[illegible]

Fourth International Conference on

User Modeling

Proceedings of the Conference

15-19 August 1994
Tara Hyannis Hotel
Hyannis, Massachusetts, USA

Fourth International Conference on

User Modeling

Proceedings of the Conference

15–19 August 1994

Tara Hyannis Hotel

Hyannis, Massachusetts, USA

Published by The MITRE Corporation

Order copies of proceedings from: Bradley A. Goodman, The MITRE Corporation, Artificial Intelligence Center, 202 Burlington Road, Bedford, MA 01730-1420, USA

User Modeling for Domains without Explicit Design Theories

Knut Pitschke

University of Oldenburg, Dept. of Computer Science
P.O.Box 2503
D 26111 Oldenburg, Germany
Knut.Pitschke@informatik.uni-oldenburg.de

Abstract[1]

This paper presents an approach to user modeling and planning in systems where no explicit domain knowledge is available. The system requires an oracle (for instance a verification tool) to a posteriori classify the users´ behaviour. Originally designed for Intelligent Tutoring Systems, it seems to be applicable to other domains as well. It consists of a learning component that builds up search spaces out of the classified users´ actions and behaviours, a prