „awdl0“ und ist über den Befehl „ifconfig“ zu sehen. Werden diese Adapter deaktiviert, funktioniert Odysseus P2P wieder. Ähnliche Probleme können unter anderen Betriebssystemen ebenfalls auftreten, wenn virtuelle Netzwerkadapter, z.B. von Virtualisierungslösungen, vorhanden sind.

10.3 Coach Application

Die Coach App kann wie eine ganz gewöhnliche Android App installiert werden. Jedoch ist die App nicht im Play Store von Google verfügbar, sondern muss aus einer apk-Datei installiert werden. Die apk-Datei ist in der Abgabe der Projektgruppe enthalten. Diese muss auf ein entsprechendes Tablet kopiert werden, auf dem mindestens Android 4.4 installiert wurde. In den Einstellungen des Tablets muss eingestellt sein, dass Anwendungen aus fremden Quellen installiert werden können. Dies ist für gewöhnlich in den Einstellungen unter dem Punkt „Sicherheit“ zu finden. Mit einem Tippen auf die apk-Datei kann die Coach App nun installiert werden.

Soll die App zusammen mit dem P2P-Netzwerk verwendet werden, muss darauf geachtet werden, dass sich das Tablet im selben Netzwerk befindet wie die Peers. Ist dies der Fall, kann die App gestartet werden. Nach dem Start können oben rechts die Einstellungen der App geöffnet werden. In dem Bildschirm, der in Abbildung 10.7 zu sehen ist, können die Einstellungen für die Peersuche getätigt werden. Wenn die Peergruppe richtig gesetzt ist, kann das Tablet die Peers automatisch finden. Sollte das nicht funktionieren, kann hier die manuelle Suche aktiviert werden, bei der die IP eines Peers direkt eingetragen wird. Der Port ist für gewöhnlich schon korrekt eingestellt. Unter dem Punkt „Que-

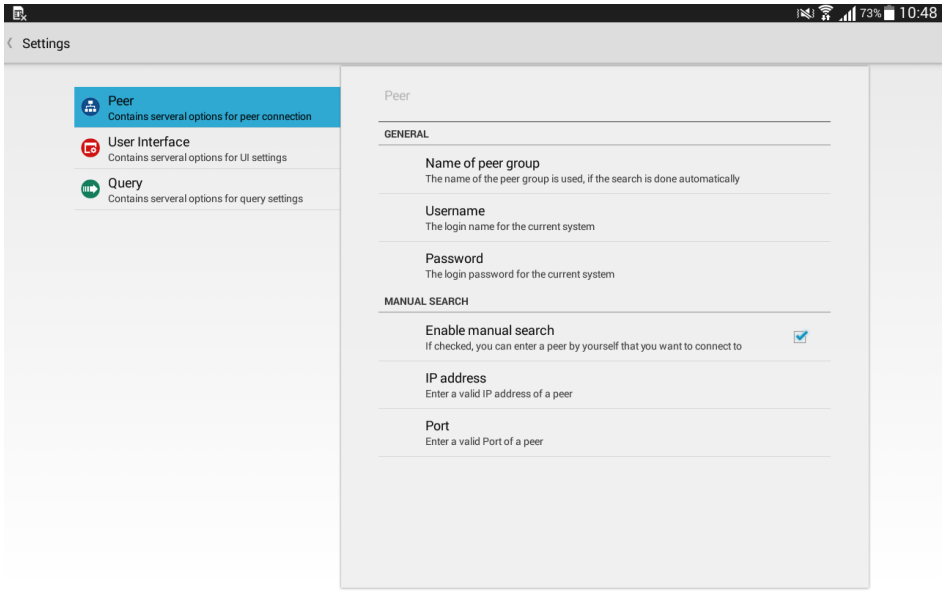


Abbildung 10.7: Einstellungen in der Coach App

ry“ auf der linken Seite können einzelne Anfragen aktiviert und deaktiviert werden. Für gewöhnlich sind alle Anfragen aktiviert. Nun ist die App bereit, eingesetzt zu werden.

10.4 Monitoring Application

Für das Ausführen der Monitoring Application wird zum einen die Datei „Odysseus.war“ benötigt, die in der Abgabe der Projektgruppe enthalten ist. Zum anderen muss „Jetty 8“[2] heruntergeladen werden, mit dem ein Webserver gestartet und die Monitoring Application ausgeführt werden kann. Dazu muss die Datei „Odysseus.war“ in den Unterordner „webapps“ des Jetty-Ordners geschoben werden. Anschließend kann der Webserver über eine Konsole und den Befehl „java -jar start.jar“ gestartet werden. Die Monitoring Application sollte nun über die URL „http://www.localhost:8080/Odysseus“ erreichbar sein. Ist der Standardport 8080 schon belegt, kann dieser in der Datei „jetty.xml“ geändert werden. Nachdem die Monitoring Application zum ersten Mal gestartet wurde, wird eine Konfigurationsdatei in dem Ordner „odysseus/webaccess“ des Home-Verzeichnisses angelegt. In dieser Datei ist es bspw. möglich, die Peer-Gruppe zu ändern oder die Peer-Suche zu deaktivieren. Nach Ändern der Datei muss der Webserver neu gestartet werden.

10.5 Zusammenfassung

In diesem Kapitel wurde beschrieben, wie Herakles eingerichtet und installiert werden kann. Dazu gehört die Installation eines Sensorsystems, z.B. auf Basis der GPSender App der Projektgruppe. Außerdem muss das Netzwerk aus Odysseus P2P-Instanzen so gestartet werden, dass sich die Peers finden und gemeinsam arbeiten können. Des Weiteren muss die Coach App installiert werden, um

[2] `http://download.eclipse.org/jetty/stable-8/dist/`

das Anzeigen von Statistiken zu ermöglichen. Für die Überwachung des Netzwerkes kann zudem die Monitoring App installiert werden.

11 Benutzerhandbuch

Dieses Kapitel stellt vor, wie der Benutzer die verschiedenen Funktionen der entwickelten Programme ausführen kann. Im Einzelnen sind dies die Coach App, die zentraler Bestandteil der Sportanalyse ist, die Monitoring Application zur Überwachung des Netzwerks, die Developer Application und die GPSender Application zur Erfassung der Positionsdaten auf dem Smartphone.

11.1 Coach Application

In den folgenden Abschnitten wird eine Übersicht über die Funktionen der Coach App gegeben. Die gängigsten Funktionalitäten sind in einzelne Abschnitte gegliedert.

11.1.1 Sportartauswahl

Die erste Ansicht, die nach dem Starten der Coach App angezeigt wird, ist die Sportartauswahl (siehe Abbildung 11.1). Hier kann der Nutzer eine Sportart auswählen (1). Dabei ist mit einer Wisch-Geste nach links oder rechts die nächste Sportart anwählbar. Mit einer Tipp-Geste wird die ausgewählte Sportart gestartet. Das Einstellungsmenü kann ebenfalls per Tipp-Geste auf das Icon bei (2) erreicht werden.

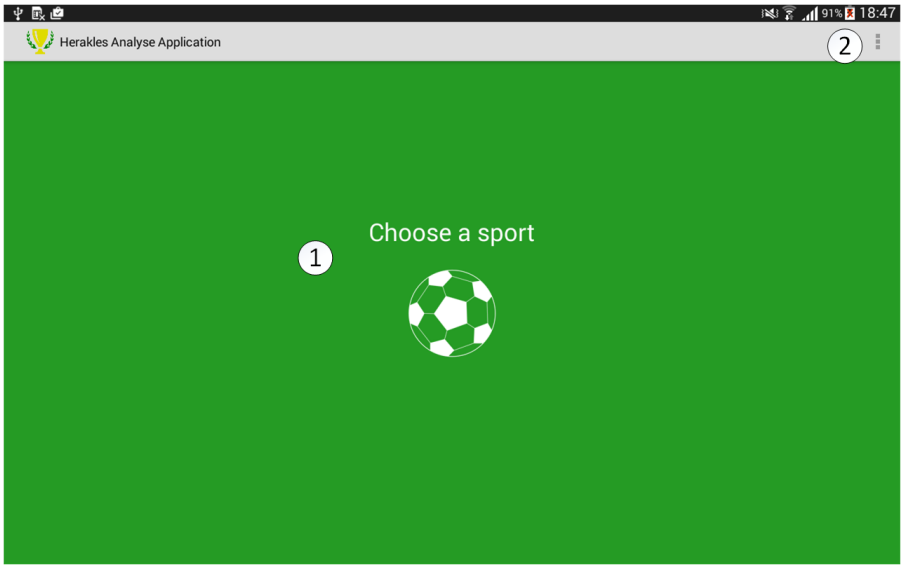


Abbildung 11.1: Sportartauswahl

11.1.2 Einstellungen

Das Einstellungsmenü umfasst mehrere Teilbereiche: Zunächst existieren Einstellungsmöglichkeiten bzgl. der Suche nach einem System, das die Statistiken liefert (siehe Abbildung 11.2). Hier kann entschieden werden, in welcher Peer Group nach einem geeigneten Peer gesucht werden soll, ob

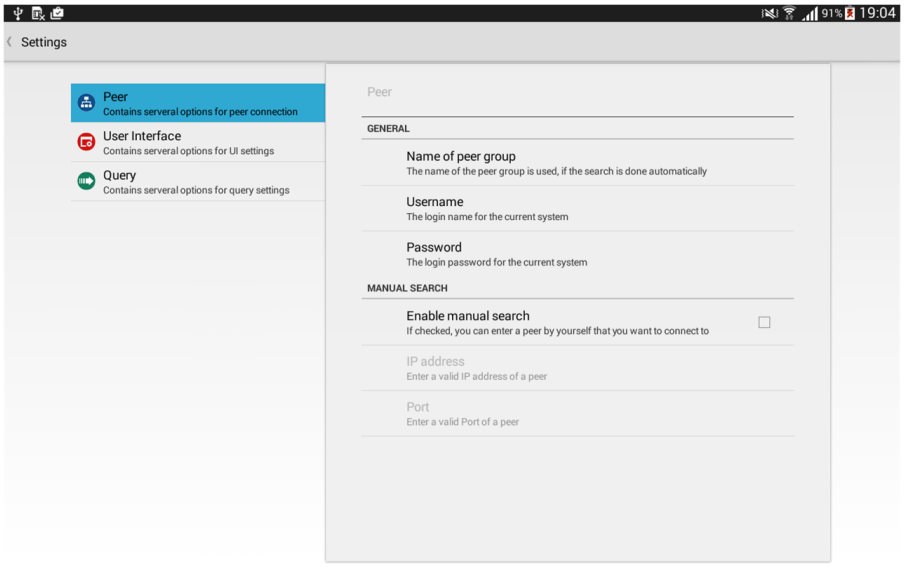


Abbildung 11.2: Einstellungsmenü - Systemsuche

eine manuelle oder automatische Verbindung mit einem Peer aufgebaut werden soll und wie die Zugangsdaten zum System lauten. Bei der manuellen Verbindung muss eine IP-Adresse sowie ein Port angegeben werden.

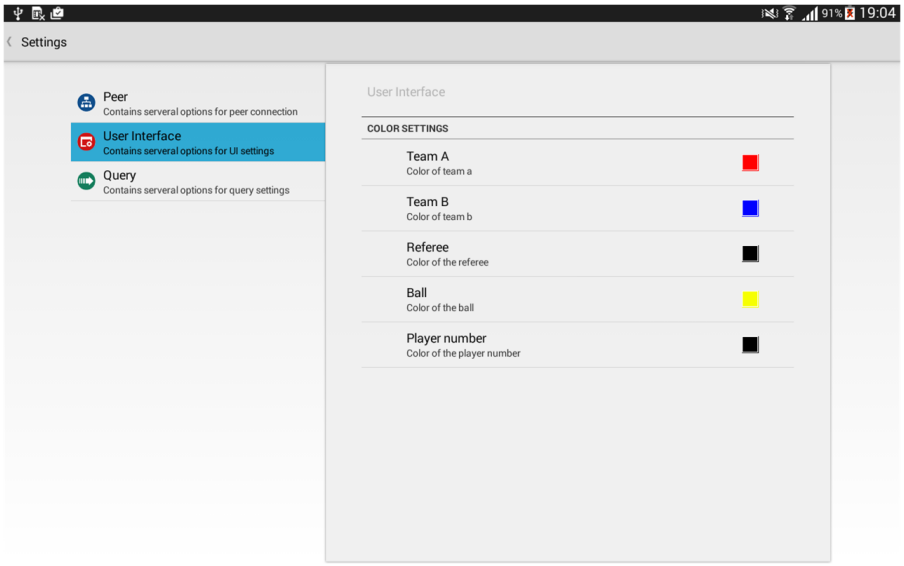


Abbildung 11.3: Einstellungsmenü - Ansicht

Der nächste Teilbereich umfasst allgemeine Einstellungen, die die Darstellung der Mannschaften, Schiedsrichter und Bälle auf dem Platz betreffen (siehe Abbildung 11.3). Der Nutzer hat die Möglichkeit, die Farben individuell festzulegen.

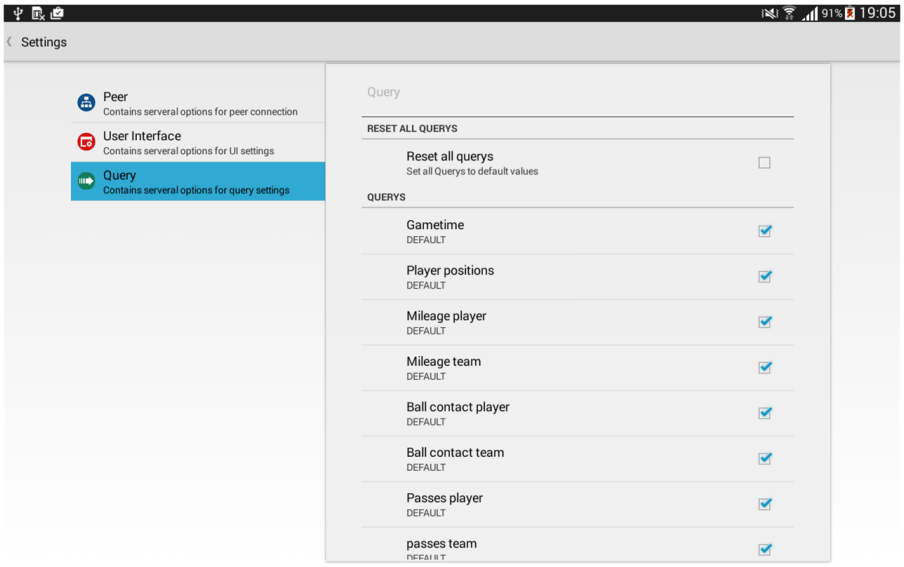


Abbildung 11.4: Einstellungsmenü - Anfragen

Das Einstellungsmenü für Anfragen ist speziell darauf ausgelegt, Einfluss auf die installierten Anfragen zu nehmen (siehe Abbildung 11.4). Dadurch kann bestimmt werden, welche Anfragen an das System gestellt werden sollen und wie die Verteilung der Anfrage im Netzwerk stattfinden soll.

11.1.3 Analyse - Hauptbildschirm

Nachdem der Nutzer sich für eine Sportart entschieden hat und das Laden der Anfragen abgeschlossen ist, wird der Hauptbildschirm der Analyse angezeigt (siehe Abbildung 11.5).

Auf der linken Seite befindet sich die Teamübersicht (1). Hier kann jeweils zwischen der Mannschaftsansicht des ersten und zweiten Teams per Tipp-Geste gewechselt werden. Neben dieser Option können auch einzelne Spieler selektiert werden, diese werden dann mittels eines gelben Kreises auf dem Spielfeld hervorgehoben. Per Tipp-Geste auf einen der Spieler im Team gelangt man zu einer kleinen Spielerstatistik, in der wesentliche Statistikwerte eines Spieler aufgelistet werden.
Im oberen Bereich befindet sich die Actionbar mit Aktionsknöpfen, die betätigt werden können. Diese Aktionen teilen sich auf in die Bereiche:

- Sportartabhängige Aktionen (2) und
- Sportartunabhängige Aktionen (3).

Über die sportartunabhängigen Aktionen können zum einen die Anfragen gesteuert werden (starten, pausieren, stoppen und löschen). Zum anderen kann zwischen den Ansichten Spielfeld-Vollansicht, Statistik-Vollansicht und dem geteilten Bildschirm gewählt werden. Die sportartabhängigen Aktionen beziehen sich auf die Sportart Fußball und beinhalten das Anzeigen einer Taktik-Tafel sowie das Auswählen und Anzeigen von Ketten und Laufwegen.
Die Spielfeldansicht besteht aus einer Ergebnisanzeige, der aktuellen Zeit, die seit Beginn der Analyse

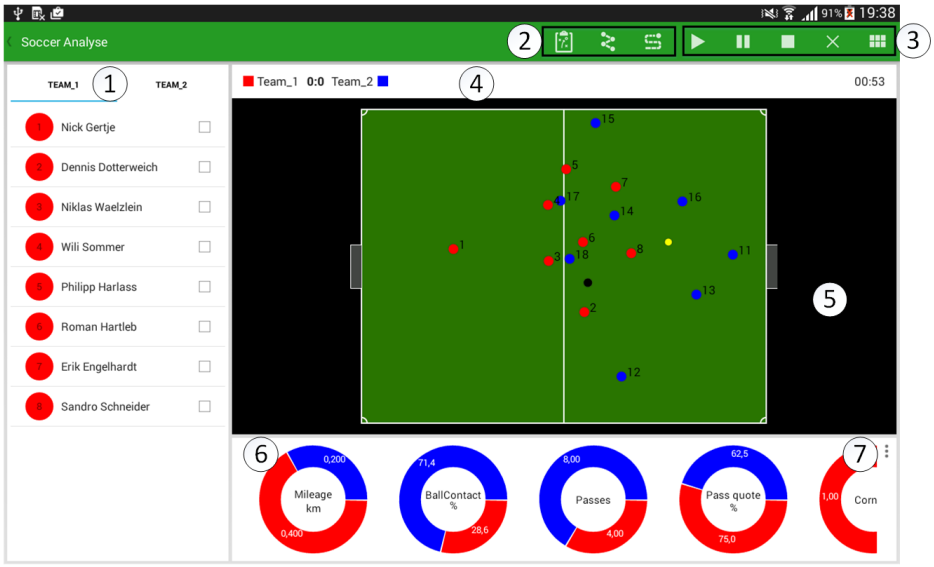


Abbildung 11.5: Hauptbildschirm

bereits vergangen ist (4) sowie dem Spielfeld selbst mit allen Feldelementen (5). Im unteren Bereich befindet sich die Anzeige der Teamstatistiken (6). Hier erhält der Nutzer einen ersten Eindruck über die einzelnen Teams und deren Leistungen. Mit einer Wisch-Geste können weitere Statistiken angezeigt werden. Einstellungsmöglichkeiten werden hier mittels des Buttons in (7) ermöglicht. In einem Dialog werden dann Optionen rund um die Teamstatistiken angeboten. Dabei können Statistiken ausgeblendet oder neu angeordnet werden.

11.1.4 Spielermenü

Mithilfe des Spielermenüs (siehe Abbildung 11.6) können Aktionen auf einzelne Spieler ausgeführt werden. Das Menü öffnet sich, sobald ein Spieler entweder in der Teamübersicht selektiert oder auf dem Spielfeld per Tipp-Geste ausgewählt wird.

Je nach Anzahl der ausgewählten Spieler (Anzahl wird in (1) angezeigt) werden verschiedene Funktionen sichtbar. Für einen einzelnen Spieler kann eine Heatmap angezeigt werden (2). Für einen bzw. mehrere ausgewählte Spieler besteht die Möglichkeit, neue Ketten zu erstellen (3). Es öffnet sich ein Dialog mit einem Textfeld, in das der Name der Kette eingetragen werden muss. Zusätzlich kann eine Farbe ausgewählt werden. Ist die Kette erstellt, können weitere Spieler hinzugefügt werden. Hierfür gibt es die Funktion `Add to...` (4). Sollten Ketten für das Team des ausgewählten Spieler vorhanden sein, so werden sie in einer Liste angezeigt. Letzte Funktion ist das Anzeigen des Laufwegs (5). Per Klick auf diesen Button wird der Laufweg auf dem Spielfeld für den Spieler / die Spieler angezeigt.

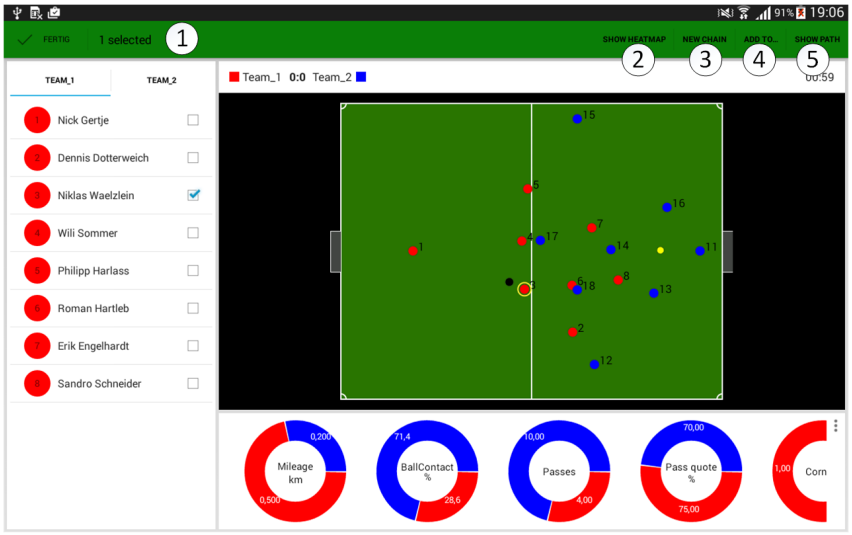


Abbildung 11.6: Spielermenü

11.1.5 Seitenmenü

Das Seitenmenü (siehe Abbildung 11.7) ist in erster Linie für die sportartabhängigen Aktionen gedacht. Per Klick auf den „Ketten“- (1) oder „Laufweg“-Button (2) öffnet sich das Seitenmenü und zeigt die bereits bestehenden Ketten bzw. Laufwege von beiden Team in einer Liste an. Zwischen den verschiedenen Reitern des Seitenmenüs (3) kann per Wisch-Geste gewechselt werden. Zu den einzelnen Listeneinträgen existieren weitere Aktionsbuttons (4). Diese ermöglichen das Ändern und Löschen der Ketten bzw. Laufwege. Zusätzlich können die Ketten ein- bzw. ausgeblendet werden.

Abbildung 11.7: Seitenmenü

11.1.6 Taktik-Tafel

Die Taktik-Tafel (siehe Abbildung 11.8) ist über einen Button (sportartabhängig) in der Actionbar erreichbar und ermöglicht es dem Nutzer, Zeichnungen auf dem Spielfeld vorzunehmen. Hierfür stehen verschiedene Zeichen-Funktionen zur Verfügung. Es ist möglich, Farbe und Stiftdicke festzulegen, mittels Radiergummi Linien zu entfernen oder mit Hilfe des Papierkorbs komplette Zeichnungen zu löschen.

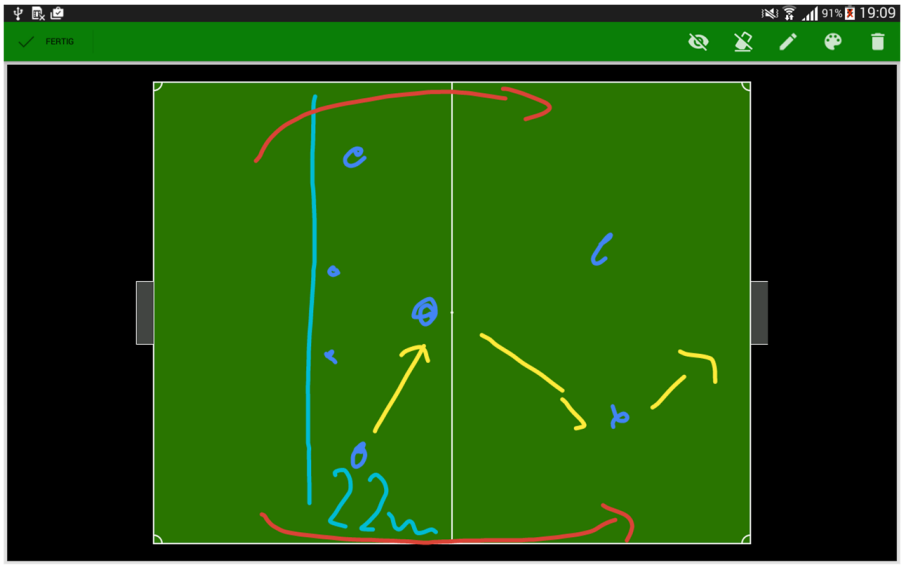


Abbildung 11.8: Taktik-Tafel

11.1.7 Statistiken

Wählt der Benutzer die Statistik-Vollbild-Ansicht über das entsprechende Ansicht-Menü, kann er über eine Tipp-Geste einen Spieler aus der Spielerliste auswählen und so seine Statistiken anzeigen lassen (siehe Abbildung 11.9). Diese Ansicht umfasst neben den wesentlichen Statistiken auch detailliertere Statistiken zu Pässen und Sprints.

11.1.8 Spielervergleich

Neben der Detailansicht eines einzelnen Spielers gibt es im Statistik-Vollbild auch die Möglichkeit, Spieler miteinander zu vergleichen (siehe Abbildung 11.10). Hierfür müssen genau zwei Spieler selektiert werden. Werden weniger oder mehr als zwei selektiert, wird kein Vergleich durchgeführt. Vergleiche sind zudem mannschaftsübergreifend möglich.

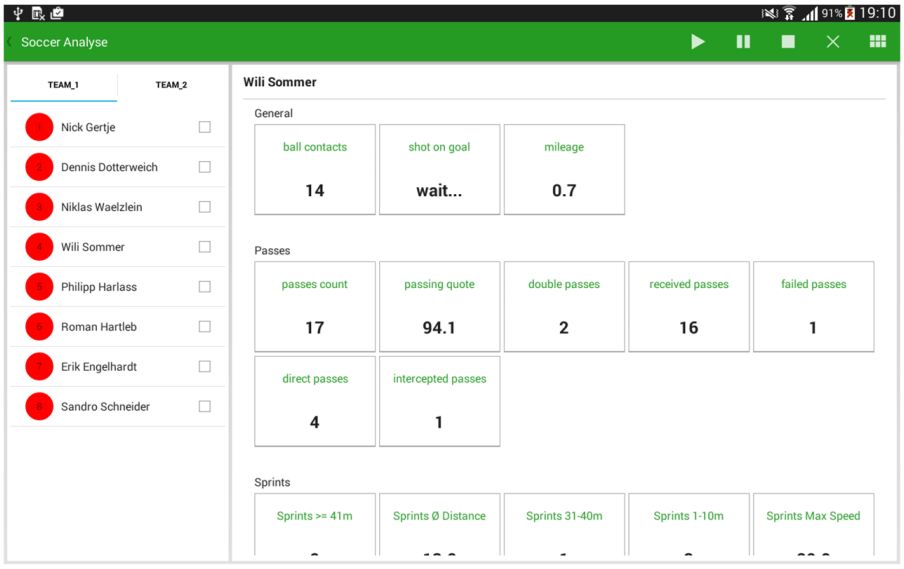


Abbildung 11.9: Statistiken im Vollbild

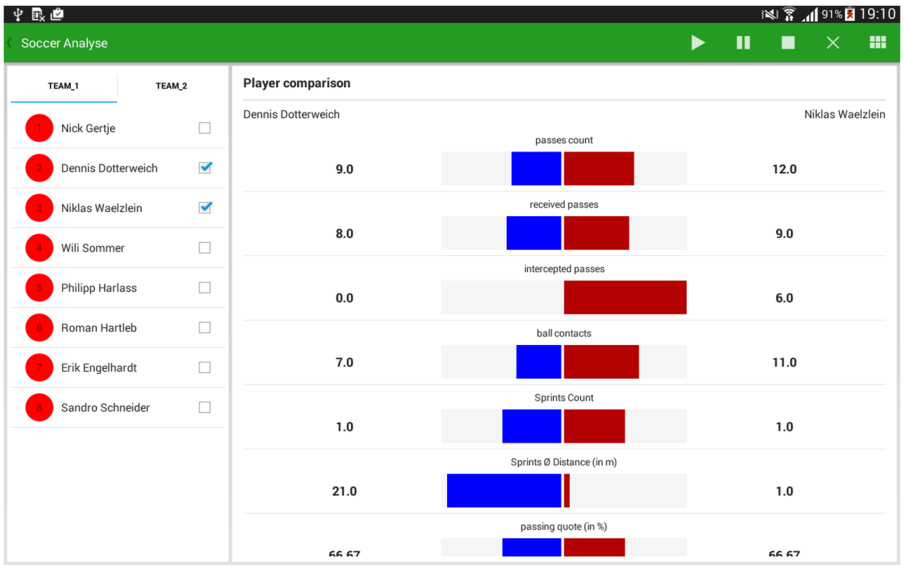


Abbildung 11.10: Spielervergleiche

11.2 Monitoring Application

In diesem Abschnitt werden die wichtigsten Funktionen der Monitoring App erläutert. Dazu gehören eine Konsole zum Absetzen von Steuerungsbefehlen, ein Log zur Ausgabe von Meldungen, eine Pingmap sowie die Darstellung eines verteilten Anfrageplans.

11.2.1 Konsole

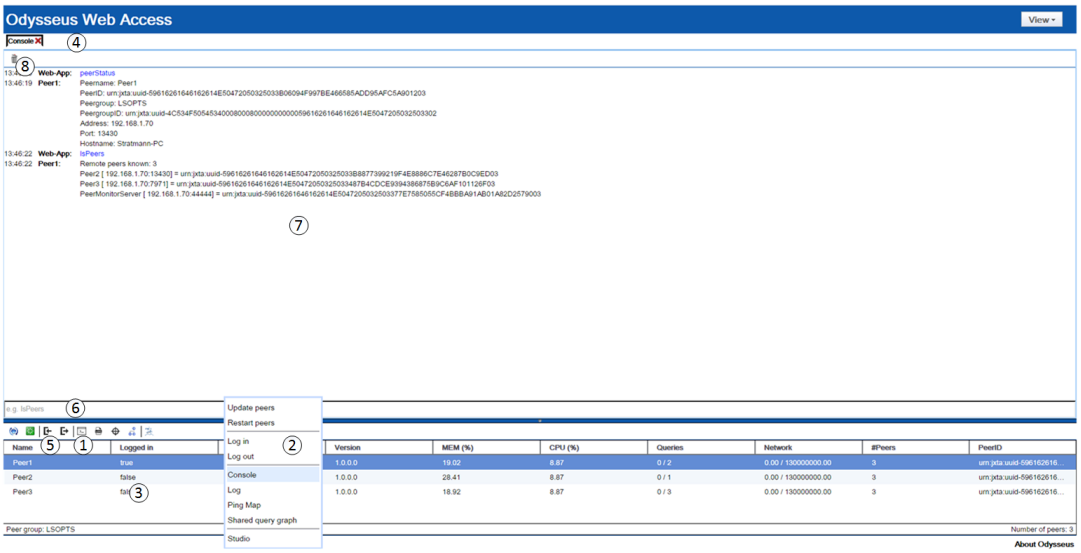


Abbildung 11.11: Konsole in der Monitoring Application

Zum Aufrufen der Konsole (siehe Abbildung 11.11) kann der Nutzer entweder den Button (1) aus der Toolbar betätigen oder das Kontextmenü (2) nutzen, welches über einen Rechtsklick in der Peer-Liste (3) aufgerufen werden kann. Die Konsole wird anschließend innerhalb eines Tabs (4) dargestellt. Um einen Befehl an einen Peer zu senden, muss sich der Nutzer zunächst bei dem Peer einloggen. Dazu kann wiederum ein Button (5) oder das Kontextmenü (2) genutzt werden. Anschließend kann der Nutzer einen Befehl (z.B. `peerStatus`) in das Texteingabefeld der Konsole (6) eingeben und diesen über die Enter-Taste an die in der Peerliste ausgewählten Peers senden. Der Befehl selber und die Antwort auf den Befehl werden anschließend in der Ansicht (7) über dem Texteingabefeld dargestellt. Um alle Einträge aus der Konsole zu löschen, kann der Button (8) aus der Toolbar der Konsole genutzt werden.

11.2.2 Log

Die Log-Ausgabe eines Peers kann aufgerufen werden, indem zunächst der Peer in der Peerliste (1) selektiert und anschließend der Button (2) aus der Toolbar betätigt wird. Alternativ kann auch das Kontextmenü der Peerliste (1) genutzt werden. Standardmäßig werden alle Log-Einträge eines Peers innerhalb einer Ansicht (3) dargestellt. Um die Log-Einträge nach einem Mindest-Log-Level zu filtern, kann die Selektionsliste (4) aus der Toolbar des Logs genutzt werden. Alternativ können die Log-Einträge auch nach bestimmten Schlüsselwörtern (5) gefiltert werden. Um die Filterfunktionen zu aktivieren, muss der Button (6) betätigt werden. Das Anzeigen der Log-Einträge kann zudem über einen Button (7) aktiviert bzw. deaktiviert werden. Um alle Logeinträge zu entfernen, kann ebenfalls ein Button (8) verwendet werden.

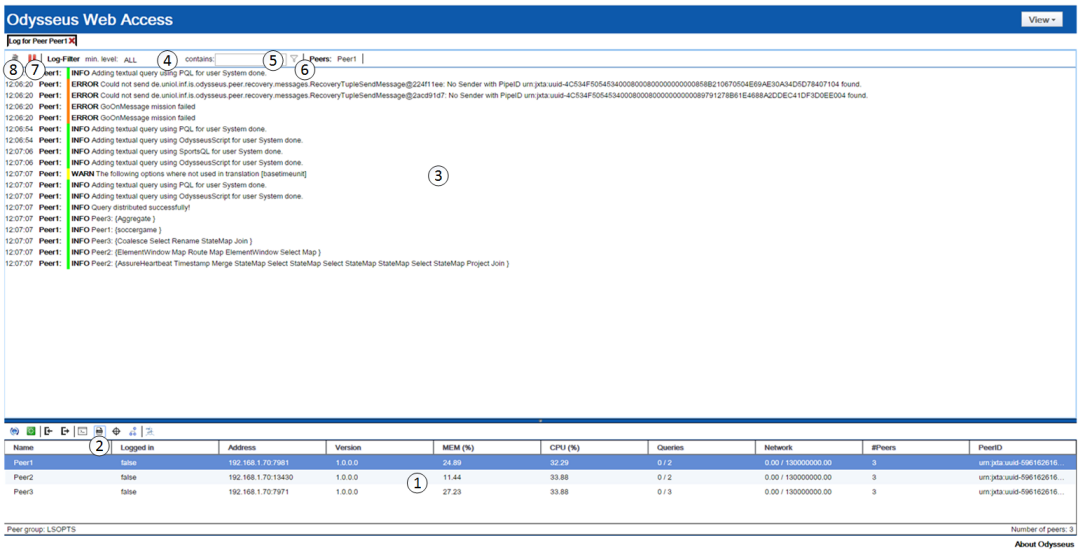


Abbildung 11.12: Log in der Monitoring Application

11.2.3 Ping-Map

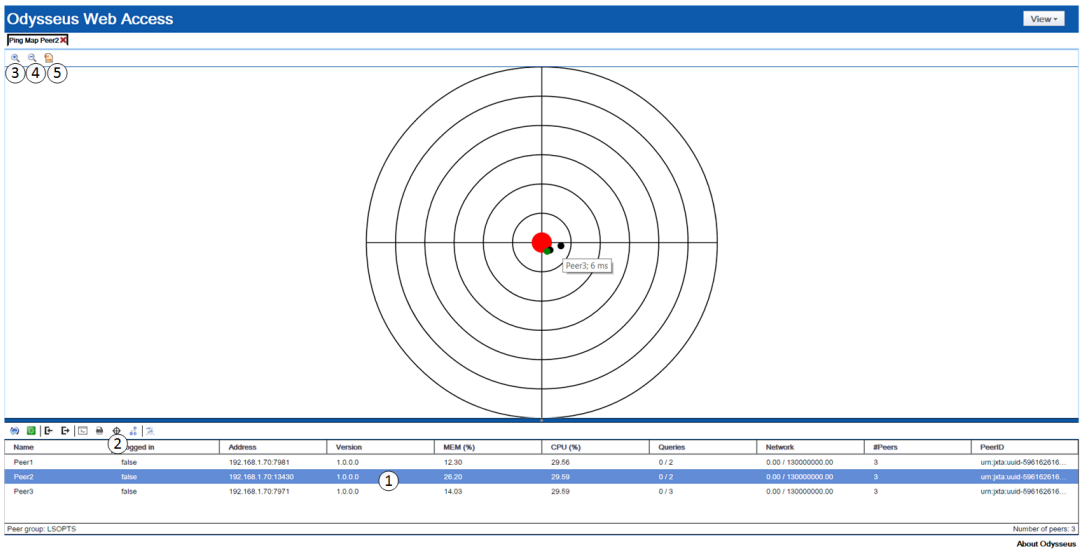


Abbildung 11.13: Ping Map in der Monitoring Application

Die Ping-Map eines Peers wird aufgerufen indem ein Peer in der Peerliste (1) selektiert wird. Anschließend kann die Ping-Map entweder über einen Button (2) in der Toolbar oder über das Kontextmenü der Tabelle (1) aufgerufen werden. Innerhalb der Ping-Map stellt der rote Punkt den lokalen Peer dar, zu dem die Ping-Map aufgerufen wurde. Die Monitoring Application wird als grüner Punkt dargestellt. Alle anderen Odysseus-Peers werden als schwarze Punkte dargestellt. Um mehr Informationen zu einem Punkt zu erhalten, muss die Maus über den Punkt bewegt werden, so dass ein Tooltip

mit dem Namen des Peers und dem Ping angezeigt wird. Über die Toolbar der Ping-Map kann die Ping-Map vergrößert (3) und verkleinert (4) werden. Des Weiteren kann die Ping-Map über einen Button (5) als SVG-Datei heruntergeladen werden.

11.2.4 Verteilter Anfrageplan

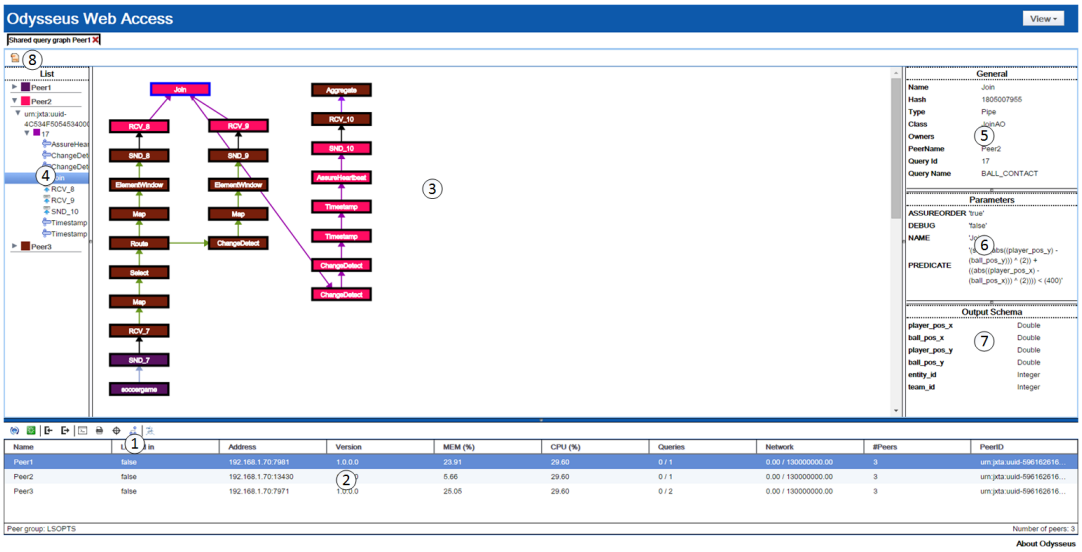


Abbildung 11.14: Verteilter Anfrageplan in der Monitoring Application

Um einen verteilten Anfrageplan darzustellen, kann wiederum ein Button (1) aus der Toolbar oder das Kontextmenü der Peerliste (2) genutzt werden. Dazu muss jedoch zunächst ein Peer in der Tabelle selektiert werden. Anschließend werden alle Shared-Query-Ids eines Peers innerhalb eines Pop-ups aufgelistet. Nach Auswahl einer Shared-Query-Id durch den Nutzer, wird der verteilte Anfrageplan zu der ausgewählten Shared-Query-Id angezeigt. Die Knoten des Anfrageplanes können beliebig ausgewählt und mit der Maus verschoben werden. Zum Hinein- und Herauszoomen kann das Mausrad verwendet werden, wobei die rechte Maustaste gehalten werden muss. Die Operatoren des Anfrageplans werden zudem in einer Liste (4) dargestellt. Um mehr Informationen zu einem Operator zu erhalten, kann ein Operator in der Liste (4) oder in dem Anfrageplan (3) ausgewählt werden. Anschließend werden zu dem Operator allgemeine Informationen (5), Parameter (6) und das Ausgabeschema (7) angezeigt. Des Weiteren kann der verteilte Anfrageplan über einen Button (8) in der Toolbar als SVG-Datei exportiert werden.

11.3 Developer Application

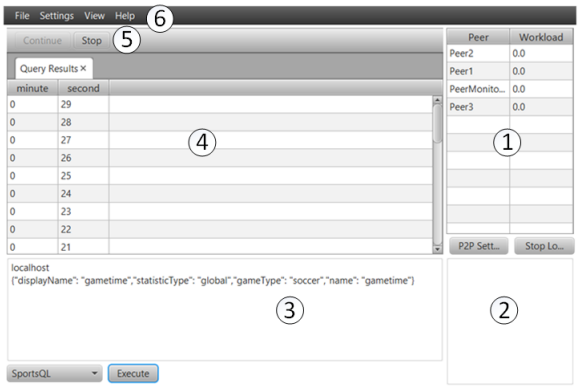


Abbildung 11.15: Developer Application

In der Developer Application werden alle gefundenen Peers innerhalb einer Liste angezeigt (1). Über einen Button (*P2P Settings*) unterhalb dieser Liste kann ein Pop-up geöffnet werden, in dem Einstellungen wie z.B. die Peer-Gruppe festgelegt werden können. Mit Hilfe eines weiteren Buttons unterhalb der Liste lässt sich die Peer-Suche starten und stoppen. Die Log-Nachrichten der App werden innerhalb eines Textfeldes (2) angezeigt. In einem weiteren Textfeld (3) können Anfragen definiert werden, welche über einen Button unterhalb des TextFeldes ausgeführt werden können. In der View über der Tabelle (4) wird nach Ausführen einer Anfrage automatisch eine Tabelle erstellt, in der die Ergebnisse der ausgeführten Anfrage angezeigt werden. Das Anzeigen der Ergebnisse kann gestartet und gestoppt werden (5). Über das Menü der App (6) sollten ursprünglich noch weitere Funktionen angeboten werden. Diese wurden jedoch nicht mehr realisiert, da die Entwicklung der App eingestellt wurde.

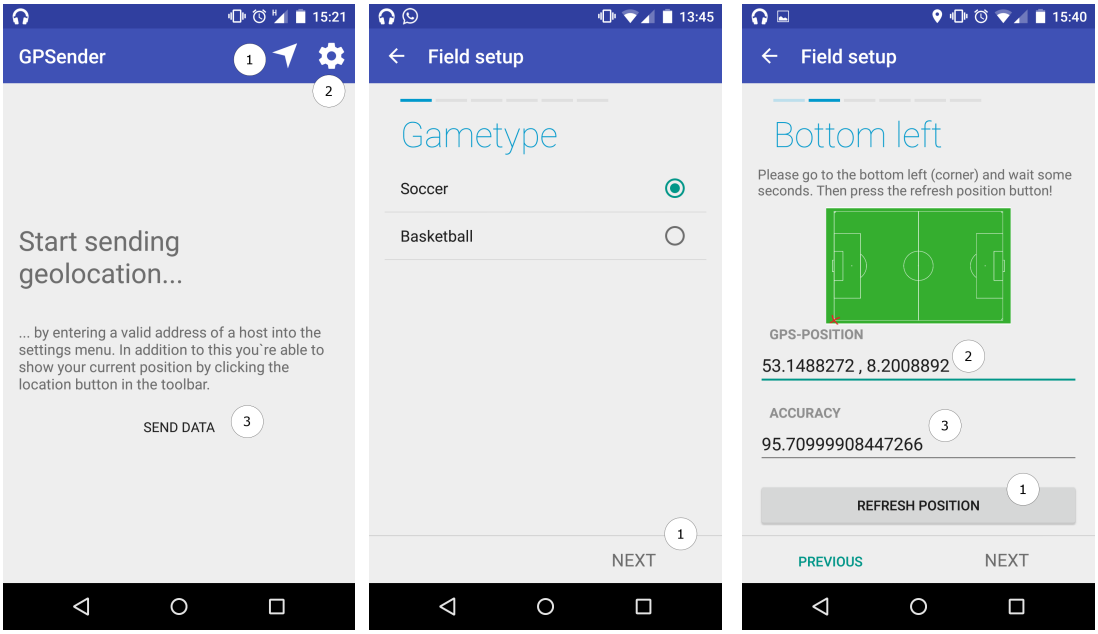
11.4 GPSender App

In den folgenden Abschnitten werden die Funktionen der GPSender App näher erläutert. Hierzu werden die einzelnen Einstellungsmöglichkeiten, der Field Setup Assistent und das Senden von Positionsdaten beschrieben. Wird die GPsender App gestartet, gelangt man in den Hauptbildschirm der Anwendung (siehe Abbildung 11.16 (a)). Über diesen Bildschirm sind sämtliche Funktionen und Einstellungen der App zu erreichen.

11.4.1 Field Setup

Mit dem Button (1a) wird der Field Setup Assistent gestartet, der dem Anwender dabei hilft, die benötigten DDC Werte für das Spielfeld zu ermitteln. Wurde der Field Setup Assistent gestartet, muss zunächst die Sportart ausgewählt werden. Zurzeit werden die Sportarten Fußball und Basketball unterstützt. Folgende Beschreibungen beziehen sich auf die Sportart Fußball.

Wurde die Sportart mit einem Klick auf den Button (1b) bestätigt, wird der Nutzer dazu aufgefordert, sich in die untere Linke-Ecke des Spielfeldes zu bewegen. Befindet sich der Anwender in der linken unteren Ecke, klickt er auf den Button (1c) die GPS-Position erfasst. Im Textfeld (2c) wird die aktuell erfasste GPS-Position angezeigt. Das Textfeld (3c) zeigt die Genauigkeit der erfassten GPS-Position an. Je niedriger der Wert ist, desto genauer wurde die Position ermittelt. Durch ein erneutes Klicken auf den Button (1c) wird eine erneute Positionsbestimmung durchgeführt. Wurde eine ausreichende Genauigkeit erreicht, kann mit der Einrichtung fortgeschritten werden. Nach der gleichen Vorgehensweise werden die Positionen für die obere linke und die obere rechte Ecke bestimmt. Zum Beenden des Assistenten kann in der nächsten Ansicht optional ein Name und eine E-Mail Adresse angegeben werden. Im letzten Schritt des Field Setup Assistenten, erfolgt eine Zusammenfassung der erfassten Daten. Diese Daten können über den Button *Send data* an eine E-Mail Adresse gesendet werden. Zusätzlich werden die erfassten Daten in das passende Format für das DDC konvertiert und können somit bequem in das DDC eingefügt werden.



(a) Hauptbildschirm *(b) Sportartauswahl* *(c) Spielfeld erfassen*

Abbildung 11.16: GPSender Application

11.4.2 Einstellungen

Über den Button (2a, Abbildung 11.16) gelangt man in das Einstellungsmenü der Anwendung. So lassen sich folgende Einstellungen vornehmen:

ID of device (1)
Legt die Spieler-ID fest. Diese muss eindeutig sein und muss mit den Daten im DDC übereinstimmen.

Name of user (2)
Legt den Spielernamen fest. Dieser Namen wird nur zur Anzeige im GPServer benutzt. Und ist daher optional.

Host address (3)
Legt die IP-Adresse des Rechners fest, an den die Positionsdaten geschickt werden sollen. Im Normalfall wird hier die IP-Adresse vom GPServer angegeben.

Port (4)
Legt den Port fest, auf dem der GPServer erreicht werden kann. Standardmäßig kann hier der Port 56565 angegeben werden.

Type (5)
Legt fest, auf welche Art die Positionsdaten erfasst werden sollen. Zur Verfügung stehen die Optionen GPS und Gauß-Krüger.

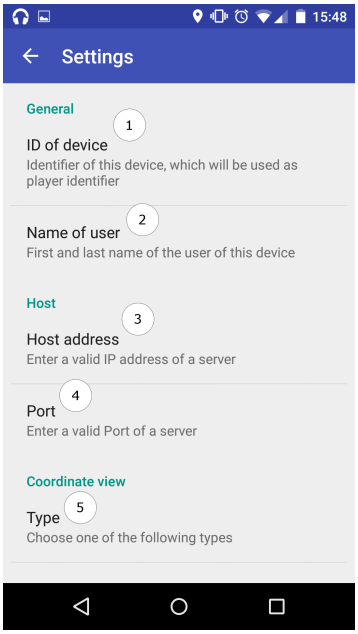
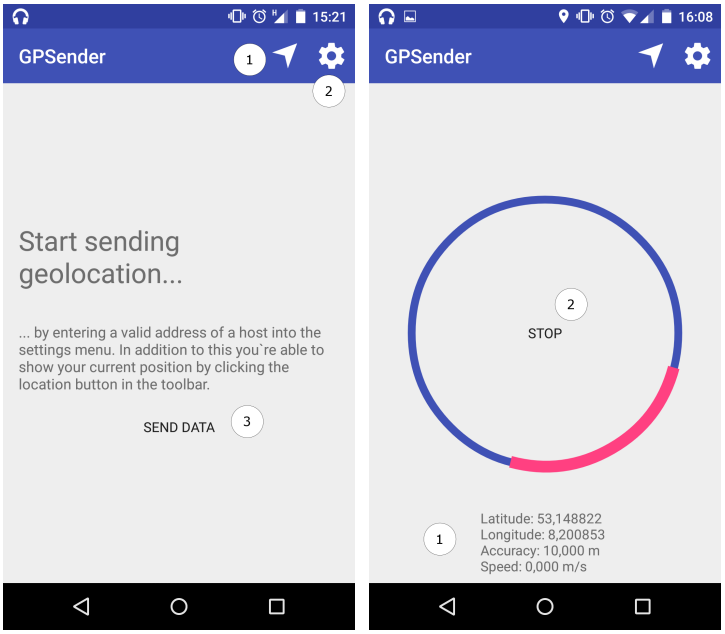


Abbildung 11.17: GPSender Application Einstellungen

11.4.3 Hauptbildschirm

Wurde die Anwendung korrekt über das Einstellungsmenü konfiguriert, kann über den Hauptbildschirm mit einem Klick auf den Button (3) das Senden der Positionsdaten gestartet werden. Im unteren Bereich des Bildschirms (1b) werden anschließend die aktuellen Positionsdaten angezeigt. Durch einen Klick auf den Button (2b) kann die Übermittlung von Positionsdaten gestoppt werden. Sollte das Gerät mit der GPSender App in den Ruhemodus versetzt werden läuft die Anwendung im Hintergrund weiter. Dies hat den Vorteil, dass die App kontinuierlich aktuelle Positionsdaten an den GPServer sendet.



(a) Hauptbildschirm *(b) Positionsdaten senden*

Abbildung 11.18: GPSender Application Aktive

12 Zusammenfassung und Ausblick

Die mit diesem Dokument vorliegende Abschlussdokumentation beinhaltet Grundlagen, Konzepte und Erläuterungen zu den Ergebnissen und dem Vorgehen der Projektgruppe LSOP. Dabei wird vor allem das entstandene System Herakles genau erläutert.

In dem 1. Kapitel wird eine Einleitung in das Projekt gegeben und die Motivation und die Ziele dargelegt. Kernziel der Gruppe war es, Live-Sportanalysen mit einem DSMS zu ermöglichen und die Ergebnisse ansprechend auf einem Tablet zu visualisieren. Anschließend folgen in Kapitel 2 zunächst die wissenschaftlichen Grundlagen des Projekts, darunter zum Beispiel Datenströme, Peer Computing, Recovery und Load-Balancing. Danach geht es mit den Anforderungen an das zu erstellende System Herakles weiter. In Kapitel 3 werden dafür sowohl die fachlichen Anforderungen als auch die daraus entstandenen Epics und User Stories des Scrum-Prozesses beschrieben. Die Anforderungen wurden vom Project Owner, in diesem Fall dem Projektgruppenbetreuer, gestellt und im laufenden agilen Scrum-Prozess erweitert, angepasst und neu bewertet.

In den folgenden Kapiteln folgen Details zu Herakles. Das Kapitel 4 gibt zunächst eine Übersicht über die Systemarchitektur. Dort wird gezeigt, wie Herakles in die drei Hauptbestandteile Sensorik, Datenanalyse und Visualisierung aufgeteilt ist. Nach diesem Überblick werden Teilbereiche des Systems in Kapitel 5 auf konzeptioneller Ebene detailliert erklärt. Die Ideen und Probleme einzelner Bereiche wie etwa dem Load-Balancing werden dargestellt und analysiert. Wie diese Konzepte für Herakles umgesetzt wurden, wird im nächsten Kapitel beschrieben. Das 6. Kapitel zeigt dazu die Implementierung der einzelnen Bestandteile des Gesamtsystems, wozu z.B. die Datenabstraktion und die Coach Application gehören.

Um zu überprüfen, wie gut die einzelnen Bestandteile von Herakles funktionieren, wurden diese evaluiert. Wie die Projektgruppe für die einzelnen Bereiche dafür vorgegangen ist und wie die Ergebnisse sind, wird in Kapitel 7 beschrieben. Ein Ergebnis war z.B., dass die Berechnung der Statistiken in einem P2P-Netzwerk sehr viel Performance kostet, dafür aber durch das Recovery mehr Sicherheit gibt.

Die folgenden zwei Kapitel beschäftigen sich mit dem Vorgehen des Projekts. In Kapitel 8 wurden dafür unter anderem das Vorgehensmodell Scrum und die Projektplanung erklärt. Das folgende Kapitel 9 beschreibt die IT Infrastruktur, die von der Gruppe eingesetzt wurde. So wurden etwa die Softwarelösungen von Atlassian wie Confluence und Jira eingesetzt, um die Zusammenarbeit zu koordinieren.

Für mögliche Anwender von Herakles sind vor allem die letzten beiden Kapitel 10 und 11 interessant. Im ersten wird beschrieben, wie Herakles, beginnend vom Sensorsystem über die Datenanalyse bis zur Coach Application, eingerichtet werden kann. Das folgende Kapitel beinhaltet Benutzerhandbücher für die drei Anwendungen „Coach Application“, „Monitoring Application“ und „Developer Application“.

12.1 Fazit

LSOP war ein umfangreiches und sehr vielseitiges Projekt. Es hat den Teammitgliedern in sehr unterschiedlichen Bereichen viel Können abverlangt und dabei ebenso viele neue Kenntnisse geschaffen. Die Vielseitigkeit begründet sich auf der Aufgabe, die der Projektgruppe gestellt wurde. Eine Live-

Sportanalyse mittels Datenströmen in einem P2P-Netzwerk durchzuführen beinhaltet bereits Aufgaben in unterschiedlichen Bereichen wie der Datenstromverarbeitung und der Sportanalyse. Dazu kamen jedoch noch weitere Schwerpunkte wie die Entwicklung einer Android-Applikation zur Visualisierung von Statistiken und einer Möglichkeit, Positionsdaten von Spielern live aufnehmen zu können.

Durch den Ehrgeiz der Gruppe, eine nutzbare Lösung zu schaffen, die das vorgegebene und mit der Zeit erweiterte Ziel erreicht, entstand mit Herakles ein System, das die Potentiale der verwendeten grundlegenden Technologien in diesem Bereich zeigt. Die Kernaufgabe, nämlich die Analyse von Sportdaten und das Berechnen von Statistiken aus ebendiesen in einem DSMS, wurde mit Erfolg gelöst. Mit Herakles ist es möglich, Sensordaten von unterschiedlichen Sensorsystemen zu nutzen und live zu analysieren. Die Berechnung in einem verteilten P2P-Netzwerk funktioniert ebenfalls, bringt jedoch deutliche Einbußen in der Performance mit sich, wie die Evaluation gezeigt hat. Dennoch wurden mit dem Load-Balancing und Recovery zwei Techniken erfolgreich eingebaut, die die Zuverlässigkeit erhöhen und damit Vorteile im Vergleich mit einem monolithischen DSMS ermöglichen.

Die App für den Trainer, auf dem die Statistiken visualisiert werden und die sinnvolle Funktionen für die Arbeit mit ebendiesen bietet, war eine weitere Kernaufgabe des Projekts. Mit der Coach App wurde eine umfangreiche Anwendung erstellt, die sowohl das Betrachten von Statistiken als auch die Interaktion mit diesen ermöglicht. Die bei der Entwicklung dieser Anwendung erlangten Kenntnisse in der Android Programmierung konnten zudem genutzt werden, um ein während des Projekts neu hinzugekommenes Ziel, nämlich das Aufzeichnen von Positionsdaten mit Smartphones, zu realisieren. Auch weitere zusätzliche Aufgaben wurden gemeistert: So wurde eine Monitoring Application entwickelt, mit der sich ein verteiltes P2P-Netzwerk überwachen und steuern lässt.

Diese große Menge an Arbeit lässt sich nur dann bewältigen, wenn die Teammitglieder engagiert und koordiniert zusammenarbeiten. Dies wurde durch die positive Einstellung der Mitglieder zum Projekt erreicht, die durch das gewählte Vorgehensmodell unterstützt wurde. Durch die sinnvolle Verwendung von Scrum konnten die Aufgaben geplant und verteilt werden, sodass ein kontinuierlicher Fortschritt zu erkennen war.

Das Projekt war so von verschiedenen Perspektiven betrachtet aus sehr erfolgreich. Das vorgegebene Ziel wurde klar erreicht und auch zusätzlich aufkommende Aufgaben bewältigt. Das Team hat sowohl fachlich als auch überfachlich einiges gelernt. Und gerade das Lernen neuer Fähigkeiten war das wichtigste Ziel der Projektgruppe.

12.2 Ausblick

Auch wenn die Ziele des Projektes erreicht wurden, gibt es Raum für Verbesserungen und Optimierungen verschiedener Art.

Bisher wurde Herakles nur mit dem selbst entwickelten Sensorsystem „GPSender“ und anderen Datenquellen genutzt, nicht jedoch mit kommerziellen Sport-Sensorsystemen. Deshalb sollte Herakles mit einem Sensorsystem als Datenquelle ausgeführt werden, um es dann in einem realen Szenario auf dem Spielfeld zu testen.

Das Recovery bietet noch vielfältige Möglichkeiten der Verbesserung und Erweiterung. So bestehen noch kleinere Fehler oder ungelöste Probleme für das allgemeine Recovery sowie für das Active-Standby im Besonderen. Einige Ansatzpunkte werden im Fazit der Recovery Evaluation in

Kapitel 7 genannt, jedoch wäre hier noch eine umfangreichere Evaluation hilfreich, um weiterführende Erkenntnisse zu erlangen. Sind diese Probleme gelöst, können weitere Recovery-Strategien implementiert werden, um flexibler in der Anwendung des Recovery zu werden. Das Upstream Backup bietet zum Beispiel die Möglichkeit, den Tupelverlust zu minimieren ohne eine aktive doppelte Berechnung wie beim Active-Standby durchzuführen. Darauf aufbauend könnte dann eine dynamische Nutzung der verschiedenen Strategien implementiert werden. Dadurch werden die Query-Parts optimal behandelt, je nachdem ob sie zustandslos oder zustandsbehaftet sind.

Das Load-Balancing kann weiter verbessert werden, indem bessere Allokationsstrategien entwickelt werden. Dadurch können Fälle, bei denen ungünstig allokiert wird und es damit zu schlechteren Ergebnissen kommt, vermieden werden. Weiterhin können weitere Load-Balancing-Strategien, wie das Kräftemodell, implementiert werden.

Auch auf Seiten der Datenanalyse gibt es Optionen zur Verbesserung. Zunächst können hier weitere Anfragen zu Statistiken erstellt werden. Beispiele wären Flanken oder durchschnittliche Passlänge. Es sollte die Erfolgsrate noch genauer analysiert werden, um zu ermitteln, wie viele der erkannten Ereignisse „false positives“ sind. Daraufhin können die Anfragen optimiert werden. Eine weitere Möglichkeit, die Genauigkeit der Anfragen zu erhöhen, ist die Erkennung von Spielunterbrechungen und das Stoppen der Anfragen bei diesen. Momentan wird während der Ausführung der Anfragen ebenfalls nicht behandelt, dass die Mannschaften nach der Halbzeit die Seiten wechseln. Hier sollte es ebenfalls eine automatische Erkennung geben.

In der Coach App ist die reine numerische Anzeige der Statistiken für Spieler nicht optimal, um schnell zu erkennen, wie gut oder schlecht der Spieler spielt. Im Basketball gibt es dafür zum Beispiel einen „Performance-Index“. Im Fußball muss recherchiert werden, ob es einen vergleichbaren Wert gibt. Aber auch ohne einen solchen Wert könnten die Einzelstatistiken grafisch übersichtlicher gestaltet werden. Um die Abhängigkeit von Android zu verringern, könnte die Coach App auch auf andere Plattformen wie z.B. iOS, Windows Phone oder gar als Webanwendung portiert werden. In diesem Zusammenhang müsste allerdings geprüft werden, inwiefern sich der LSOP Service auf anderen Plattformen nutzen ließe.

A Anhang

A.1 Quellendefinitionen

```
#PARSER PQL
#DEFINE sourcename soccergame
#DEFINE firstddcKey "sensorid."
#DEFINE secondddcKey ",entity_id"
#DEFINE extendedFieldParameter 3000

#RUNQUERY
etwSoccerGame = TIMESTAMPTOPAYLOAD(ACCESS({
                        transport = 'tcpclient',
                        source = 'etwSoccerGame',
                        datahandler = 'tuple',
                        wrapper = 'GenericPush',
                        protocol = 'simplecsv',
                        options=[
                          ['port', '19991'],
                          ['host', '127.0.0.1'],
                          ['Delimiter', ','],
                          ['KeepDelimiter', 'false'],
                          ['Charset', 'utf-8'],
                          ['BaseTimeUnit', '${sensors,basetimeunit}']
                        ],

                        schema=[
                          ['sid', 'String'],
                          ['name', 'String'],
                          ['latitude', 'String'],
                          ['longitude', 'String'],
                          ['altitude', 'String'],
                          ['accuracy', 'String']
                        ]
                      }
                    )
                    )


enrichEtwSoccerGame = STATEMAP({
expressions = [
                  ['toInteger(fromDDC(concat(concat(${firstddcKey},
                     ↪ sid),${secondddcKey})))','entity_id'],
                  ['toInteger(fromDDC(concat(concat(${firstddcKey},
                     ↪ sid),",team_id")))','team_id'],
```

```
41                  ['meta_valid_start','ts'],
42                  ['gpsToLocalCoord(ToDouble(latitude),␣ToDouble(
                        ↪ longitude),␣"x")', 'x'],
43                  ['gpsToLocalCoord(ToDouble(latitude),␣ToDouble(
                        ↪ longitude),␣"y")', 'y'],
44                  ['ToDouble(altitude)', 'z']],
45                group_by= ['sid']
46 }, etwSoccerGame)
47
48
49 /// substract first timestamp from ts to start with timestamp 0
50 min_ts = AGGREGATE({
51                  name='AGG',
52                  aggregations=[ ['FIRST', 'ts', 'min_ts', 'double']
                        ↪ ]
53              }, enrichEtwSoccerGame)
54 min_ts_cd = CHANGEDETECT({
55                  attr = ['min_ts'],
56                  DELIVERFIRSTELEMENT = 'true'
57              }, min_ts)
58 debs_min_ts = ENRICH({PREDICATE = 'true'}, min_ts_cd,
      ↪ enrichEtwSoccerGame)
59 zero_debsstream = MAP({
60                  expressions = [
61                          'entity_id',
62                          'team_id',
63                                  ['ts␣-␣min_ts', 'ts'],
64                                  'y',
65                                  'x',
66                                  'z'
67                  ]
68                }, debs_min_ts)
69
70
71
72
73 /// reducedLoad = REDUCELOAD({TIMEVALUE = [${sensors,timebetween},
      ↪ '${sensors,basetimeunit}'],GROUP_BY = ['entity_id'],ASSUMEBTU
      ↪  = '${sensors,basetimeunit}'},enrichEtwSoccerGame)
74
75  /// calculate euclidean distance
76  /// calculate delta T
77 spaceDistance = STATEMAP({
78                expressions = ['entity_id','team_id', 'ts', 'x', 'y',
                      ↪  'z',
```

```
79                      ['sqrt(((__last_1.x - x)^2)+((__last_1.y - y)
                           ↪ ^2)+((__last_1.z - z)^2))','
                           ↪ euclideanDistance'],
80                      ['(ts - __last_1.ts) / 1000000 ', 'deltaT']],
81              group_by= ['entity_id']
82             }, zero_debsstream)
83
84  /// calculate velocity v
85 speed = MAP({
86              expressions = ['entity_id', 'team_id', 'ts', 'x', 'y'
                   ↪ , 'z',
87              ['euclideanDistance / deltaT', 'v']]
88              }, spaceDistance)
89
90  /// v to micrometers
91 /// calculate delta V
92 /// calculate delta T
93 speedDifference = STATEMAP({
94              expressions = ['entity_id', 'team_id', 'ts', 'x', 'y'
                   ↪ , 'z',
95                      ['v * 1000', 'v'],
96                      ['v - __last_1.v','deltaV'],
97                      ['(ts - __last_1.ts) / 1000000 ', 'deltaT']],
98              group_by= ['entity_id']
99             }, speed)
100
101 /// convert x into double
102 /// convert y into double
103 /// convert z into double
104 /// calculate acceleration
105 outputGPSStream = MAP({
106             expressions = ['entity_id', 'team_id', 'ts',
107             ['ToDouble(x)', 'x'],
108             ['ToDouble(y)', 'y'],
109             ['ToDouble(z)', 'z'],
110             'v',
111             ['deltaV / deltaT', 'a']]
112             }, speedDifference)
113
114
115 ///Reduce data
116 convertedTimestamp = MAP({EXPRESSIONS = ['entity_id','team_id',['
    ↪ Floor(ts*toLong(${sensors,conversionfactor}))','ts'],'x','y',
    ↪ 'z','v','a']}, outputGPSStream)
117 setTimestamp = Timestamp({start='ts'},convertedTimestamp)
118 reAddedTimestamp = TimestampToPayload(setTimestamp)
```

```
119 ${sourcename} := MAP({EXPRESSIONS = ['entity_id','team_id',['
    ↪ meta_valid_start','ts'],'x','y','z','v','a']},
    ↪ reAddedTimestamp)
```

Listing A.1: Quellendefinition für die GPServer-Quelle

```
#PARSER PQL
#DEFINE sourcename soccergame
#DEFINE firstddcKey "sensorid."
#DEFINE secondddcKey ",entity_id"
#DEFINE extendedFieldParameter 3000
#RUNQUERY
/// The base stream
 debsstream = PROJECT({
                  ATTRIBUTES = ['sid', 'ts', 'x', 'y', 'z', 'v', 'a
                     ↪ ']
                },
                CSVFILESOURCE({FILENAME = 'Pfad zur CSV-Datei.',
 SOURCE = 'soccergame',
 SCHEMA = [
        ['sid', 'integer'],
        ['ts', 'starttimestamp'],
        ['y', 'integer'],
        ['x', 'integer'],
        ['z', 'integer'],
        ['v', 'integer'],
        ['a', 'integer'],
        ['vx', 'integer'],
        ['vy', 'integer'],
        ['vz', 'integer'],
        ['ax', 'integer'],
        ['ay', 'integer'],
        ['az', 'integer']
 ],
 options=[
        ['Delimiter',','],
        ['debug','true'],
        ['dumpEachLine','1000000'],
        ['measureEachLine','100000'],
        ['lastLine','49576078'],
    ['baseTimeUnit', 'MICROSECONDS']
    ]}))


/// calculate the datarate
datarateAdded = DATARATE({UPDATERATE = 100},  debsstream)

```

```
41
42 /// substract first timestamp from ts to start with timestamp 0
43 min_ts = AGGREGATE({
44                 name='AGG',
45                 aggregations=[ ['FIRST', 'ts', 'min_ts', 'double']
                       ↪ ]
46             }, datarateAdded)
47 min_ts_cd = CHANGEDETECT({
48                 attr = ['min_ts'],
49                 DELIVERFIRSTELEMENT = 'true'
50           }, min_ts)
51 debs_min_ts = ENRICH({PREDICATE = 'true'}, min_ts_cd, datarateAdded
       ↪ )
52 zero_debsstream = MAP({
53                 expressions = [
54                     'sid',
55                     ['ts - min_ts', 'ts'],
56                     'y',
57                     'x',
58                     'z',
59                     'v',
60                     'a'
61                 ]
62               }, debs_min_ts)
63
64 ///Port 0: Balls Port 1: no Balls
65 split_balls = ROUTE({PREDICATES = ['isSensorBall(sid, "sid")','
       ↪ isSensorLeg(sid, "sid")']}, zero_debsstream)
66 balls_in_game_field = SELECT({predicate='y>${field,ymin}-${
       ↪ extendedFieldParameter} AND y<${field,ymax}+${
       ↪ extendedFieldParameter} AND x>${field,xmin}-${
       ↪ extendedFieldParameter} AND x<${field,xmax}+${
       ↪ extendedFieldParameter}'},split_balls)
67 ball_source = MAP({
68               expressions = [
69               ['toInteger(fromDDC(concat(concat(${firstddcKey},sid)
                     ↪ ,${secondddcKey})))','entity_id'],
70               ['toInteger(fromDDC(concat(concat(${firstddcKey},sid)
                     ↪ ,",team_id")))','team_id'],
71               'ts',
72               ['ToDouble(x)', 'x'],
73               ['ToDouble(y)', 'y'],
74               ['ToDouble(z)', 'z'],
75               ['ToDouble(v)', 'v'],
76               ['ToDouble(a)', 'a']
77               ]
```

```
},
balls_in_game_field)

debsstreamEntityID = MAP({
                    expressions = [
                                        ['toInteger(fromDDC(concat(
                                           ↪ concat(${firstddcKey},sid
                                           ↪ ),${secondddcKey})))','
                                           ↪ entity_id'],
                                        ['toInteger(fromDDC(concat(
                                           ↪ concat(${firstddcKey},sid
                                           ↪ ),",team_id")))','team_id
                                           ↪ '],
                                        'x',
                                        'y',
                                        'z',
                                        'v',
                                        'a',
                                        'ts'
                                    ]
                }
,1:split_balls)
spaceDistance = STATEMAP({
               expressions = ['entity_id', 'team_id',
                 ['(__last_1.x + x)/2','neu_x'],
                 ['(__last_1.y + y)/2','neu_y'],
                 ['(__last_1.z + z)/2','neu_z'],
                 ['(__last_1.v + v)/2','neu_v'],
                 ['(__last_1.a + a)/2','neu_a'],
                 'ts'
                 ],
               group_by= ['entity_id']
              },
debsstreamEntityID)

direct_source = MAP({
               expressions = [
               ['toInteger(entity_id)','entity_id'],
               ['toInteger(team_id)','team_id'],
               'ts',
               ['ToDouble(neu_x)', 'x'],
               ['ToDouble(neu_y)', 'y'],
               ['ToDouble(neu_z)', 'z'],
               ['ToDouble(neu_v)', 'v'],
               ['ToDouble(neu_a)', 'a']]
},
```

```
spaceDistance)

mergeDataWithBalls = UNION({STRICTORDER = false}, ball_source,
    ↪ direct_source)

///Reduce data
convertedTimestamp = MAP({EXPRESSIONS = ['entity_id','team_id',['
    ↪ Floor(ts/toLong(${sensors,conversionfactor}))','ts'],'x','y',
    ↪ 'z','v','a']}, mergeDataWithBalls)
setTimestamp = Timestamp({start='ts'},convertedTimestamp)
reAddedTimestamp = TimestampToPayload(setTimestamp)
properTimestamp= MAP({EXPRESSIONS = ['entity_id','team_id',['
    ↪ meta_valid_start','ts'],'x','y','z','v','a']},
    ↪ reAddedTimestamp)
${sourcename} ::= REDUCELOAD({TIMEVALUE = [${sensors,timebetween},
    ↪ '${sensors,basetimeunit}'],ASSUMEBTU='${sensors,basetimeunit}
    ↪ ',GROUP_BY = ['entity_id']},properTimestamp)
}
```

Listing A.2: Quellendefinition für die DEBS Quelle

A.2 Distributed Data Container

```
############################
# DDC for sports analysis #
############################


# general game information
leftgoalteamid = 1
rightgoalteamid = 2

# the boundaries of the field
# field,xmin = <xmin>
# field,xmax = <xmax>
# field,ymin = <ymin>
# field,ymax = <ymax>

field,ymin = -50.0
field,ymax = 52489.0
field,xmin = -33960.0
field,xmax = 33965.0

# the goal areas
# goalarea.<left or right>,xmin = <xmin of left or right goal area>
# goalarea.<left or right>,xmax = <xmax of left or right goal area>
```

```
24 # goalarea.<left or right>,y = <y of left or right goal area>
25 # goalarea.<left or right>,zmax = <zmax of left or right goal area>
26
27 goalarea.left,ymin = 22560.0
28 goalarea.left,ymax = 29880.0
29 goalarea.left,x = -33968.0
30 goalarea.left,zmax = 2440.0
31
32 goalarea.right,ymin = 22578.5
33 goalarea.right,ymax = 29898.5
34 goalarea.right,x = 33941.0
35 goalarea.right,zmax = 2440.0
36
37 # the sensors
38 # sensoridlist = <sensorid1>,...,<sensoridn>
39 # sensoridseparator = <separator>
40 # sensors,basetimeunit = <basetimeunit> e.g. Milliseconds
41 # sensors,conversionfactor = <conversionfactor> Factor to convert
      ↪ between time units in sensor and basetimeunit (used in
      ↪ intermediate schema)
42 # sensors,timebetween = <timebetween> How much time units (in
      ↪ basetimeunits) should pass between each tuple from one sid
43
44 sensoridlist =
      ↪ 4,8,10,12,13,14,16,19,23,24,28,38,40,44,47,49,52,53,54,
45 57,58,59,61,62,63,64,65,66,67,68,69,71,73,74,75,
46 88,97,98,99,100,105,106
47 sensoridseparator = ,
48 sensors,basetimeunit = MICROSECONDS
49 sensors,conversionfactor = 1000000
50 sensors,timebetween = 10000
51
52 # the sensor mapping
53 # sensorid.<id>,entity_id = <entity_id>
54 # sensorid.<id>,entity = <entity>
55 # sensorid.<id>,remark = <remark> (may be <null>)
56 # sensorid.<id>,team_id = <team_id> (may be <null>)
57
58 sensorid.4,entity_id = 100
59 sensorid.4,entity = Ball
60 sensorid.4,remark = <null>
61 sensorid.4,team_id = -1
62
63 sensorid.8,entity_id = 100
64 sensorid.8,entity = Ball
65 sensorid.8,remark = <null>
```

```
66  sensorid.8,team_id = -1
67
68  sensorid.10,entity_id = 100
69  sensorid.10,entity = Ball
70  sensorid.10,remark = <null>
71  sensorid.10,team_id = -1
72
73  sensorid.12,entity_id = 100
74  sensorid.12,entity = Ball
75  sensorid.12,remark = <null>
76  sensorid.12,team_id = -1
77
78  sensorid.13,entity_id = 1
79  sensorid.13,entity = Nick Gertje
80  sensorid.13,remark = Left Leg
81  sensorid.13,team_id = 1
82
83  sensorid.14,entity_id = 1
84  sensorid.14,entity = Nick Gertje
85  sensorid.14,remark = Right Leg
86  sensorid.14,team_id = 1
87
88  sensorid.16,entity_id = 2
89  sensorid.16,entity = Dennis Dotterweich
90  sensorid.16,remark = Right Leg
91  sensorid.16,team_id = 1
92
93  sensorid.19,entity_id = 4
94  sensorid.19,entity = Wili Sommer
95  sensorid.19,remark = Left Leg
96  sensorid.19,team_id = 1
97
98  sensorid.23,entity_id = 6
99  sensorid.23,entity = Roman Hartleb
100 sensorid.23,remark = Left Leg
101 sensorid.23,team_id = 1
102
103 sensorid.24,entity_id = 6
104 sensorid.24,entity = Roman Hartleb
105 sensorid.24,remark = Right Leg
106 sensorid.24,team_id = 1
107
108 sensorid.28,entity_id = 8
109 sensorid.28,entity = Sandro Schneider
110 sensorid.28,remark = Right Leg
111 sensorid.28,team_id = 1
```

```
112
113 sensorid.38,entity_id = 15
114 sensorid.38,entity = Vale Reitstetter
115 sensorid.38,remark = Right Leg
116 sensorid.38,team_id = 2
117
118 sensorid.40,entity_id = 16
119 sensorid.40,entity = Christopher Lee
120 sensorid.40,remark = Right Leg
121 sensorid.40,team_id = 2
122
123 sensorid.44,entity_id = 18
124 sensorid.44,entity = Leo Langhans
125 sensorid.44,remark = Right Leg
126 sensorid.44,team_id = 2
127
128 sensorid.47,entity_id = 2
129 sensorid.47,entity = Dennis Dotterweich
130 sensorid.47,remark = Left Leg
131 sensorid.47,team_id = 1
132
133 sensorid.49,entity_id = 3
134 sensorid.49,entity = Niklas Waelzlein
135 sensorid.49,remark = Left Leg
136 sensorid.49,team_id = 1
137
138 sensorid.52,entity_id = 4
139 sensorid.52,entity = Wili Sommer
140 sensorid.52,remark = Right Leg
141 sensorid.52,team_id = 1
142
143 sensorid.53,entity_id = 5
144 sensorid.53,entity = Philipp Harlass
145 sensorid.53,remark = Left Leg
146 sensorid.53,team_id = 1
147
148 sensorid.54,entity_id = 5
149 sensorid.54,entity = Philipp Harlass
150 sensorid.54,remark = Right Leg
151 sensorid.54,team_id = 1
152
153 sensorid.57,entity_id = 7
154 sensorid.57,entity = Erik Engelhardt
155 sensorid.57,remark = Left Leg
156 sensorid.57,team_id = 1
157
```

```
158 sensorid.58,entity_id = 7
159 sensorid.58,entity = Erik Engelhardt
160 sensorid.58,remark = Right Leg
161 sensorid.58,team_id = 1
162
163 sensorid.59,entity_id = 8
164 sensorid.59,entity = Sandro Schneider
165 sensorid.59,remark = Left Leg
166 sensorid.59,team_id = 1
167
168 sensorid.61,entity_id = 11
169 sensorid.61,entity = Leon Krapf
170 sensorid.61,remark = Left Leg
171 sensorid.61,team_id = 2
172
173 sensorid.62,entity_id = 11
174 sensorid.62,entity = Leon Krapf
175 sensorid.62,remark = Right Leg
176 sensorid.62,team_id = 2
177
178 sensorid.63,entity_id = 12
179 sensorid.63,entity = Kevin Baer
180 sensorid.63,remark = Left Leg
181 sensorid.63,team_id = 2
182
183 sensorid.64,entity_id = 12
184 sensorid.64,entity = Kevin Baer
185 sensorid.64,remark = Right Leg
186 sensorid.64,team_id = 2
187
188 sensorid.65,entity_id = 13
189 sensorid.65,entity = Luca Ziegler
190 sensorid.65,remark = Left Leg
191 sensorid.65,team_id = 2
192
193 sensorid.66,entity_id = 13
194 sensorid.66,entity = Luca Ziegler
195 sensorid.66,remark = Right Leg
196 sensorid.66,team_id = 2
197
198 sensorid.67,entity_id = 14
199 sensorid.67,entity = Ben Mueller
200 sensorid.67,remark = Left Leg
201 sensorid.67,team_id = 2
202
203 sensorid.68,entity_id = 14
```

```
204 sensorid.68,entity = Ben Mueller
205 sensorid.68,remark = Right Leg
206 sensorid.68,team_id = 2
207
208 sensorid.69,entity_id = 15
209 sensorid.69,entity = Vale Reitstetter
210 sensorid.69,remark = Left Leg
211 sensorid.69,team_id = 2
212
213 sensorid.71,entity_id = 16
214 sensorid.71,entity = Christopher Lee
215 sensorid.71,remark = Left Leg
216 sensorid.71,team_id = 2
217
218 sensorid.73,entity_id = 17
219 sensorid.73,entity = Leon Heinze
220 sensorid.73,remark = Left Leg
221 sensorid.73,team_id = 2
222
223 sensorid.74,entity_id = 17
224 sensorid.74,entity = Leon Heinze
225 sensorid.74,remark = Right Leg
226 sensorid.74,team_id = 2
227
228 sensorid.75,entity_id = 18
229 sensorid.75,entity = Leo Langhans
230 sensorid.75,remark = Left Leg
231 sensorid.75,team_id = 2
232
233 sensorid.88,entity_id = 3
234 sensorid.88,entity = Niklas Waelzlein
235 sensorid.88,remark = Right Leg
236 sensorid.88,team_id = 1
237
238 sensorid.97,entity_id = 1
239 sensorid.97,entity = Nick Gertje
240 sensorid.97,remark = Left Arm
241 sensorid.97,team_id = 1
242
243 sensorid.98,entity_id = 1
244 sensorid.98,entity = Nick Gertje
245 sensorid.98,remark = Right Arm
246 sensorid.98,team_id = 1
247
248 sensorid.99,entity_id = 11
249 sensorid.99,entity = Leon Krapf
```

```
sensorid.99,remark = Left Arm
sensorid.99,team_id = 2

sensorid.100,entity_id = 11
sensorid.100,entity = Leon Krapf
sensorid.100,remark = Right Arm
sensorid.100,team_id = 2

sensorid.105,entity_id = 21
sensorid.105,entity = Referee
sensorid.105,remark = Left Leg
sensorid.105,team_id = -2

sensorid.106,entity_id = 21
sensorid.106,entity = Referee
sensorid.106,remark = Right Leg
sensorid.106,team_id = -2
```

Listing A.3: DDC für DEBS Daten

A.3 Anfragen

```

#PARSER SportsQL
#METADATA TimeInterval
#METADATA Latency
#METADATA Datarate
#METADATA SystemLoad

#CONFIG DISTRIBUTE true
#PEER_PARTITION OPERATORCLOUD
#PEER_ALLOCATE ROUNDROBIN

#PEER_POSTPROCESSOR FORCELOCALSOURCES
#PEER_POSTPROCESSOR LOCALSINK
#PEER_POSTPROCESSOR CALCLATENCY

#RUNQUERY
{
    "statisticType": "global",
    "gameType": "soccer",
    "name": "game",
    "parameters": {
        "evaluation": {
            "evaluation": "true",
            "sinkName": "sinknameTest",
```

```
            "fileName": "output.csv",

        }
    }
}
```

Listing A.4: SportsQL für die Spielerpositionen-Anfrage (Evaluation Load-Balancing)

```

#PARSER SportsQL
#METADATA TimeInterval
#METADATA Latency
#METADATA Datarate
#METADATA SystemLoad

#CONFIG DISTRIBUTE true
#PEER_PARTITION OPERATORSETCLOUD
#PEER_ALLOCATE ROUNDROBIN

#PEER_POSTPROCESSOR FORCELOCALSOURCES
#PEER_POSTPROCESSOR LOCALSINK
#PEER_POSTPROCESSOR CALCLATENCY
#RUNQUERY
{
    "statisticType": "team",
    "gameType": "soccer",
    "name": "ball_contact",
    "parameters": {
        "evaluation": {
            "evaluation": "true",
            "sinkName": "sinknameTest",
            "fileName": "output.csv",

        }
    }
}
```

Listing A.5: SportsQL für die Ballkontakte-Anfrage (Evaluation Load-Balancing)

```
#PARSER SportsQL

#METADATA TimeInterval
#METADATA Latency
#METADATA Datarate
#METADATA SystemLoad

#CONFIG DISTRIBUTE true
```

```
9
10 #PEER_PARTITION OPERATORSETCLOUD
11 #PEER_ALLOCATE ROUNDROBIN
12 #PEER_POSTPROCESSOR MERGE
13 #PEER_POSTPROCESSOR FORCELOCALSOURCES
14 #PEER_POSTPROCESSOR CALCLATENCY
15
16 #ADDQUERY
17 {
18     "statisticType": "global",
19     "gameType": "soccer",
20     "name": "gameTime",
21     "parameters": {
22         "evaluation": {
23             "evaluation": "true",
24             "sinkName": "sinknameTest",
25             "calcLatenzOp": "false",
26             "fileName": "RECOVERY_GAMETIME_4PEERS_SINK_01.csv",
27             "fileAppend": "true"
28         }
29     }
30 }
```

Listing A.6: SportsQL für die GameTime(SINK)-Evaluation des Recovery

```
1  #PARSER SportsQL
2
3  #METADATA TimeInterval
4  #METADATA Latency
5  #METADATA Datarate
6  #METADATA SystemLoad
7
8  #CONFIG DISTRIBUTE true
9
10 #PEER_PARTITION OPERATORCLOUD
11 #PEER_ALLOCATE ROUNDROBIN
12 #PEER_POSTPROCESSOR MERGE
13 #PEER_POSTPROCESSOR FORCELOCALSOURCES
14 #PEER_POSTPROCESSOR CALCLATENCY
15
16 #ADDQUERY
17 {
18     "statisticType": "global",
19     "gameType": "soccer",
20     "name": "gameTime",
21     "parameters": {
22         "evaluation": {
```

```
            "evaluation": "true",
            "sinkName": "sinknameTest",
            "calcLatenzOp": "false",
            "fileName": "RECOVERY_GAMETIME_4PEERS_MIDDLE_01.csv",
            "fileAppend": "true"
        }
    }
}
```

Listing A.7: SportsQL für die GameTime(MITTE)-Evaluation des Recovery

```
#PARSER SportsQL

#METADATA TimeInterval
#METADATA Latency
#METADATA Datarate
#METADATA SystemLoad

#CONFIG DISTRIBUTE true

#PEER_PARTITION OPERATORSETCLOUD
#PEER_ALLOCATE ROUNDROBIN
#PEER_MODIFICATION REPLICATION 2
#PEER_MODIFICATION RECOVERY_ACTIVESTANDBY
#PEER_POSTPROCESSOR MERGE
#PEER_POSTPROCESSOR FORCELOCALSOURCES
#PEER_POSTPROCESSOR CALCLATENCY

#ADDQUERY
{
    "statisticType": "global",
    "gameType": "soccer",
    "name": "gameTime",
    "parameters": {
        "evaluation": {
            "evaluation": "true",
            "sinkName": "sinknameTest",
            "calcLatenzOp": "false",
            "fileName": "RECOVERY_GAMETIME_AS_4PEERS_MIDDLE_01.csv"
                ↪ ,
            "fileAppend": "true"
        }
    }
}
```

Listing A.8: SportsQL für die GameTimeAS-Evaluationen des Recovery

```
#PARSER PQL
#CONFIG DISTRIBUTE true

#METADATA TimeInterval
#METADATA Latency
#METADATA Datarate
#METADATA SystemLoad

#PEER_PARTITION USER
#PEER_ALLOCATE ROUNDROBIN
#PEER_POSTPROCESSOR FORCELOCALSOURCES
#PEER_POSTPROCESSOR LOCALSINK
#PEER_POSTPROCESSOR CALCLATENCY
#PEER_POSTPROCESSOR CALCDATARATE 100
#PEER_POSTPROCESSOR MERGE

#ADDQUERY
StreamAO_679 = STREAM({DESTINATION = 'local',
                    name='soccergame',
                    source='soccergame'
                  }
                )
MapAO_678 = MAP({DESTINATION = 'Peer8',
                 threads=0,
                 name='Map',
                 suppresserrors='false',
                 allownull='false',
                 expressions=[
'entity_id',                    ['doubleToInteger((ts) / (60000000))'
   ↪ ,'minute'],
                   ['doubleToInteger(((ts) / (1000000)) % (60))','
                      ↪ second'],'x','y','z','v','a','ts','team_id'
                 ],
                 evaluateonpunctuation='false',
                 debug='false'
               },
               0:StreamAO_679
             )
SelectAO_677 = SELECT({DESTINATION = 'Peer8',
                    name='Select',
                    predicate='((second) >= (0)) AND (((minute) <=
                       ↪ (90)) AND ((minute) >= (0)))',
                    heartbeatrate=0,
                    debug='false'
                  },
```

```
43                  0:MapAO_678
44                )
45 RouteAO_674 = ROUTE({DESTINATION = 'Peer8',
46                  overlappingpredicates='false',
47                  name='Route',
48                  sendingheartbeats='false',
49                  predicates=[
50 '((((entity_id) = (100)) || ((entity_id) = (100))) || ((entity_id)
     ↪ = (100))) || ((entity_id) = (100))','((team_id) = (1)) || ((
     ↪ team_id) = (2))'
51                  ],
52                  debug='false'
53                },
54                0:SelectAO_677
55              )
56 MapAO_675 = MAP({DESTINATION = 'Peer8',
57                threads=0,
58                name='Map',
59                suppresserrors='false',
60                allownull='false',
61                expressions=[
62                  ['entity_id','player_entity_id'],
63                  ['team_id','player_team_id'],
64                  ['x','player_x'],
65                  ['y','player_y'],
66                  ['z','player_z']
67                ],
68                evaluateonpunctuation='false',
69                debug='false'
70              },
71              1:RouteAO_674
72            )
73 MapAO_673 = MAP({DESTINATION = 'Peer8',
74                threads=0,
75                name='Map',
76                suppresserrors='false',
77                allownull='false',
78                expressions=[
79                  ['ts','ball_ts'],
80                  ['x','ball_x'],
81                  ['y','ball_y'],
82                  ['z','ball_z']
83                ],
84                evaluateonpunctuation='false',
85                debug='false'
86              },
```

```
87                0:RouteAO_674
88              )
89  ElementWindowAO_676 = ELEMENTWINDOW({DESTINATION = 'Peer8',
90                              name='ElementWindow',
91                              advance=1,
92                              size=1,
93                              debug='false'
94                            },
95                            0:MapAO_675
96                          )
97  ElementWindowAO_672 = ELEMENTWINDOW({DESTINATION = 'Peer8',
98                              name='ElementWindow',
99                              advance=1,
100                             size=1,
101                             debug='false'
102                           },
103                           0:MapAO_673
104                         )
105 JoinAO_671 = JOIN({DESTINATION = 'Peer8',
106                   name='Join',
107                   assureorder='true',
108                   debug='false'
109                 },
110                 0:ElementWindowAO_672,
111                 0:ElementWindowAO_676
112               )
113 StateMapAO_670 = STATEMAP({DESTINATION = 'Peer8',
114                       threads=0,
115                       name='StateMap',
116                       suppresserrors='false',
117                       allownull='false',
118                       expressions=[
119 'ball_ts','ball_x','ball_y','ball_z','player_entity_id','
    ↪ player_team_id',                          ['sqrt(((abs((ball_x)
    ↪ ␣-␣(player_x)))␣^␣(2))␣+␣((abs((ball_y)␣-␣(player_y)))␣^␣(2))
    ↪ )','distance']
120                       ],
121                       evaluateonpunctuation='false',
122                       allownullinoutput='false',
123                       debug='false'
124                     },
125                     0:JoinAO_671
126                   )
127 SelectAO_668 = SELECT({DESTINATION = 'Peer8',
128                     name='Select',
129                     predicate='(distance)␣<␣(800)',
```

```
130                     heartbeatrate=0,
131                     debug='false'
132                   },
133                   0:StateMapAO_670
134                 )
135 CoalesceAO_667 = COALESCE({DESTINATION = 'Peer8',
136                       aggregations=[
137                         ['MIN','soccergamesoccergame.distance','
                              ↪ min_distance','Double']
138                       ],
139                       name='Coalesce',
140                       maxelementspergroup=-1,
141                       fastgrouping='false',
142                       drainatdone='true',
143                       outputpa='false',
144                       drainatclose='false',
145                       createonheartbeat='false',
146                       attr=['soccergamesoccergame.ball_ts'],
147                       heartbeatrate=-1,
148                       debug='false',
149                       dumpatvaluecount=-1
150                     },
151                     0:SelectAO_668
152                   )
153 RenameAO_669 = RENAME({DESTINATION = 'Peer8',
154                       name='Rename',
155                       aliases=['ball_ts','ball_ts_2'],
156                       isnoop='false',
157                       pairs='true',
158                       debug='false'
159                     },
160                     0:SelectAO_668
161                   )
162 JoinAO_666 = JOIN({DESTINATION = 'Peer10',
163                     name='Join',
164                     assureorder='true',
165                     predicate='(distance) = (min_distance)',
166                     card='ONE_ONE',
167                     debug='false'
168                   },
169                   0:CoalesceAO_667,
170                   0:RenameAO_669
171                 )
172 ProjectAO_665 = PROJECT({DESTINATION = 'Peer10',
173                       name='Project',
174                       attributes=[
```

```
175 'soccergamesoccergame.ball_ts','soccergamesoccergame.ball_x','
    ↪ soccergamesoccergame.ball_y','soccergamesoccergame.ball_z','
    ↪ soccergamesoccergame.player_entity_id','soccergamesoccergame.
    ↪ player_team_id'
176                     ],
177                     debug='false'
178                   },
179                   0:JoinAO_666
180                 )
181 StateMapAO_664 = STATEMAP({DESTINATION = 'Peer10',
182                       threads=0,
183                       name='StateMap',
184                       suppresserrors='false',
185                       allownull='false',
186                       expressions=[
187                         ['__last_1.ball_ts','ball_ts_1'],
188                         ['__last_1.ball_x','ball_x_1'],
189                         ['__last_1.ball_y','ball_y_1'],
190                         ['__last_1.ball_z','ball_z_1'],
191                         ['__last_1.player_entity_id','
                              ↪ player_entity_id_1'],
192                         ['__last_1.player_team_id','
                              ↪ player_team_id_1'],
193                         ['ball_ts','ball_ts_2'],
194                         ['ball_x','ball_x_2'],
195                         ['ball_y','ball_y_2'],
196                         ['ball_z','ball_z_2'],
197                         ['player_entity_id','player_entity_id_2'],
198                         ['player_team_id','player_team_id_2']
199                       ],
200                       evaluateonpunctuation='false',
201                       allownullinoutput='false',
202                       debug='false'
203                     },
204                     0:ProjectAO_665
205                   )
206 SelectAO_663 = SELECT({DESTINATION = 'Peer10',
207                       name='Select',
208                       predicate='(player_entity_id_1) != (
                              ↪ player_entity_id_2)',
209                       heartbeatrate=0,
210                       debug='false'
211                     },
212                     0:StateMapAO_664
213                   )
214 StateMapAO_662 = STATEMAP({DESTINATION = 'Peer10',
```

```
                        threads=0,
                        name='StateMap',
                        suppresserrors='false',
                        allownull='false',
                        expressions=[
                          ['ball_ts_1','ball_ts_1'],
                          ['ball_x_1','ball_x_1'],
                          ['ball_y_1','ball_y_1'],
                          ['ball_z_1','ball_z_1'],
                          ['player_entity_id_1','player_entity_id_1'
                             ↪ ],
                          ['player_team_id_1','player_team_id_1'],
                          ['ball_ts_2','ball_ts_2'],
                          ['ball_x_2','ball_x_2'],
                          ['ball_y_2','ball_y_2'],
                          ['ball_z_2','ball_z_2'],
                          ['player_entity_id_2','player_entity_id_2'
                             ↪ ],
                          ['player_team_id_2','player_team_id_2'],
                          ['player_team_id_2','player_team_id_2'],
                          ['sqrt(((abs((ball_x_2) - (ball_x_1))) ^
                             ↪ (2)) + ((abs((ball_y_2) - (ball_y_1))
                             ↪ ) ^ (2)))','pass_distance'],
                          ['((atan2((ball_x_2) - (ball_x_1), (
                             ↪ ball_y_2) - (ball_y_1))) * (180)) /
                             ↪ (3.142)','pass_angle'],
                          ['(ball_ts_1) - (__last_1.ball_ts_2)','
                             ↪ time_between']
                        ],
                        evaluateonpunctuation='false',
                        allownullinoutput='false',
                        debug='false'
                      },
                      0:SelectAO_663
                    )
StateMapAO_661 = STATEMAP({DESTINATION = 'Peer10',
                        threads=0,
                        name='StateMap',
                        suppresserrors='false',
                        allownull='false',
                        expressions=[
                          ['ball_ts_1','ball_ts_1'],
                          ['ball_x_1','ball_x_1'],
                          ['ball_y_1','ball_y_1'],
                          ['ball_z_1','ball_z_1'],
```

```
253                         ['player_entity_id_1','player_entity_id_1'
                               ↪ ],
254                         ['player_team_id_1','player_team_id_1'],
255                         ['ball_ts_2','ball_ts_2'],
256                         ['ball_x_2','ball_x_2'],
257                         ['ball_y_2','ball_y_2'],
258                         ['ball_z_2','ball_z_2'],
259                         ['player_entity_id_2','player_entity_id_2'
                               ↪ ],
260                         ['player_team_id_2','player_team_id_2'],
261                         ['player_team_id_2','player_team_id_2'],
262                         ['pass_distance','pass_distance'],
263                         ['eif(((pass_angle) >= (-75)) && ((
                               ↪ pass_angle) <= (75)), eif((
                               ↪ player_team_id_1) = (1), "back", "
                               ↪ forwards"), eif((((pass_angle) >=
                               ↪ (75)) && ((pass_angle) <= (105))) ||
                               ↪ (((pass_angle) <= (-75)) && ((
                               ↪ pass_angle) >= (-105))), "cross", eif
                               ↪ ((player_team_id_1) = (1), "forwards
                               ↪ ", "back")))','pass_direction'],
264                         ['eif((pass_distance) > (10000), "long", "
                               ↪ short")','pass_length'],
265                         ['eif((player_entity_id_2) = (__last_1.
                               ↪ player_entity_id_1), true, false)','
                               ↪ double_pass'],
266                         ['eif((time_between) < (500000), true, "
                               ↪ false)','direct_pass']
267                       ],
268                       evaluateonpunctuation='false',
269                       allownullinoutput='false',
270                       debug='false'
271                     },
272                     0:StateMapAO_662
273                   )
274 SelectAO_659 = SELECT({DESTINATION = 'Peer10',
275                     name='Select',
276                     predicate='((pass_distance) > (1000)) AND (((((
                           ↪ ball_y_2) <= (52489)) AND ((ball_y_1) <=
                           ↪ (52489))) AND ((ball_x_2) <= (33965))) "
                           ↪ AND (((((ball_x_1) <= (33965)) AND ((
                           ↪ ball_y_2) >= (-50))) AND ((ball_y_1) >=
                           ↪ (-50))) AND (((ball_x_2) >= (-33960)) AND
                           ↪ ((ball_x_1) >= (-33960)))))',
277                     heartbeatrate=0,
278                     debug='false'
```

```
279                     },
280                     0:StateMapAO_661
281                 )
282 StateMapAO_658 = STATEMAP({DESTINATION = 'Peer10',
283                         threads=0,
284                         name='StateMap',
285                         suppresserrors='false',
286                         allownull='false',
287                         expressions=[
288                           ['player_entity_id_1','player_entity_id'],
289                           ['eif((player_team_id_1) = (
                                ↪ player_team_id_2), 1, 0)','
                                ↪ passes_successful'],
290                           ['eif((player_team_id_1) = (
                                ↪ player_team_id_2), 0, 1)','
                                ↪ passes_misplaced'],
291                           ['0','passes_received'],
292                           ['0','passes_intercepted'],
293                           ['eif(((player_team_id_1) = (
                                ↪ player_team_id_2)) && ((pass_length)
                                ↪ = ("short")), 1, 0)','
                                ↪ short_passes_successful'],
294                           ['eif(((player_team_id_1) != (
                                ↪ player_team_id_2)) && ((pass_length)
                                ↪ = ("short")), 1, 0)','
                                ↪ short_passes_misplaced'],
295                           ['eif(((player_team_id_1) = (
                                ↪ player_team_id_2)) && ((pass_length)
                                ↪ = ("long")), 1, 0)','
                                ↪ long_passes_successful'],
296                           ['eif(((player_team_id_1) != (
                                ↪ player_team_id_2)) && ((pass_length)
                                ↪ = ("long")), 1, 0)','
                                ↪ long_passes_misplaced'],
297                           ['eif(((player_team_id_1) = (
                                ↪ player_team_id_2)) && ((
                                ↪ pass_direction) = ("forwards")), 1,
                                ↪ 0)','forward_passes_successful'],
298                           ['eif(((player_team_id_1) != (
                                ↪ player_team_id_2)) && ((
                                ↪ pass_direction) = ("forwards")), 1,
                                ↪ 0)','forward_passes_misplaced'],
299                           ['eif(((player_team_id_1) = (
                                ↪ player_team_id_2)) && ((
                                ↪ pass_direction) = ("cross")), 1, 0)',
                                ↪ 'cross_passes_successful'],
```

```
300                          ['eif(((player_team_id_1) != (
                                ↪ player_team_id_2)) && ((
                                ↪ pass_direction) = ("cross")), 1, 0)',
                                ↪ 'cross_passes_misplaced'],
301                          ['eif(((player_team_id_1) = (
                                ↪ player_team_id_2)) && ((
                                ↪ pass_direction) = ("back")), 1, 0)','
                                ↪ back_passes_successful'],
302                          ['eif(((player_team_id_1) != (
                                ↪ player_team_id_2)) && ((
                                ↪ pass_direction) = ("back")), 1, 0)','
                                ↪ back_passes_misplaced'],
303                          ['eif(((player_team_id_1) = (
                                ↪ player_team_id_2)) && ((direct_pass)
                                ↪ = (true)), 1, 0)','
                                ↪ direct_passes_successful'],
304                          ['eif(((player_team_id_1) != (
                                ↪ player_team_id_2)) && ((direct_pass)
                                ↪ = (false)), 1, 0)','
                                ↪ direct_passes_misplaced'],
305                          ['eif(((player_team_id_1) = (
                                ↪ player_team_id_2)) && ((double_pass)
                                ↪ = (true)), 1, 0)','
                                ↪ double_passes_successful'],
306                          ['eif(((player_team_id_1) != (
                                ↪ player_team_id_2)) && ((double_pass)
                                ↪ = (false)), 1, 0)','
                                ↪ double_passes_misplaced']
307                        ],
308                        evaluateonpunctuation='false',
309                        allownullinoutput='false',
310                        debug='false'
311                      },
312                      0:SelectAO_659
313                    )
314 StateMapAO_660 = STATEMAP({DESTINATION = 'Peer10',
315                        threads=0,
316                        name='StateMap',
317                        suppresserrors='false',
318                        allownull='false',
319                        expressions=[
320                          ['player_entity_id_2','player_entity_id'],
321                          ['0','passes_successful'],
322                          ['0','passes_misplaced'],
```

```
                        ['eif((player_team_id_1) = (
                           ↪ player_team_id_2), 1, 0)','
                           ↪ passes_received'],
                        ['eif((player_team_id_1) = (
                           ↪ player_team_id_2), 0, 1)','
                           ↪ passes_intercepted'],
                        ['0','short_passes_successful'],
                        ['0','short_passes_misplaced'],
                        ['0','long_passes_successful'],
                        ['0','long_passes_misplaced'],
                        ['0','forward_passes_successful'],
                        ['0','forward_passes_misplaced'],
                        ['0','cross_passes_successful'],
                        ['0','cross_passes_misplaced'],
                        ['0','back_passes_successful'],
                        ['0','back_passes_misplaced'],
                        ['0','direct_passes_successful'],
                        ['0','direct_passes_misplaced'],
                        ['0','double_passes_successful'],
                        ['0','double_passes_misplaced']
                      ],
                      evaluateonpunctuation='false',
                      allownullinoutput='false',
                      debug='false'
                    },
                    0:SelectAO_659
                  )
MergeAO_657 = MERGE({DESTINATION = 'Peer10',
                    name='Merge',
                    debug='false'
                  },
                  0:StateMapAO_658,
                  0:StateMapAO_660
                )
TimestampAO_656 = TIMESTAMP({DESTINATION = 'Peer10',
                      name='Timestamp',
                      systemtime='true',
                      offset=0,
                      factor=0,
                      clearend='true',
                      debug='false'
                    },
                    0:MergeAO_657
                  )
AssureHeartbeatAO_655 = ASSUREHEARTBEAT({DESTINATION = 'Peer10',
                            sendalwaysheartbeat='false',
```

```
                                allowoutoforder='false',
                                name='AssureHeartbeat',
                                startatcurrenttime='false',
                                realtimedelay=10000,
                                applicationtimedelay=10000,
                                starttimerafterfirstelement='false',
                                debug='false'
                              },
                              0:TimestampAO_656
                            )
AggregateAO_653 = AGGREGATE({DESTINATION = 'Peer7',
                         aggregations=[
          ['SUM','soccergamesoccergamesoccergamesoccergame.
             ↪ passes_successful','passes_successful','Double'],
          ['SUM','soccergamesoccergamesoccergamesoccergame.
             ↪ passes_misplaced','passes_misplaced','Double'],
          ['SUM','soccergamesoccergamesoccergamesoccergame.
             ↪ passes_received','passes_received','Double'],
          ['SUM','soccergamesoccergamesoccergamesoccergame.
             ↪ passes_intercepted','passes_intercepted','Double'],
          ['SUM','soccergamesoccergamesoccergamesoccergame.
             ↪ short_passes_successful','short_passes_successful','
             ↪ Double'],
          ['SUM','soccergamesoccergamesoccergamesoccergame.
             ↪ short_passes_misplaced','short_passes_misplaced','
             ↪ Double'],
          ['SUM','soccergamesoccergamesoccergamesoccergame.
             ↪ long_passes_successful','long_passes_successful','
             ↪ Double'],
          ['SUM','soccergamesoccergamesoccergamesoccergame.
             ↪ long_passes_misplaced','long_passes_misplaced','
             ↪ Double'],
          ['SUM','soccergamesoccergamesoccergamesoccergame.
             ↪ forward_passes_successful','
             ↪ forward_passes_successful','Double'],
          ['SUM','soccergamesoccergamesoccergamesoccergame.
             ↪ forward_passes_misplaced','forward_passes_misplaced'
             ↪ ,'Double'],
          ['SUM','soccergamesoccergamesoccergamesoccergame.
             ↪ cross_passes_successful','cross_passes_successful','
             ↪ Double'],
          ['SUM','soccergamesoccergamesoccergamesoccergame.
             ↪ cross_passes_misplaced','cross_passes_misplaced','
             ↪ Double'],
```

```
389             ['SUM','soccergamesoccergamesoccergamesoccergame.
                   ↪ back_passes_successful','back_passes_successful','
                   ↪ Double'],
390             ['SUM','soccergamesoccergamesoccergamesoccergame.
                   ↪ back_passes_misplaced','back_passes_misplaced','
                   ↪ Double'],
391             ['SUM','soccergamesoccergamesoccergamesoccergame.
                   ↪ direct_passes_successful','direct_passes_successful'
                   ↪ ,'Double'],
392             ['SUM','soccergamesoccergamesoccergamesoccergame.
                   ↪ direct_passes_misplaced','direct_passes_misplaced','
                   ↪ Double'],
393             ['SUM','soccergamesoccergamesoccergamesoccergame.
                   ↪ double_passes_successful','double_passes_successful'
                   ↪ ,'Double'],
394             ['SUM','soccergamesoccergamesoccergamesoccergame.
                   ↪ double_passes_misplaced','double_passes_misplaced','
                   ↪ Double']
395                         ],
396                         name='Aggregate',
397                         group_by=[
398 'soccergamesoccergamesoccergamesoccergame.player_entity_id'
399                         ],
400                         fastgrouping='false',
401                         drainatdone='true',
402                         outputpa='false',
403                         drainatclose='false',
404                         debug='false',
405                         dumpatvaluecount=1
406                       },
407                       0:AssureHeartbeatAO_655
408                     )
409 CSVFileSink_654 = CSVFILESINK({DESTINATION = 'local',
410                         name='passesCSVSink',
411                         filename='RECOVERY_PASSES_4PEERS_test.csv',
412                         sink='system.passesCSVSink',
413                         options=[['csv.writemetadata','true'],['
                               ↪ append','true']],
414                         debug='false'
415                       },
416                       0:AggregateAO_653
417                     )
```

Listing A.9: PQL für die Passes-Evaluation des Recovery

```
1 #METADATA TimeInterval
2 #METADATA Latency
```

```
#METADATA Datarate
#METADATA SystemLoad

#CONFIG DISTRIBUTE true

#PEER_PARTITION QUERYCLOUD
#PEER_ALLOCATE ROUNDROBIN
#PEER_MODIFICATION REPLICATION 2
#PEER_MODIFICATION RECOVERY_ACTIVESTANDBY
#PEER_POSTPROCESSOR FORCELOCALSOURCES
#PEER_POSTPROCESSOR LOCALSINK
#PEER_POSTPROCESSOR CALCLATENCY
#PEER_POSTPROCESSOR CALCDATARATE 100
#PEER_POSTPROCESSOR MERGE

#ADDQUERY
{
    "statisticType": "player",
    "gameType": "soccer",
    "name": "passes",
    "parameters": {
        "evaluation": {
            "evaluation": "true",
            "sinkName": "passesCSVSink",
            "calcLatenzOp": "false",
            "fileName": "RECOVERY_PASSES_AS_4PEERS_01.csv",
            "fileAppend": "true"
        }
    }
}
```

Listing A.10: SportsQL für die PassesAS-Evaluation des Recovery

A.4 Anfragepläne

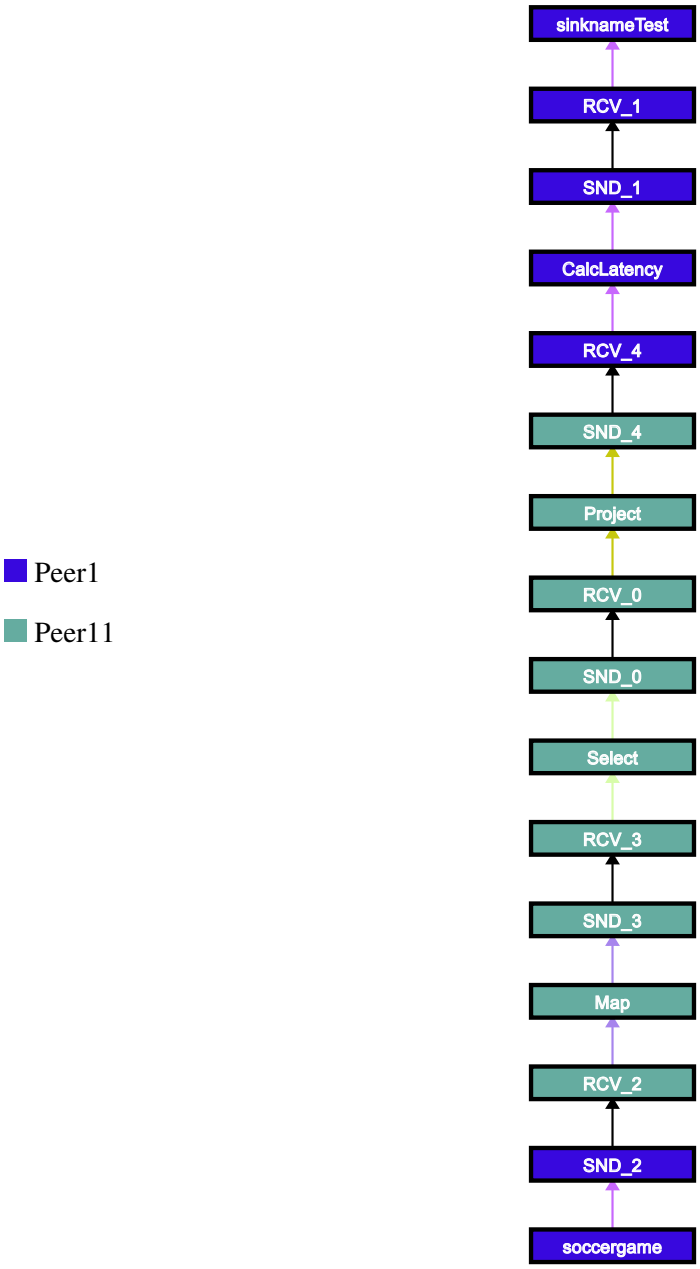


Abbildung A.1: Spielerpositionen-Anfrage verteilter Anfrageplan (Evaluation Load-Balancing)

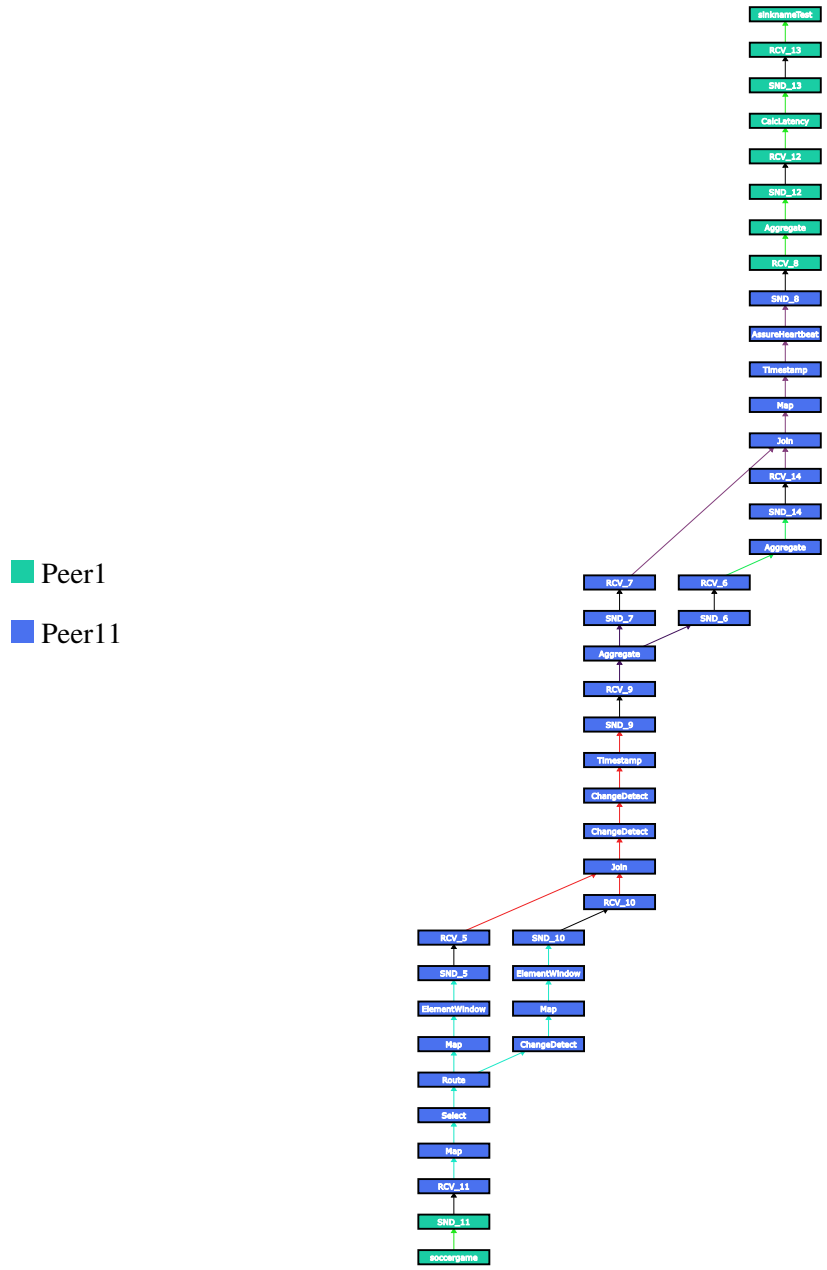


Abbildung A.2: Ballkontakte-Anfrage verteilter Anfrageplan (Evaluation Load-Balancing)

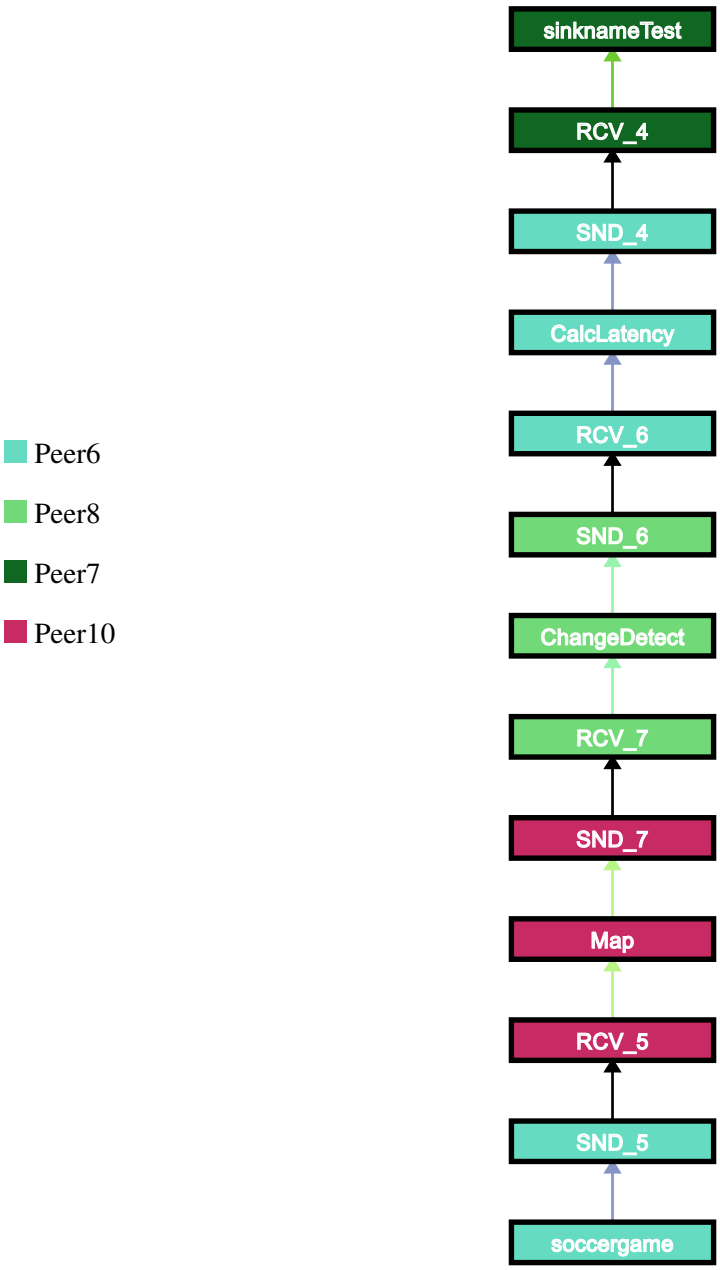


Abbildung A.3: Analysezeit(Mitte)-Anfrageplan vor dem Ausfall

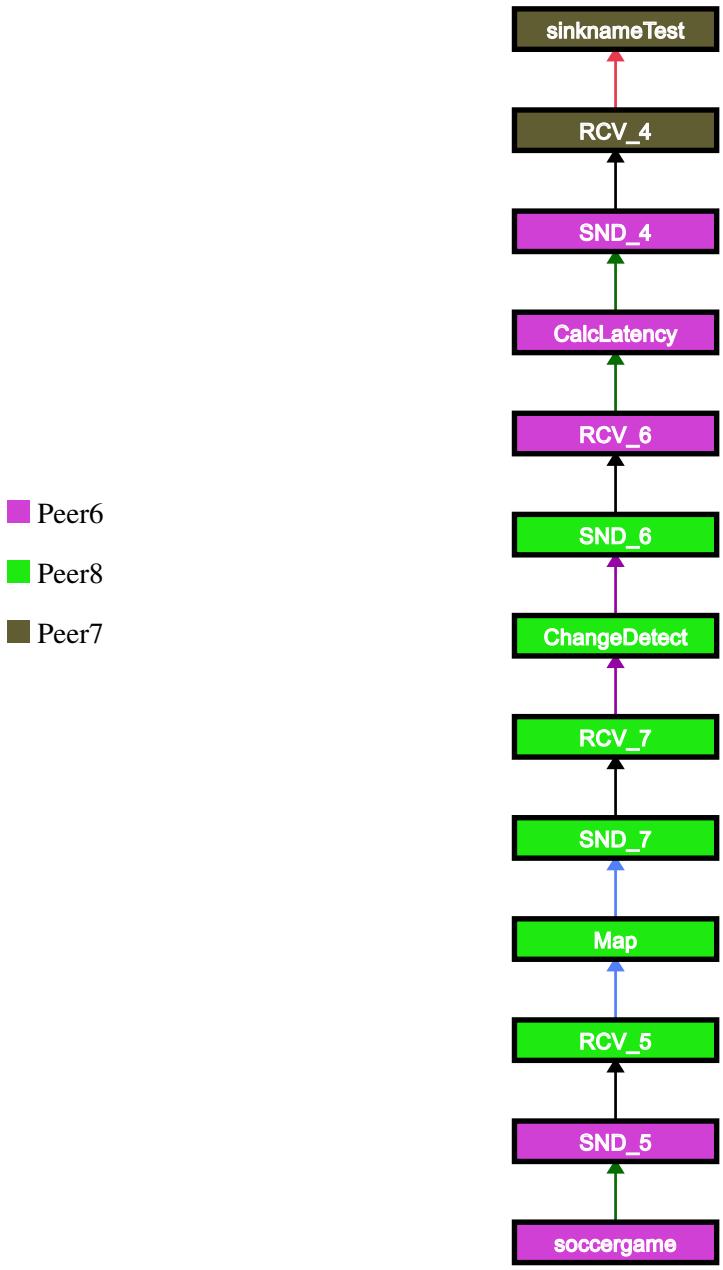


Abbildung A.4: Analysezeit(Mitte)-Anfrageplan nach dem Recovery

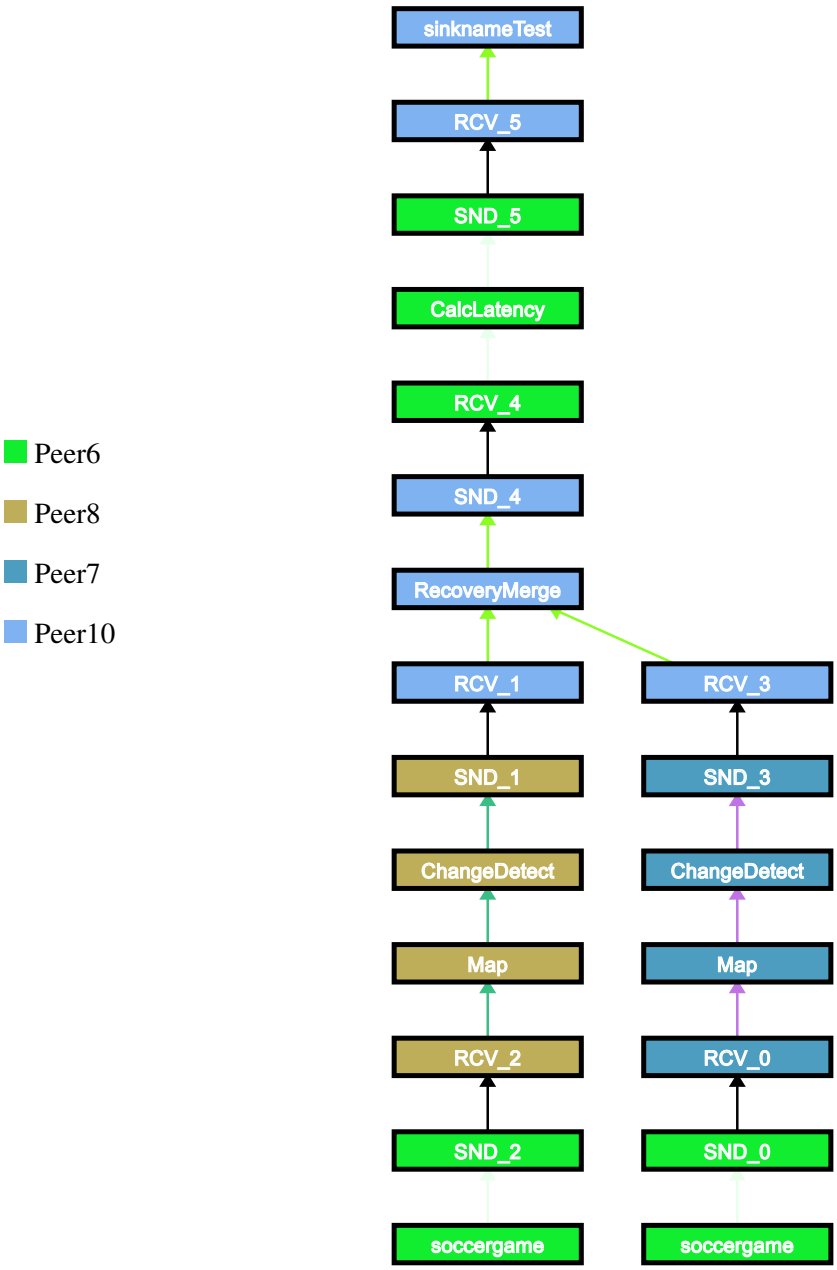


Abbildung A.5: AnalysezeitAS(RecoveryMerge)-Anfrageplan vor dem Ausfall

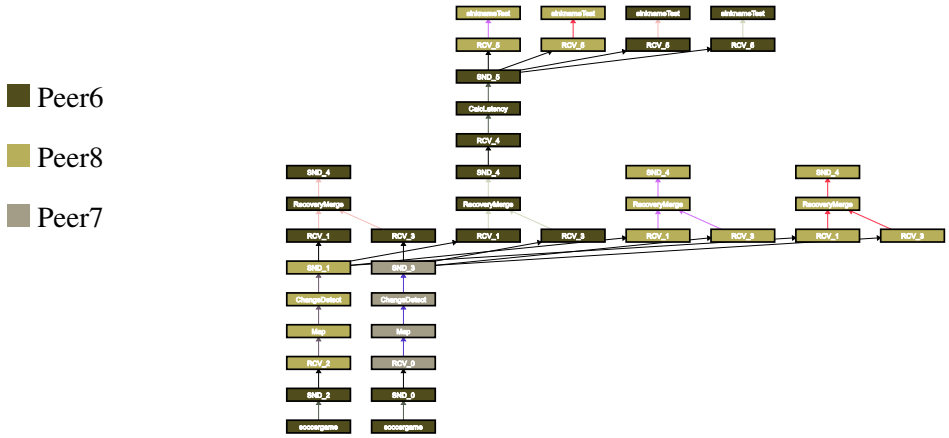


Abbildung A.6: AnalysezeitAS(RecoveryMerge)-Anfrageplan nach dem Recovery

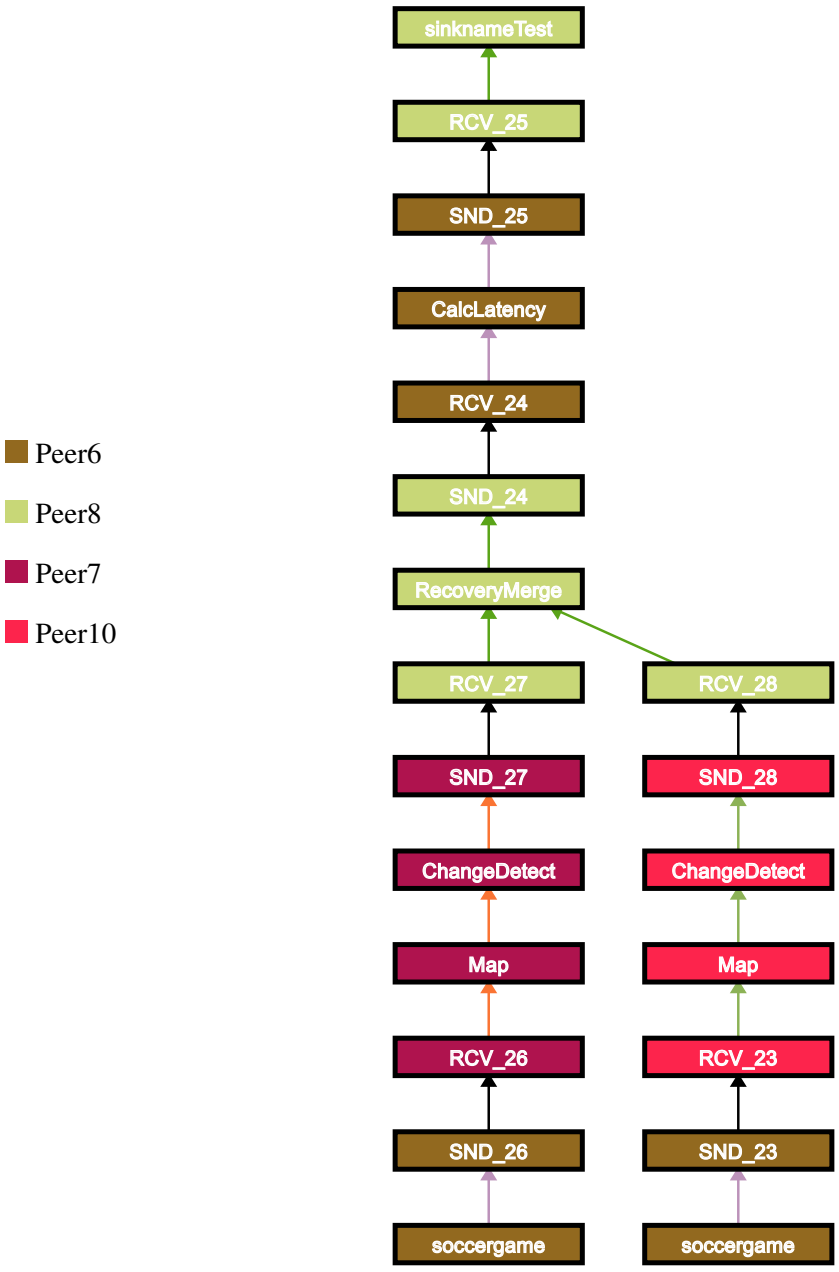


Abbildung A.7: AnalysezeitAS(Mitte)-Anfrageplan vor dem Ausfall

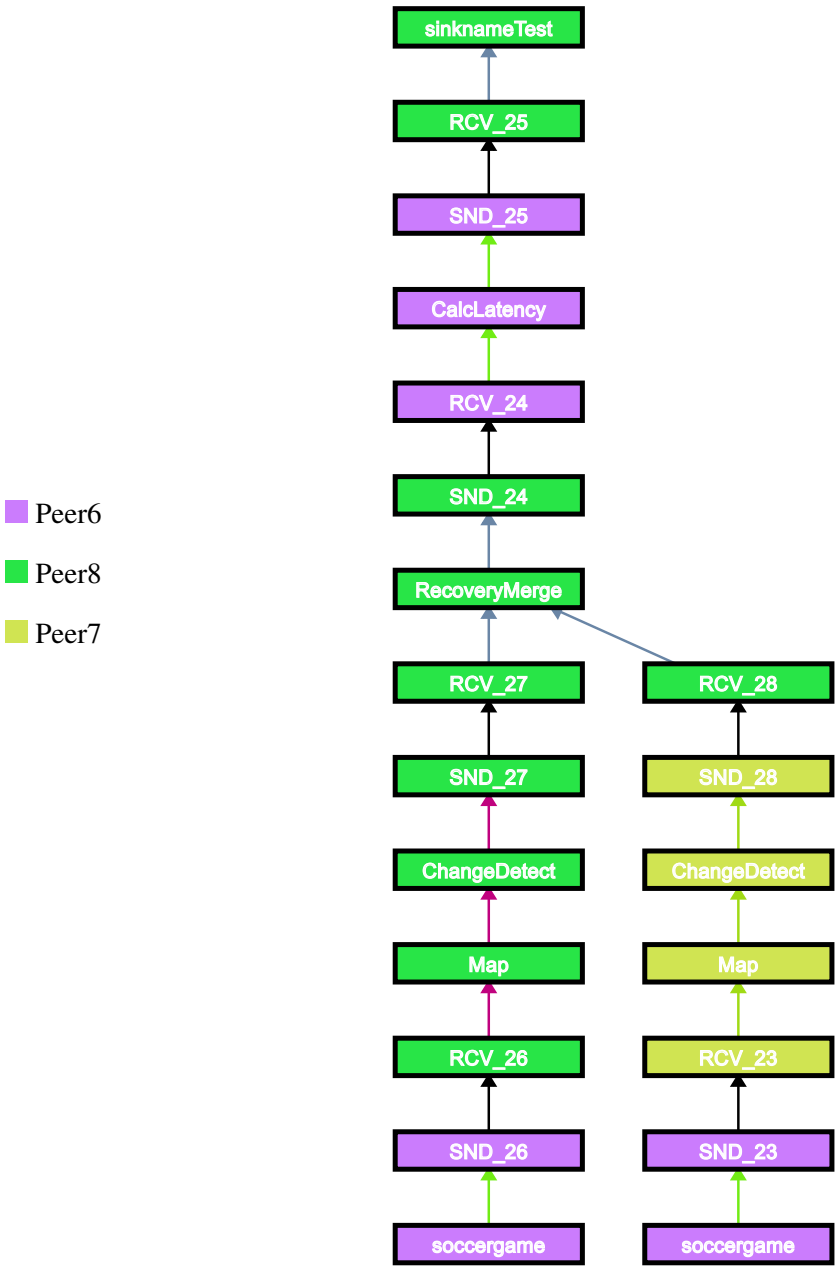


Abbildung A.8: AnalysezeitAS(Mitte)-Anfrageplan nach dem Recovery

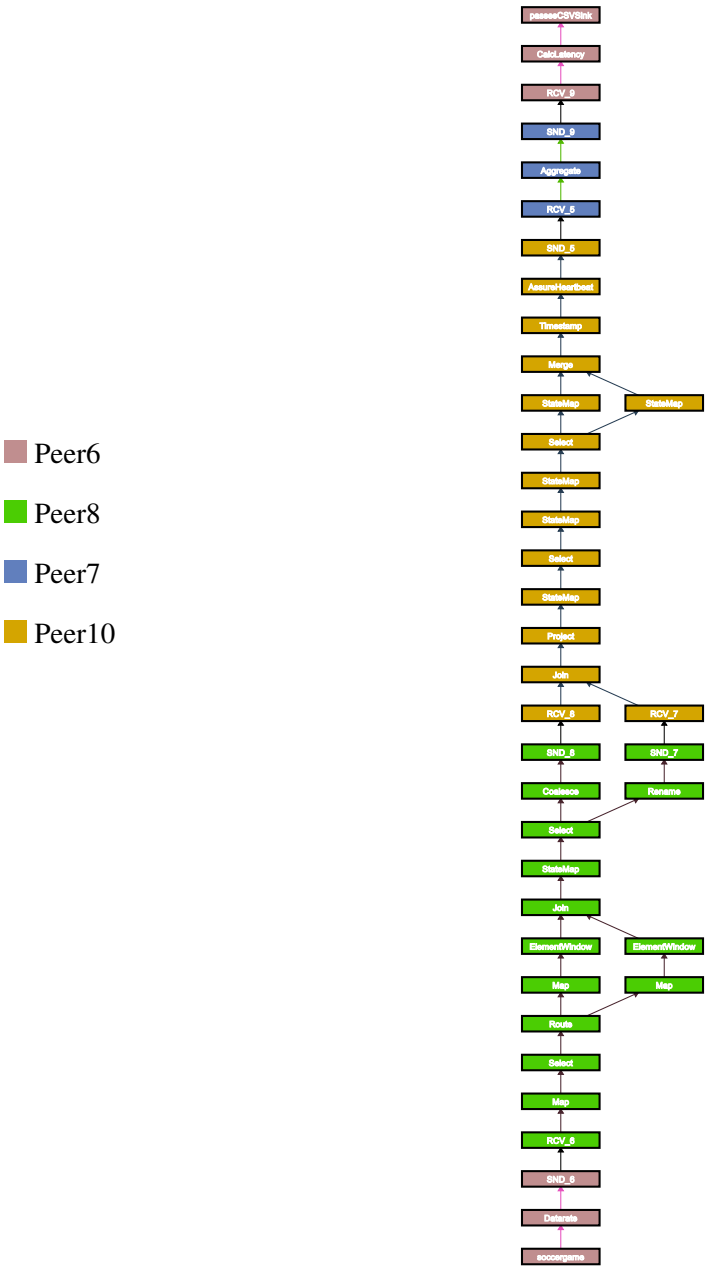


Abbildung A.9: Pässe-Anfrageplan vor dem Ausfall

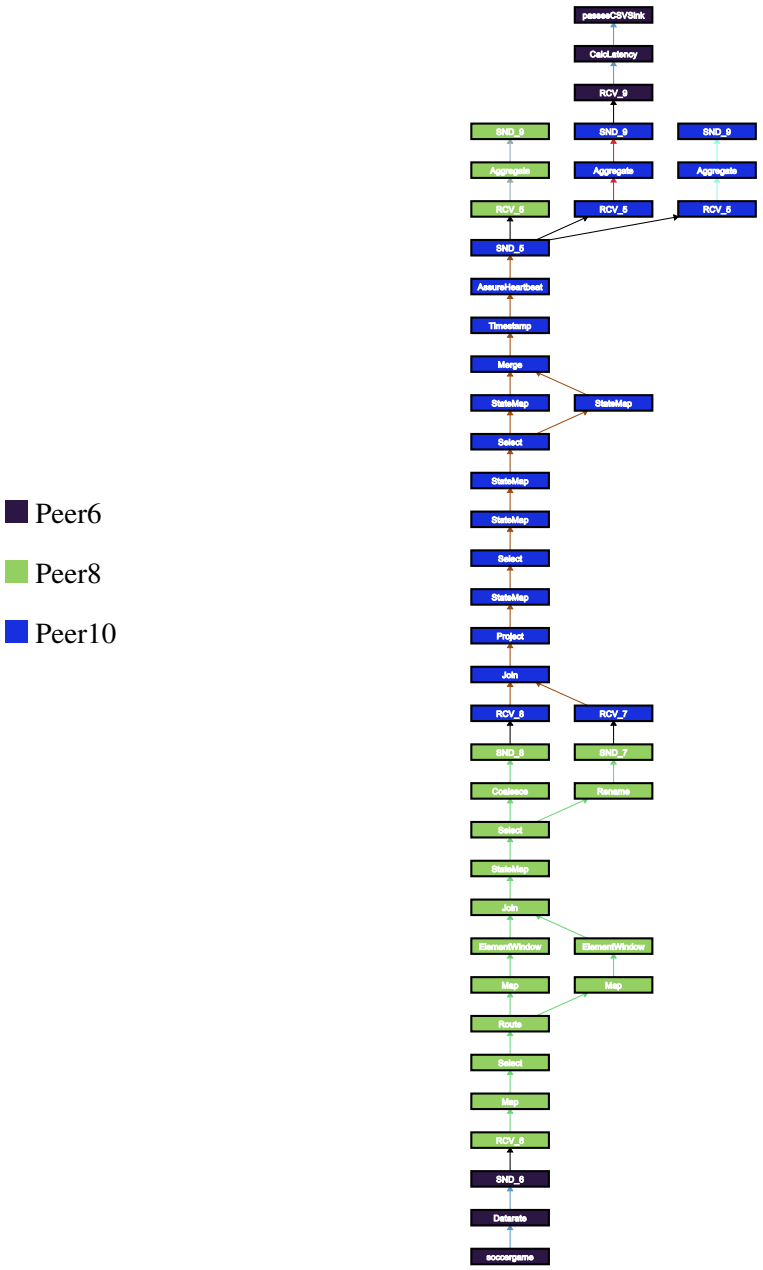


Abbildung A.10: Pässe-Anfrageplan nach dem Recovery

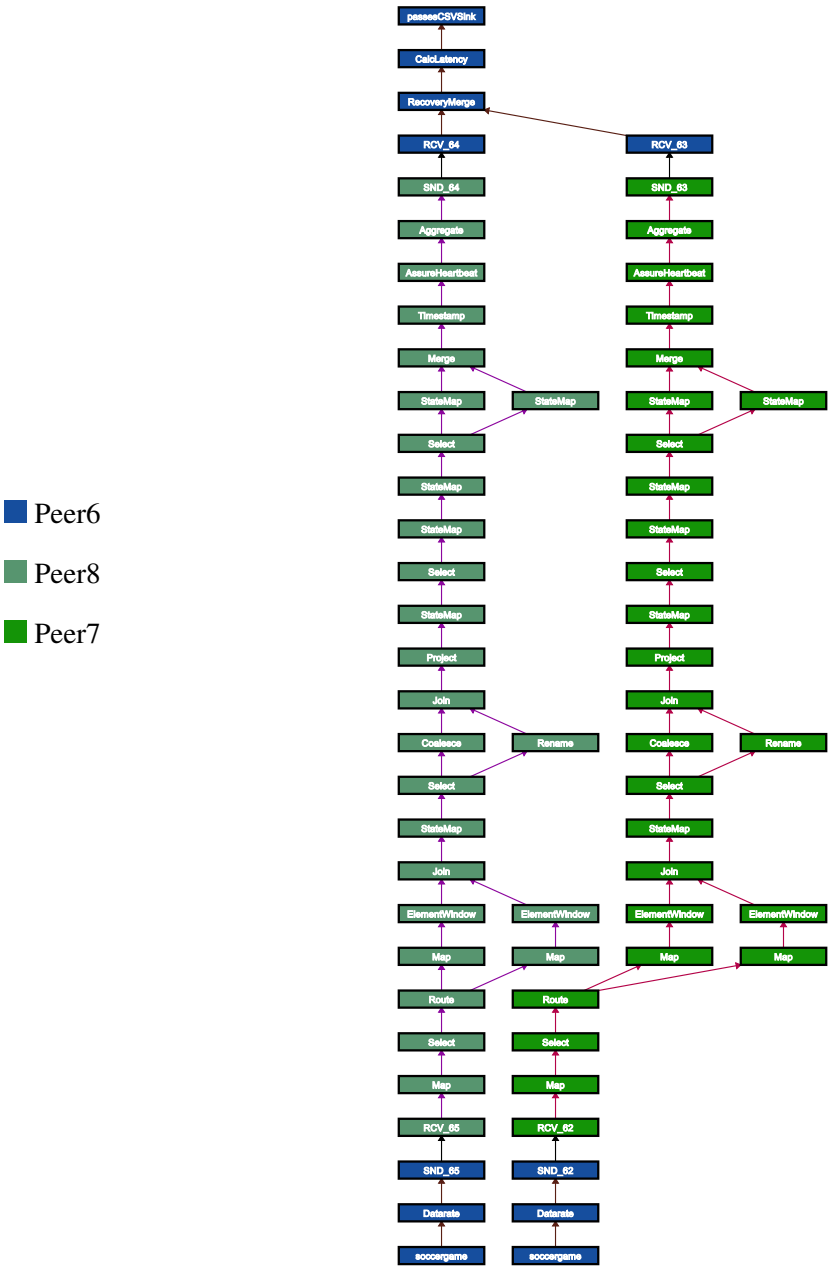


Abbildung A.11: PässeAS-Anfrageplan vor dem Ausfall

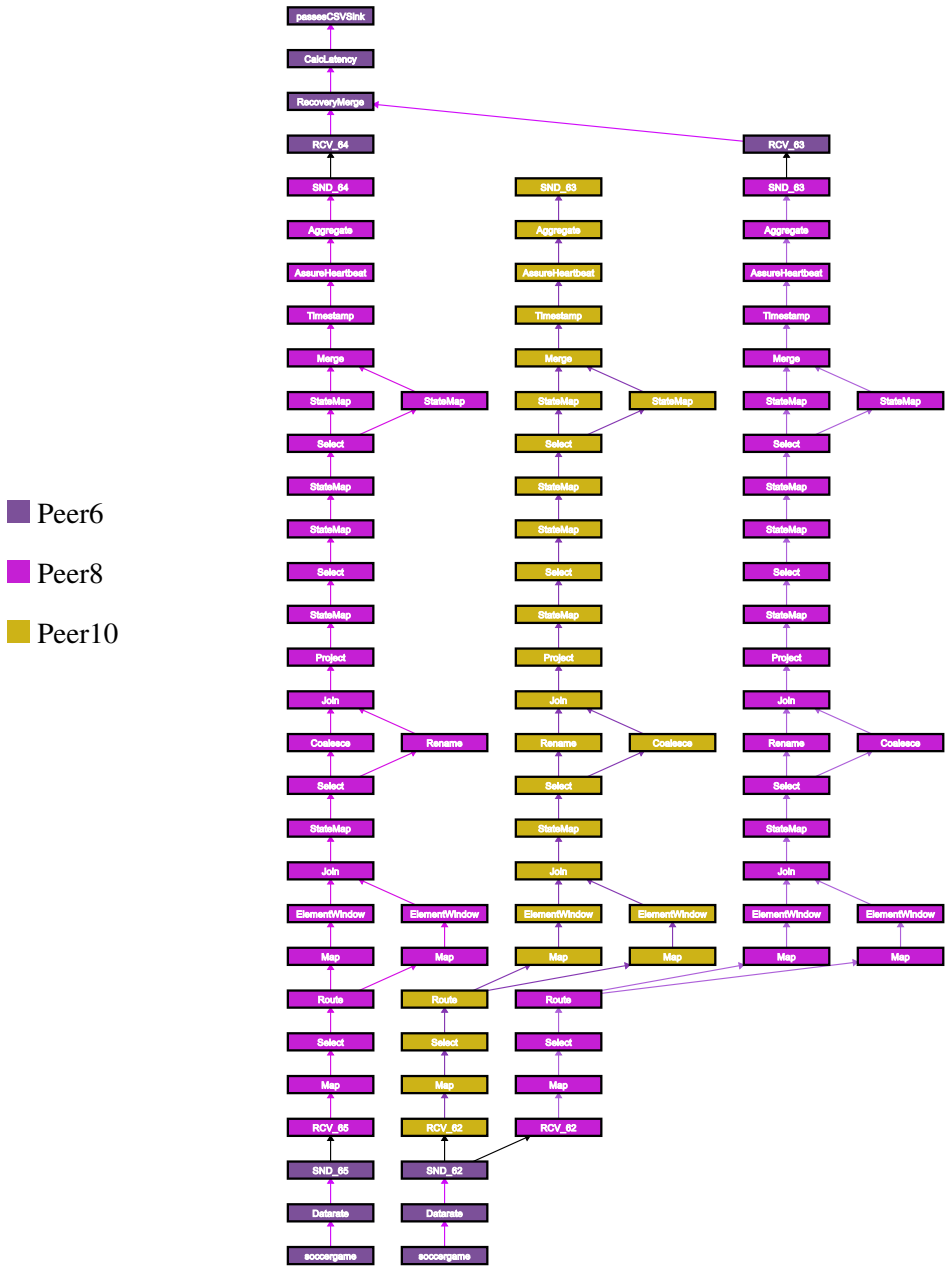


Abbildung A.12: PässeAS-Anfrageplan nach dem Recovery

Glossar

Nachfolgend sind noch einmal wesentliche Begriffe dieser Arbeit zusammengefasst und erläutert. Eine ausführliche Erklärung findet sich jeweils in den einführenden Abschnitten sowie der jeweils darin angegebenen Literatur. Das im Folgenden im Rahmen der Erläuterung verwendete Symbol $\sim$ bezieht sich jeweils auf den im Einzelnen vorgestellten Begriff, das Symbol $\uparrow$ verweist auf einen ebenfalls innerhalb dieses Glossars erklärten Begriff.

Abort $\sim$ meint im $\uparrow$Load-Balancing den Abbruch des $\uparrow$Load-Balancing Vorgangs aufgrund eines Fehlers. Tritt ein Fehler auf, werden (je nach $\uparrow$Load-Balancing Phase) an alle beteiligten Peers Abort-Nachrichten geschickt, die das bisher geschehene Rückgängig machen.

Acknowledgement Ein $\sim$ stellt eine Empfangsbestätigung innerhalb eines Kommunikationsprotokolls dar. In LSOP werden sie unter anderem für das $\uparrow$Upstream-Backup eingesetzt.

Active Standby $\sim$ bezeichnet eine Recovery-Technik, welche auf Replikation aufbaut. In der normalen Ausführung, führen zwei Knoten parallel die Bearbeitung des gleichen $\uparrow$Query-Parts aus. Dies soll zu einer schnellen Übernahme führen, sollte einer der beiden Knoten ausfallen.

Allokator Ein $\sim$, im Kontext von $\uparrow$Datenstrommanagementsystemen, weist einer Menge von $\uparrow$Query-Parts $\uparrow$Peers zu. Es handelt sich dabei um eine N:1-Beziehung zwischen $\uparrow$Query-Parts und $\uparrow$Peers.

Android $\sim$ ist ein Betriebssystem von Google für mobile Geräte, zum Beispiel Smartphones und Tablets. Es wird zum großen Teil als Open Source Projekt entwickelt und basiert auf dem Linux-Kernel. Es ist möglich, für Android-Systeme Anwendungen zu entwickeln.

ARBI Als $\sim$ werden die Rechnerräume der Information der Universität Oldenburg bezeichnet. Die „Abteilung Rechner- und Netzbetrieb Informatik“ stellt zudem Server bereit, die z.B. für Serveranwendungen genutzt werden können.

Artifactory Bei $\sim$ handelt es sich um einen Maven Dependency Manager. Mit Hilfe einer solchen Server Software lassen sich Java-Artefakte .jar austauschen. In unserem Fall wird der LSOPService durch Bamboo gebaut und die erstellte jar-File wird in das Artifactory geladen und kann dann durch Gradle-Dependencies genutzt werden. Neben dem reinen Austauschen von Artefakten dient eine solche Software auch als Dependency-Proxy, da die benötigten Dependencies von Repositories aus dem Internet abgerufen werden und dort zwischengespeichert werden.

Backup-Peer Ein $\sim$ bezeichnet in der $\uparrow$Recovery den Peer, der die Bearbeitung eines $\uparrow$Query-Parts absichern soll. In statischen Recovery-Techniken ist dieser vordefiniert und hat gegebenenfalls schon Informationen über den Zustand des bearbeitenden $\uparrow$Peers. In dynamischen Recovery-Techniken wird bei einem Ausfall ein Backup-Peer gewählt. Bei unserem $\uparrow$Active Standby gibt es keine strikte Unterteilung in $\uparrow$Haupt-Peer und $\uparrow$Backup-Peer, da jeweils die Ausgabe mit dem höchsten Zeitstempel verwendet wird.

Bamboo $\sim$ ist ein $\uparrow$Continious Integration System der Firma Atlassian. Durch den Einsatz einer solchen Software lässt sich der Entwicklungsprozess verbessern, da jederzeit klar ist ob eine Software noch kompilierbar ist. Darüber hinaus lassen sich auch Tests ausführen, Artefakte generieren, Code-Metriken erstellen und viele weitere Aufgaben. Derzeit ist unser System so eingerichtet, dass

alle drei Client-Projekte jeweils nach einem Commit im SVN einen Build durchführen (Continious Build) und einmal nächtlich (Nightly Build).

Catch-Up ∼ bezeichnet die Phase in der ↑Recovery, in der der ↑Peer danach strebt, die korrekte Bearbeitung der ↑Query wieder aufzunehmen. Ist die alte Ausfallsicherheit gegeben, ist die ↑Catch-Up-Phase abgeschlossen.

Code Coverage Die ∼ beschreibt wie gut ein Produktiv-Code durch entsprechende Unit-Tests abgedeckt ist. Um die ∼ zu ermitteln gibt es im Java Bereich verschiedene Tools. Bei uns kommt hierfür ↑JaCoco in Verbindung mit ↑SonarQube zum Einsatz. Dabei kann analysiert werden welche Zeile durch Tests erreicht wurde und vor allem bei if/else Blöcken welche der Ausführungspfade erreicht werden. Durch eine hohe ∼ und gute Tests kann die Qualität einer Software gesteigert werden. Die ∼ unserer Projekte ist in ↑SonarQube einsehbar.

Continuous Integration ∼ beschreibt eine Software, die die entwickelte Software kontinuierlich auf Kombilierbarkeit und Funktionsfähigkeit testen kann. Durch den Einsatz einer solchen Software lässt sich der Entwicklungsprozess verbessern, da jederzeit klar ist ob eine Software noch kompilierbar ist. Darüber hinaus lassen sich auch Tests ausführen, Artefakte generieren, Code-Metriken erstellen und viele weitere Aufgaben.Bei uns ist hierfür die Software ↑Bamboo von Atlassian im Einsatz.

Datahandler Ein ∼ ist in ↑Odysseus für das Lesen und Schreiben von Datenstromelementen eines externen Anwendungsprotokolls verantwortlich.

Datenstrommanagementsystem Ein ∼ (DSMS) ist analog zu einem Datenbankmanagementsystem (DBMS) für die Verwaltung von Datenströmen verantwortlich.

Deckungsschatten Der ∼ bezeichnet eine bestimmte Nutzung des eigenen Körpers und der Bewegung. Die Nutzung des ∼ ist somit eine Art zu verteidigen, welche vorrangig im Bereich der Raumdeckung genutzt wird und für das ↑Kettenspiel sehr wichtig ist. Indem sich der Spieler zwischen den Ball-führenden Gegenspieler und einer möglichen Anspielstation stellt, nimmt er diesen Gegner in den ∼ . Somit kann er ein Anspiel verhindern und nahezu unmöglich machen.

Dependency Injection Durch ∼ oder auch *Inversion of Control* lässt sich die feste Bindung zwischen Klassen bzw. Komponenten lösen. Das wird vor allem dadurch erreicht, dass abhängige Klassen nicht mehr durch die aufrufende Klasse mit new() erzeugt werden, sondern in die Klasse injiziert werden. Dadurch, dass die Klassen injiziert werden ist eine Austauschbarkeit von unterschiedlichen Implementierungen sehr einfach möglich. Durch die einfache Austauschbarkeit können die abhängigen Klassen bei der Ausführung von Tests durch Dummy-Objekte, sogenannte ↑Mocks, ersetzt werden. Durch den Einsatz von ↑Mocks kann dann die Funktionalität der abhängigen Klassen simuliert werden und auf diese Weise wird dann nur die Funktionalität der zu testenden Klasse getestet und nicht zusätzlich die der abhängigen Klassen.

Distributed Data Container Ein ∼ (DDC) ist ein Key-Value-Store für verteilte Key-Value-Paare in ↑Odysseus. Sein Konzept beinhaltet das Speichern und Laden solcher Paare sowie das Veröffentlichen im P2P-Netwerk.

Downstream ∼ meint im Kontext von Datenströmen die Richtung des Datenstromes, also hin zur Senke.

Eclipse Bei ∼ handelt es sich um eine Entwicklungsumgebung zur Entwicklung von Software verschiedenster Art. Dabei werden neben der Programmiersprache Java durch Erweiterungen auch andere Programmiersprachen unterstützt.

Epic ∼ bezeichnet in Scrum eine Oberkategorie einer ↑User Story. Dies dient vor allem dem Gliedern der User Stories in funktionale Zusammenhänge.

Executor Der ∼ ist eine Komponente in ↑Odysseus. Er ist für die Ausführung und das Management von ↑Queries zuständig.

Global Statistic Statistiken diesen Typs beinhalten Statistiken, die über Spieler oder Teams hinweg gelten. Darunter könnte z.B. der aktuelle Spielstand oder die Spielminute fallen.

Google Guice ∼ ist ein ↑Dependency Injection Framework für Java von Google. Dieses Framework wird bei uns in den Client Projekten eingesetzt um eine losere Kopplung zwischen den Komponenten herzustellen und den Einsatz von ↑Mocks in Tests zu ermöglichen.

Gradle Bei ∼ handelt es sich um ein Build-Tool ähnlich wie Maven, das vor allem im Android zum Einsatz kommt, aber auch bei herkömmlichen Java-Projekten eingesetzt werden kann. Primärer Nutzen eines solchen Build Tools ist die Angabe von Dependencies zu anderen Frameworks die durch das Build Tool automatisch aufgelöst werden und diese von Repositories heruntergeladen werden. Dabei werden vor allem auch transitive Abhängigkeiten aufgelöst und geladen. ↑Gradle bietet allerdings auch noch viele weitere Möglichkeiten auf die Erstellung einer Software einzuwirken, sodass beispielsweise Plugins eingebunden werden können, die eine spezielle Funktionalität integrieren, wie ↑Jacoco die ↑Code Coverage.

GuiceFX Bei ∼ handelt es sich um eine spezielle Erweiterung des ↑Google Guice Dependency Injection Frameworks für ↑JavaFX basierte Anwendungen. Grundlegende Aufgabe dieses Frameworks ist es, dass der eigentliche Injektionsvorgang innerhalb der View gestartet wird.

Haupt-Peer ∼ bezeichnet in der ↑Recovery den ↑Peer, der die Bearbeitung eines Query-Parts ausführt. Bei unserem ↑Active Standby gibt es keine strikte Unterteilung in ∼ und ↑Backup-Peer, da jeweils die Ausgabe mit dem höchsten Zeitstempel verwendet wird.

Initiierender Peer Der ∼ bezeichnet bei uns im aktiven ↑Load-Balancing denjenigen Peer, welcher das ↑Load-Balancing anstößt. Dieser Peer ist für die komplette Steuerung des ↑Load-Balancing Vorgangs zuständig und kommuniziert dazu mit mehreren ↑Slave Peers.

Intermediate Schema Das ∼ ist ein Zwischenschema, das allen Anfragen dieses Projektes zugrunde liegt. Der Sinn dieses ∼ ist es, eine Unabhängigkeit der Anfragen von dem Schema der ↑Sensoren zu erreichen. Dies führt zu einem verringerten Aufwand, wenn ↑Sensoren ausgetauscht werden. So muss bei einem neuen Sensorenschema lediglich ein Adapter geschrieben werden, der das Sensorenschema in das ∼ überführt.

JavaFX ∼ ist ein Framework für plattformübergreifende Rich-Client-Applications

JaCoco ∼ ist ein Plugin für ↑Gradle mit dem es möglich ist die ↑Code Coverage von Java-Projekten zu ermitteln. Die Coverage wird bei uns derzeit für die Projekte LSOPService und die DeveloperApp ermittelt und wird beim Continious Build in ↑Bamboo ermittelt. Die Ergebnisse dieser Code Metrik lassen sich übersichtlich in ↑SonarQube einsehen.

JUnit $\sim$ ist ein Testframework für Java. Durch das Erstellen von Unit-Tests lassen sich Fehler im Code ermitteln und eine bessere Code Qualität erreichen. Wichtig ist allerdings, dass der Einsatz von Unit-Tests nicht für eine fehlerfreie Software sorgen kann, da in Unit-Tests vor allem einzelne Klassen getestet werden sollen und nicht deren Zusammenspiel. Die Qualität von Unit-Tests kann durch den Einsatz von ↑Mocks weiter verbessert werden, da in diesem Fall keine abhängigen Klassen mehr zusätzlich getestet werden, sondern durch Dummy-Objekte (↑Mocks) ersetzt werden.

Kettenspiel Das $\sim$ oder auch Kettenverteidigung ist ein Verteidigungssystem, welches die Verteidiger in einer Raum-deckenden Anordnung formiert, die möglichst nah beieinander sein soll. Dadurch werden Pässe in die Schnittstelle verhindert und es ist für die Mannschaft möglich, mit Abseitsfalle zu spielen.

Kommunikator Der $\sim$ regelt im ↑Load-Balancing die Kommunikation zwischen den beteiligten ↑Peers.

Load-Balancing Eine Lastverteilung (engl. $\sim$) verteilt eine Last (Beanspruchung von CPU und Speicher) auf mehrere Kerne oder Rechner (auch ↑Peers). Es wird in ↑Odysseus unterschieden zwischen statischer und dynamischer Lastverteilung. Die statische Lastverteilung wird tritt bei dem Ausführen von Anfragen in Kraft. Die dynamische Lastverteilung überwacht die Auslastung zur Laufzeit und stößt gegebenenfalls eine Umverteilung der Last an.

Manndeckung $\sim$ ist eine Abwehrtaktik bei Mannschaftssportarten. Im Gegensatz zur ↑Raumdeckung, bei der die Passwege zugestellt werden, werden bei der $\sim$ einem Spieler ein direkter Gegenspieler zuordnet.

Master-Peer Im ↑Load-Balancing: ↑Initiierender Peer.

Mockito Bei $\sim$ handelt es sich um eine Java-Framework für das erstellen von ↑Mocks in Tests. In Verbindung mit der ↑Dependency Injection können in Tests bestehende Implementierung durch Dummy-Objekte (↑Mocks) ersetzt werden. ↑Mocks haben den entscheidenden Vorteil, dass diese konfigurierbar sind, also immer das erwartete Ergebnis liefern und somit die Fehlerquelle bei der Ausführung von Tests auf die zu testende Klasse verlagern. Darüber hinaus können ↑Mocks auch mitteilen, wie oft spezielle Methoden aufgerufen wurden.

Mocks Bei $\sim$ handelt es sich um Dummy-Objekte, die in Tests eingesetzt werden um die Fehlerquelle auf die zu testende Klasse zu beschränken und somit zu isolieren. Das Verhalten von $\sim$ kann zu Beginn des Tests festgelegt werden (beispielsweise der Rückgabewert einer Methode, wenn diese mit speziellen Parametern aufgerufen wird). Darüber hinaus können $\sim$ auch mitteilen, wie oft spezielle Methoden aufgerufen wurden.

Odysseus $\sim$ ist ein Framework zur Erstellung maßgeschneideter ↑Datenstrommanagementsysteme.

Operator Ein $\sim$ ist in dem Kontext von ↑Datenstrommanagementsystemen eine Einheit, die Operationen auf einem oder mehreren eingehenden Datenströmen ausführt und einen veränderten Ausgangsstrom produziert. $\sim$ werden in ↑Odysseus in logische und physische unterteilt, wobei logische $\sim$ bestimmen, was für eine Operation ausgeführt wird (z.B. das Verbinden zweier Eingabeströme) und physische $\sim$, wie diese Operation ausgeführt wird (ein konkreter Algorithmus).

Peer Ein ∼ ist ein Rechner in einem Peer-to-Peer-Netzwerk. ∼ sind durch ihre mögliche Heterogenität und Ressourcenautonomie ausgezeichnet.

Pipeline Eine ∼ im Datenstromkontext bezeichnet eine Abfolge von ↑Operatoren, die von einer Quelle Datenströme erhalten. Die ↑Operatoren bearbeiten jeden dieser Datenströme gemäß ihrer Reihenfolge. Schlussendlich führt dies zu einer Ausgabe.

Player Statistic Eine Statistik diesen Typs beinhaltet Informationen zu einem Spieler. Es könnte z.B. der Ballbesitz eines Spielers darunter fallen.

Procedural Query Language Die ∼ (PQL) ist eine ↑Odysseus-eigene Sprache um Datenstromanfragen (↑Queries) zu formulieren. Es handelt sich dabei um eine prozedurale, funktionale Sprache im Gegensatz zu z.B. SQL bei Datenbanken.

Protocolhandler Ein ∼ ist in ↑Odysseus für das Konvertieren eines externen Anwendungsprotokolls in eine ↑Odysseus-interne Repräsentation verantwortlich.

Query Eine Anfrage (engl. ∼) ist in ↑Odysseus ein Konstrukt, das im wesentlichen einen logischen Anfrageplan, einen Nutzer und zusätzliche Parameter enthält.

Query-Part Ein ∼ ist eine Sammlung von logischen ↑Operatoren, die zwar der selben ↑Query angehören, aber nicht zwingend direkt mit einander verbunden sein müssen. Der Sinn von ∼ ist es auszusagen, welche ↑Operatoren auf ein und demselben ↑Peer ausgeführt werden sollen.

Raumdeckung ∼ ist eine Abwehrtaktik bei Mannschaftssportarten. Im Gegensatz zur Manndeckung, die einem Spieler einen direkten Gegenspieler zuordnet, werden bei der ∼ die Passwege zugestellt.

Recovery ∼ bezeichnet Sicherheitsmechanismen und Maßnahmen, die gegenüber Fehlerfällen in Datenbank- oder ↑Datenstrommanagementsystemen getroffen werden. Im P2P und Datenstrom Kontext wird vor allem die Absicherung vor ↑Peer Ausfällen betrachtet. Diese können jeder Zeit im System auftreten, weshalb eine gewisse Ausfallsicherheit gegeben sein sollte. Diese wird über verschiedene ∼ -Techniken realisiert. Für ↑Odysseus P2P sind derzeit ↑Active Standby und ↑Upstream-Backup vorgesehen.

Replication ∼ im Kontext von ↑Odysseus P2P bezeichnet die mehrfache Ausführung von ↑Query-Parts.

Roboguice ∼ ist eine Erweiterung von ↑Google Guice für den Einsatz der ↑Dependency Injection unter ↑Android. Dabei ist die grundlegende Aufgabe den eigentlichen Injektionsvorgang innerhalb der Activity zu starten. Darüber hinaus bietet ∼ auch noch die Möglichkeit Views zu injecten.

Robolectric ∼ ist eine Erweiterung der normalen Unit-Tests unter ↑Android. Da normalen Tests unter ↑Android es erfordern, dass entweder der Emulator gestartet wird oder ein entsprechendes Endgerät angeschlossen ist, ist eine einfach Testbarkeit erstmal nicht möglich. Auch ein Ausführung der Tests in ↑Continious Integration-Systemen ist nahezu unmöglich. Aus diesem Grund werden ∼ Tests geschrieben. Das Framework emuliert mit sogenannten Schatten-Objekten die ↑Android Umgebung und ermöglicht somit ein Ausführen von Tests ohne das Starten des Emulators oder das Android SDK.

Round-Robin $\sim$ meint eine zirkuläre Verteilung. Beginnend bei einem Ziel werden alle folgenden Ziele reihum ausgewählt.

Sensor Ein $\sim$ ist eine Hardware, die physikalische Gegebenheiten der realen Welt (z.B. Position eines Objekts oder die Raumtemperatur) in digitale Signale überführt, die wiederum ausgewertet werden können. Man unterscheidet zwischen aktiven $\sim$, die Daten schicken und passiven $\sim$, bei denen die Daten abgeholt werden müssen.

Shared-Query-Id Eine $\sim$ ist eine eindeutige Id, die einer verteilten Anfrage zugeordnet wird.

Slave Peer Im ↑Load-Balancing bezeichnen wir jeden ↑Peer, der neben dem ↑initiierenden Peer noch am ↑Load-Balancing beteiligt ist als $\sim$. Dazu gehört zum einen der freiwillige ↑Peer, d.h. der ↑Peer, der den ungewünschten ↑Query übernehmen wird. Zum anderen zählen dazu aber auch die ↑Peers, deren Stream ümgebogen"werden muss, damit der Datenstrom über den neuen ↑Peer läuft.

SonarQube $\sim$ ist eine serverbasierte Software, die es ermöglicht, Code-Metriken zu erfassen und auszuwerten. Unter anderem werden ↑Code Coverage, Testausführung, Duplikatserkennung, Komplexität, Lines of Code und Einhaltung von Regeln analysiert.

SportsQL $\sim$ ist eine Anfragesprache mit JSON-Syntax für Sport-Statistik-Anfragen. Ihr Vorteil gegenüber ↑Procedure Query Language ist ihr einfacher und schlanker Syntax. Dies schränkt allerdings auch die Erweiterbarkeit ein.

Task Ein $\sim$ stellt in Scrum eine (Teil-)Aufgabe dar, welche durch den Bearbeiter, der sich diesen $\sim$ zuweisen kann, erledigt wird. Er soll möglichst so formuliert werden, dass er unabhängig von anderen Tasks bearbeitet werden kann. Ein $\sim$ kann zu einer ↑User Story gehören und wird damit zu einem Subtask dieser ↑User Story.

Take-Over Der $\sim$ bezeichnet die Phase der ↑Recovery, in der der ↑Backup-Peer damit beginnt, die Bearbeitung zu übernehmen.

Team Statistic Statistiken diesen Typs betrachten immer ein vollständiges Team. Es könnte z.B. der Ballbesitz eines Teams darunter fallen.

Teilanfrage ↑Query-Part

Transporthandler $\sim$ sind in ↑Odysseus für die Kommunikation zwischen externen Quellen und ↑Odysseus verantwortlich.

Upstream $\sim$ meint im Kontext von Datenströmen die Richtung entgegen des Datenstromes, also hin zur Quelle.

Upstream-Backup Das $\sim$ bezeichnet eine ↑Recovery-Technik, bei der bei einem Ausfall dynamisch ein neuer ↑Peer gefunden wird. Diese Technik heißt $\sim$, da noch nicht vollständig abgearbeitete Tupel ↑Upstream zwischengespeichert werden. Bei einem Ausfall kann der neue Peer mit den noch zu verarbeitenden Tupeln versorgt werden, so dass durch einen Ausfall weniger Tupel verloren gehen. Über ↑Acknowledgements wird eine Kommunikation zu den Upstream-Peers hergestellt.

User Story Eine $\sim$ stellt in Scrum ein Konstrukt dar, welches zusammengehörige Aufgaben umfasst. Eine $\sim$ kann ein oder mehrere ↑Tasks beinhalten. Die $\sim$ gilt als abgeschlossen, wenn alle ihre Subtasks abgeschlossen sind. Eine $\sim$ kann einem übergeordneten ↑Epic angehören.

Velocity In SCRUM bezeichnet die $\sim$ die Geschwindigkeit, in der Story-Points abgearbeitet werden. Sie dient dazu, abzuschätzen, wie viel in einem Sprint geschafft werden kann. Die $\sim$ berechnet sich als Verhältnis von benötigter Zeit zu geschafften Storypoints.

Abkürzungen

AC Admission Control . . . 43
ACM Association for Computing Machinery . . . 291
AoA Angle of Arrival . . . 16
ARBI Abteilung Rechner- und Netzbetrieb Informatik . . . 282
BTU BaseTimeUnit . . . 135
BTW Datenbanksysteme für Business, Technologie und Web . . . 291
CAB Contextual Action Bar . . . 208
CEP Complex Event Processing . . . 32
CI Continuous Integration . . . 104
COO Cell of Origin . . . 17
CQL Continuous Query Language . . . 38
DBMS Datenbankmanagementsystem . . . 33
DDC Distributed Data Container . . . 134
DEBS Distributed Event-Based Systems . . . 114
DFB Deutscher Fußball Bund . . . 26
DHT Distributed Hashtables . . . 81
DI Dependency Injection . . . 154
DSMS Datenstrommanagementsystem . . . I
DSEP Data Streams and Event Processing . . . 295
ERP Endpoint Routing Protocol . . . 89
ETW Eat the Whistle . . . 165
ExpG Expected Goals . . . 7
FSPL Free-space path loss . . . 132
GK Gauß-Krüger . . . 131
GNSS Global Navigation Satellite System . . . 19
GPS Global Positioning System . . . 13
GUI Graphical User Interface . . . 112
IIS Institut für integrierte Schaltungen . . . 24
JSON JavaScript Object Notation . . . 139
KDD Knowledge Discovery in Databases . . . 5
LSOP Liveanalysen im Sport mit Odysseus P2P . . . 99
NBA National Basketball Association . . . 9
OSGi Open Services Gateway initiative . . . 57
P2P Peer-to-Peer . . . 75

PBP Pipe Binding Protocol 89
PDP Peer Discovery Protocol 88
PG Projektgruppe 291
PIP Peer Information Protocol 88
PPS Pufferplatzierungsstrategie 38
PQL Procedural Query Language 38
PRP Peer Resolver Protocol 89
REST Representational State Transfer 179
RFID Radio Frequency Identification 18
RKT Real Time Kinematic 131
ROI Return on Investment 94
RSA Rational Software Architect 303
RSSI Received Signal Strength Indication 17
RTLS Real-Time Locating System 17
RvP Rendezvous Protocol 89
SAC Symposium on Applied Computing 291
ScS Scheduling-Strategie 38
SDK Software Development Kit 91
SOAP Simple Object Access Protocol 290
SVN Subversion 299
TDoA Time Difference of Arrival 16
ToA Time of Arrival 16
TTL Time-to-live 79
WLAN Wireless Local Area Network 18
WSN Wireless Sensor Network 19

Abbildungen

Tabellenverzeichnis

Literatur

[AAB+05] ABADI1, D. J. ; AHMAD, Y. ; BALAZINSKA, M. ; CETINTEMEL, U. ; CHERNIACK, M. ; HWANG, J. ; LINDNER, W. ; MASKEY, A. S. ; RASIN, A. ; RYVKINA, E. ; TATBUL, N. ; XING, Y.: *The Design of the Borealis Stream Processing Engine*. 2005

[ABB+04] ARASU, A. ; BABCOCK, B. ; BABU, S. ; CIESLEWICZ, J. ; DATAR, M. ; ITO, K. ; MOTWANI, R. ; SRIVASTAVA, U. ; WIDOM, J.: *STREAM: The Stanford Data Stream Management System*. 2004

[AD02] AGRAWAL, Rakesh (Hrsg.) ; DITTRICH, Klaus R. (Hrsg.): *Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, February 26 - March 1, 2002*. IEEE Computer Society, 2002 . – ISBN 0–7695–1531–2

[AGG+12] APPELRATH, Hans-Jürgen ; GEESEN, Dennis ; GRAWUNDER, Marco ; MICHELSEN, Timo ; NICKLAS, Daniela: Odysseus: a highly customizable framework for creating efficient event stream management systems. In: BRY, François (Hrsg.) ; PASCHKE, Adrian (Hrsg.) ; EUGSTER, Patrick T. (Hrsg.) ; FETZER, Christof (Hrsg.) ; BEHREND, Andreas (Hrsg.): *DEBS*, ACM, 2012. – ISBN 978–1–4503–1315–5, S. 367–368

[Bec10] BECHTOLD, Max: *Selektivitätsabschätzung und Kostenmodelle. Ein Überblick über grundlegende Werkzeuge der Anfrageverarbeitung in relationalen Datenbanksystemen*. 2010

[Ber10] BERKUS, Josh: *Admission Control and its Discontents*. Webseite, 2010. – Online erhältlich: http://it.toolbox.com/blogs/database-soup/admission-control-and-its-discontents-39895; zuletzt besucht am 11. April 2014.

[BGJ+09] BOLLES, Andre ; GRAWUNDER, Marco ; JACOBI, Jonas ; NICKLAS, Daniela ; APPELRATH, Hans-Jürgen: Odysseus: Ein Framework für massgeschneiderte Datenstrommanagementsystem. In: *GI Jahrestagung*, 2009, S. 2000–2014

[BGKS02] BROOKSHIER, Daniel ; GOVONI, Darren ; KRISHNAN, Navaneeth ; SOTO, Juan C.: *JXTA: Java P2P Programming*. Sams Publishing, 2002

[BH14] BERNADETTE HÖRNER, Johann Theo H. Dominik Hörner H. Dominik Hörner: *Neuer Anlauf für mehr Technik im Fußball*. `http://www.sportblog.cc/neuer-anlauf-fur-mehr-technik-im-fusball/`, 2014. – Online, zuletzt geöffnet: 28.03.2015

[BHS08] BALAZINSKA, M. ; HWANG, J. ; SHAH, M. A.: *Fault-tolerance and high availability in data stream management systems*. 2008

[Bil10] BILL, Ralf: *Grundlagen der Geo-Informationssysteme*. 5., völlig neu bearb. Aufl. Berlin ; Offenbach : Wichmann, 2010. – XV, 804 S.. – ISBN 978–3–87907–489–1. – Erschien bis 4. Aufl. als mehrbändiges Werk

[BLMM04] BECKER, Robert ; LIEBSCH, Franziska ; MICHEL, Yann-Rudolf ; MÜLLER, Daniel: *JXTA: Einführung und Überblick*. 2004

[Bra13a] BRAND, Michael: *Lastverteilung für das Datenstrommanagementframework Odysseus*, Carl von Ossietzky Universität Oldenburg, Bachelorarbeit, 2013

[Bra13b] BRANDSTÄTER, J.: *Agile IT-Projekte erfolgreich gestalten: Risikomanagement als Ergänzung zu Scrum*. Springer Fachmedien Wiesbaden, 2013 (SpringerLink : Bücher). http://books.google.de/books?id=fmkHngEACAAJ. – ISBN 9783658044299

[Cor14] CORPORATION, Zebra T.: *Location Solutions (RTLS) Zebra Technologies*. http://www.zebra.com/gb/en/solutions/location-solutions/asset-visibility.html. Version: 2014

[DBG+06] DATTA, Souptik ; BHADURI, Kanishka ; GIANNELLA, Chris ; WOLFF, Ran ; KARGUPTA, Hillol: Distributed Data Mining in Peer-to-Peer Networks. In: *IEEE Internet Computing* 10 (2006), Juli, Nr. 4, 18–26. http://dx.doi.org/10.1109/MIC.2006.74. – DOI 10.1109/MIC.2006.74. – ISSN 1089–7801

[EFGK03] EUGSTER, Patrick T. ; FELBER, Pascal A. ; GUERRAOUI, Rachid ; KERMARREC, Anne-Marie: The Many Faces of Publish/Subscribe. In: *ACM Computing Surveys* (2003)

[elf14] *How to scout a striker*. http://11tegen11.net/2014/02/15/how-to-scout-a-striker/, 2014. – [Online; Stand 07.05.2014]

[EN10] ETZION, Opher ; NIBLETT, Peter: *Event Processing in Action*. 1st. Greenwich, CT, USA : Manning Publications Co., 2010. – ISBN 1935182218, 9781935182214

[FKU14] FALKENBERG, Guido ; KISKER, Dr. H. ; URBANSKI, Jürgen: Big-Data-Technologien Wissen für Entscheider / BITKOM. 2014. – Forschungsbericht

[GFW+11] GRÜN, Thomas von d. ; FRANKE, Norbert ; WOLF, Daniel ; WITT, Nicolas ; EIDLOTH, Andreas: A real-time tracking system for football match and training analysis. In: *Microelectronic Systems*. Springer, 2011, S. 199–212

[GJPPM+12] GULISANO, Vincenzo ; JIMENEZ-PERIS, Ricardo ; PATINO-MARTINEZ, Marta ; SORIENTE, Claudio ; VALDURIEZ, Patrick: StreamCloud: An Elastic and Scalable Data Streaming System. In: *IEEE Transactions on Parallel and Distributed Systems* 23 (2012), Nr. 12, S. 2351–2365. – ISSN 1045–9219

[Glo13] GLOGER, Boris: *Scrum : Produkte zuverlässig und schnell entwickeln*. 4., überarb. Aufl. München : Hanser, 2013 http://files.hanser.de/hanser/docs/20121130_21211314455-106_978-3-446-43338_0_Vorwort.pdf. – ISBN 9783446433380

[Gmb14] GMBH inmotiotec: *Wenn die Gegenwart alt aussieht, hat die Zukunft begonnen*. http://www.abatec-ag.com/inmotiotec-rtls/. Version: 2014

[GOT02] GONG, By L. ; OAKS, Scott ; TRAVERSAT, Bernard: *JXTA in a nutshell*. O'Reilly & Associates, Inc., 2002

[Gra] GRAWUNDER, Marco: *Data Mining; Vorlesungsfolien*

[Gra02] GRADECKI, Joseph D.: *Mastering JXTA - Building Java Peer-to-Peer Applications*. John Wiley & Sons, 2002

[Gra12] GRAWUNDER, Dr. M.: *Verarbeitung von Datenströmen*. 2012. – Vorlesungsfolien Informationssysteme III

[Gra14] GRAWUNDER, Marco: *The Odysseus Operator Framework*. http://odysseus.informatik.uni-oldenburg.de:8090/display/ODYSSEUS/The+Odysseus+Operator+Framework, 2014. – Online, zuletzt geöffnet: 28.04.2014

[Grü13a] GRÜN, Thomas von d.: *Funkbasierte Lokalisierungstechnologien*. http://www.angewandte-kartographie.de/download/symposium2013/vortraege/Von_der_Gruen_RedFIR.pdf, Mai 2013

[Grü13b] GRÜN, Thomas von d.: *Funkbasierte Lokalisierungstechnologien – RedFIR - Ortung im Sport, am Flughafen und in der Logistik*. 2013

[GS02] GRAHAM, Ross L. (Hrsg.) ; SHAHMEHRI, Nahid (Hrsg.): *2nd International Conference on Peer-to-Peer Computing (P2P 2002), 5-7 September 2002, Linköping, Sweden*. IEEE Computer Society, 2002 . – ISBN 0–7695–1810–9

[Hać89] HAĆ, Anna: Load balancing in distributed systems: A summary. In: *ACM SIGMETRICS Performance Evaluation Review* 16 (1989), Nr. 2-4, 17–19. http://dl.acm.org/citation.cfm?id=1041912

[HBR+05] HWANG, J. ; BALAZINSKA, M. ; RASIN, A. ; CETINTEMEL, U. ; STONEBRAKER, M. ; ZDONIK, S.: *High-Availability Algorithms for Distributed Stream Processing*. 2005

[Hig02] HIGHSMITH, J.A.: *Agile Software Development Ecosystems*. Addison-Wesley, 2002 (Detective Inspector Carol Ashton mystery). http://books.google.de/books?id=uE4FGFOHs2EC. – ISBN 9780201760439

[HSH07] HELLERSTEIN, Joseph M. ; STONEBRAKER, Michael ; HAMILTON, James: Architecture of a Database System. In: *Foundations and Trends in Databases*, 2007, S. 141–259

[Iak12] IAKAB, M Sc CS Kinga K.: *Probabilistic Quorum Systems for Dependable Distributed Data Management*, Carl von Ossietzky Universität Oldenburg, Diss., 2012

[Ins14] INSTITUT, Fraunhofer: *Eine Empfängereinheit von RedFIR*. http://www.iis.fraunhofer.de/de/bf/ln/referenzprojekte/redfir/_jcr_content/contentPar/tabview_parsys/tabview_tab_1/tabPar/textblockwithpics_2/image1.img.jpg/Technologie_1.1343048035430.jpg, April 2014. – letzter Zugriff am 20.04.2014

[IS11] ISHII, A. ; SUZUMURA, T.: Elastic Stream Computing with Clouds. In: *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, 2011, 195–202

[Jar06] JAROSCH, Prof. Dr.-Ing. M.: *Vorlesungsmanuskript zur Grundlagenvorlesung Vermessung*. 2006

[jxt14] *JXTA - The Language and Platform Independent Protocol for P2P Networking (letzter Aufruf: 17.04.2014)*. https://jxta.kenai.com/, 2014

[KBL06] KIFER, M. ; BERNSTEIN, A. ; LEWIS, P.: *Database Systems, An Application-Oriented Approach Second Edition*. Pearson Education Inc., New York, 2006

[KGJ08] KO, Steven Y. ; GUPTA, Indranil ; JO, Yookyung: A New Class of Nature-inspired Algorithms for Self-adaptive Peer-to-peer Computing. In: *ACM Trans. Auton. Adapt. Syst.* 3 (2008), August, Nr. 3, 11:1–11:34. http://dx.doi.org/10.1145/1380422.1380426. – DOI 10.1145/1380422.1380426. – ISSN 1556–4665

[Kin13] KINOVEA: *Kinovea Videoanalyse*. http://www.kinovea.org/images/slides/004-speed2.jpg, 2013. – Online, zuletzt geöffnet: 28.03.2015

[KKP11] KLEIMINGER, Wilhelm ; KALYVIANAKI, Evangelia ; PIETZUCH, Peter: Balancing load in stream processing with the cloud. In: *Data Engineering Workshops (ICDEW), 2011 IEEE 27th International Conference on* IEEE, 2011, 16–21

[Knu14] KNUTSON, Ted: *Radardiagramm Arjen Robben*. http://statsbomb.com/2014/04/radar-love-robben-reus-sanchez-navas-firmino-griezmann/, 2014. – Online, zuletzt geöffnet: 28.03.2015

[Kos00] KOSSMANN, Donald: The State of the art in distributed query processing. In: *ACM Comput. Surv.* 32 (2000), Nr. 4, S. 422–469

[Kös08] KÖSTER, Philipp: *Noten für Fußballer: Setzen, Sechs!* http://www.spiegel.de/sport/fussball/noten-fuer-fussballer-setzen-sechs-a-543832.html, 2008. – [Online; Stand 07.05.2014]

[Krä07] KRÄMER, Jürgen: *Continuous queries over data stream - semantics and implementation*, Diss., 2007. – I–XXII, 1–291 S.

[KS04] KRÄMER, Jürgen ; SEEGER, Bernhard: PIPES: A Public Infrastructure for Processing and Exploring Streams. In: *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA : ACM, 2004 (SIGMOD '04). – ISBN 1–58113–859–8, 925–926

[Kub] KUBATKO, Justin: *Basketball Glossary*. http://www.basketball-reference.com/about/glossary.html, . – [Online; Stand 07.05.2014]

[Kud92] KUDLICH, Hermann: *Verteilte Datenbanken : Systemkonzepte und Produkte*. Berlin [u.a.] : Verl. Siemens-Aktienges., 1992 http://ubprp1/records/PAD_ALEPH000610635. – ISBN 3–8009–1589–8, 9783800915897, 3800915898

[Kud07] KUDRASS, Thomas: *Datenbanken*. Carl Hanser Verlag, Leipzig, 2007

[Kuk14a] KUKA, Christian: *Access Framework*. http://odysseus.offis.uni-oldenburg.de:8090/display/ODYSSEUS/Access+framework, 2014. – Online, zuletzt geöffnet: 15.04.2014

[Kuk14b] KUKA, Christian: *Access Operator*. http://odysseus.offis.uni-oldenburg.de:8090/display/ODYSSEUS/Access+operator, 2014. – Online, zuletzt geöffnet: 15.04.2014

[Kuk14c] KUKA, Christian: *Procedural Query Language (PQL)*. http://odysseus.offis.uni-oldenburg.de:8090/pages/viewpage.action?pageId=4587829, 2014. – Online, zuletzt geöffnet: 15.04.2014

[Kuk14d] KUKA, Christian: *Protocol Handler*. http://odysseus.offis.uni-oldenburg.de:8090/display/ODYSSEUS/Protocol+handler, 2014. – Online, zuletzt geöffnet: 15.04.2014

[Kuk14e] KUKA, Christian: *Transport Handler*. http://odysseus.offis.uni-oldenburg.de:8090/display/ODYSSEUS/Transport+handler, 2014. – Online, zuletzt geöffnet: 15.04.2014

[Kün12] KÜNNETH, Thomas: *Android 4 - Apps entwickeln mit dem Android SDK*. GalileoPress, 2012

[Lau13] *Unermüdlich - aber kaum Torgefahr*. http://www.bundesliga.de/de/liga/news/2012/unermuedlich-aber-kaum-torgefahr_0000245840.php, 2013. – Online, zuletzt geöffnet: 28.03.2015

[LY08] LOGOTHETIS, D. ; YOCUM, K.: *Wide-Scale Data Stream Management*. 2008

[Man11] MANDL, Daniel: *Milan-Lab – der AC Milan als weltweiter Vorreiter der medizinischen Individualanalyse*. http://www.abseits.at/fusball-international/italien/milan-lab-der-ac-milan-als-weltweiter-vorreiter-der-medizinischen-individualanalysel, 2011. – [Online; Stand 07.05.2014]

[Max13] MAXIMINI, D.: *Scrum - Einführung in der Unternehmenspraxis: Von starren Strukturen zu agilen Kulturen*. Springer Berlin Heidelberg, 2013 (SpringerLink : Bücher). https://books.google.de/books?id=hVt0yiGfHVUC. – ISBN 9783642348235

[Mic11] MICHELSEN, Timo: *Admission Control in Odysseus*, Carl von Ossietzky Universität Oldenburg, Masterarbeit, 2011

[Mic14] MICHELSEN, Timo: Data Stream Processing in Dynamic and Decentralized Peer-to-Peer Networks. In: *SIGMOD 2014 PhD Symposium* (2014)

[Mit95] MITSCHANG, Bernhard: *Anfrageverarbeitung in Datenbanksysteme*. Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 1995

[NAMS13] NOR AIDA MAHIDDIN, Siti Dhalila Engku Fadzli Hasan Suhailan S. Elissa Nadia Madi M. Elissa Nadia Madi ; SAFIE, Noaizan: User Position Detection In An Indoor Environment. Version: 2013. http://dx.doi.org/10.14257/ijmue.2013.8.5.30. 2013. – Forschungsbericht

[NTM12] NEUS, Sebastian ; TROMPETER, Jens ; MANDISCHER, Martin: *Scrum Kompakt*. http://www.scrum-kompakt.de/scrum-kompakt-e-book/, 2012. – Online, zuletzt geöffnet: 25.03.2015

[ÖV99] ÖZSU, M. T. ; VALDURIEZ, Patrick: *Principles of Distributed Database Systems, Second Edition*. Prentice-Hall, 1999

[Pim09] PIMENTEL, Roger: *Lakers vs. Magic: A Mathematical Breakdown of Matchups*. http://howtowatchsports.com/2009/06/lakers-vs-magic-a-mathematical-breakdown-of-matchups/1, 2009. – [Online; Stand 07.05.2014]

[PPKG03] PALPANAS, Themistoklis ; PAPADOPOULOS, Dimitris ; KALOGERAKI, Vana ; GUNOPULOS, Dimitrios: Distributed Deviation Detection in Sensor Networks. In: *SIGMOD Rec.* 32 (2003), Dezember, Nr. 4, 77–82. http://dx.doi.org/10.1145/959060.959074. – DOI 10.1145/959060.959074. – ISSN 0163–5808

[RDS11] RYAN DOBBINS, Saul G. ; SHAW, Brian: Software Defined Radio Localization Using 802.11-style Communications. 2011. – Forschungsbericht

[ref15] REFERENCE.COM basketball: *James Harden 2012-13 Shooting*. http://www.basketball-reference.com/players/h/hardeja01/shooting/2013, März 2015. – letzter Zugriff am 19.03.2015

[Röh13] RÖHRIG, Prof. Dr. C.: *Lokalisierungsverfahren für drahtlose Sensornetzwerke*. 2013

[Sch12] SCHWOTZER, Dr. T.: *Prinzipien der Ortung*. 2012

[SHB04] SHA, M. A. ; HELLERSTEIN, J. M. ; BREWER, E.: *Highly Available, Fault Tolerant, Parallel Dataflows*. 2004

[Spr13] *Die Raketen der Liga*. http://www.bundesliga.de/de/liga/news/2012/die-raketen-der-liga_0000245847.php, 2013. – Online, zuletzt geöffnet: 28.03.2015

[SRKZ12] SCHRÜFER, E. ; REINDL, L.M. ; KÖNIG, A. ; ZAGAR, B.: *Elektrische Messtechnik: Messung elektrischer und nichtelektrischer Größen*. Carl Hanser Verlag GmbH & Company KG, 2012 http://books.google.de/books?id=3alPAgAAQBAJ. – ISBN 9783446433298

[SS07] SCHILL, Alexander ; SPRINGER, Thomas: *Verteilte Systeme*. Springer-Verlag Berlin Heidelberg, 2007

[SS13] SCHWABER, Ken ; SUTHERLAND, Jeff: *Scrum Guide*. http://www.scrumguides.org/scrum-guide.html, 2013. – Online, zuletzt geöffnet: 25.03.2015

[SSC10] SCHUMAKER, Robert P. ; SOLIEMAN, Osama K. ; CHEN, Hsinchun: *SPORTS DATA MINING*. Springer Verlag, 2010

[Stö01] STÖRL, U.: *Backup und Recovery in Datenbanksystemen*. B.G. Teubner, Stuttgart, Wiesbaden, Leipzig, 2001

[Sun07a] SUN MICROSYSTEMS (Hrsg.): *JXTA JavaTM Standard Edition v2.5: Programmers Guide*. Sun Microsystems, 2007

[Sun07b] SUN MICROSYSTEMS INC. (Hrsg.): *JXTA v2.0 Protocols Specification*. Sun Microsystems Inc., 2007

[Sut10] SUTHERLAND, Jeff: *Scrum Handbook*. `http://jeffsutherland.com/scrumhandbook.pdf`, 2010. – Online, zuletzt geöffnet: 25.03.2015

[SW05] SMITH, J. ; WATSON, P.: *Fault Tolerance in Distributed Query Processing*. 2005

[TLRG08] TANIAR, David ; LEUNG, Clement H. C. ; RAHAYU, J. W. ; GOEL, Sushant: *High Performance Parallel Database Processing and Grid Databases*. John Wiley & Sons, 2008. – ISBN 978–0–470–10762–1

[TS08] T. STRANG, S. Thölert R. Oberweis et a. F. Schubert S. F. Schubert: *Lokalisierungsverfahren*. 2008

[Ver10] VERSTRYNGE, Jérôme: *Practical JXTA II*. Lulu.com, 2010

[VLO10] VU, Quang H. ; LUPU, Mihai ; OOI, Beng C.: *Peer-to-Peer Computing - Principles and Applications.* Springer, 2010. – I–XVI, 1–317 S. – ISBN 978–3–642–03513–5

[Wer03] WERTHER, Walter: *Verkehrscharakterisierung von JXTA*, Technische Universität München - Lehrstuhl für Kommunikationsnetze, Diplomarbeit, 2003

[who] *WhoScored Ratings Explained*. `http://www.whoscored.com/Explanations`, . – [Online; Stand 07.05.2014]

[WSGÖ08] WANG, Weihan ; SHARAF, Mohamed A. ; GUO, Shimin ; ÖZSU, M. T.: Potential-driven load distribution for distributed data stream processing. In: LEE, Byung S. (Hrsg.): *SSPS*, ACM, 2008 (ACM International Conference Proceeding Series). – ISBN 978–1–59593–963–0, S. 13–22

[Wüt08] WÜTHERICH, G.: *Die OSGi Service Platform: Eine Einführung mit Eclipse Equinox*. Dpunkt.Verlag GmbH, 2008. – ISBN 9783898644570

[XZH05] XING, Ying ; ZDONIK, Stan ; HWANG, Jeong-Hyon: Dynamic load distribution in the borealis stream processor. In: *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on* IEEE, 2005, 791–802

[Zeh98] ZEHNDER, C. A.: *Informationssysteme und Datenbanken 6. Auflage*. B.G. Teubner, Stuttgart, 1998

[Zei15] ZEITUNG, Süddeutsche: *Beispiel für Fußballstatistiken im Datacenter*. `http://sportergebnisse.sueddeutsche.de/sueddeutsche/fussball/1bundesliga/datencenter-statistik.html?season=1415`, März 2015. – letzter Zugriff am 19.03.2015

[ZRH04] ZHU, Yali ; RUNDENSTEINER, Elke A. ; HEINEMAN, George T.: Dynamic Plan Migration for Continuous Queries over Data Streams. In: *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA : ACM, 2004 (SIGMOD '04). – ISBN 1–58113–859–8, S. 431–442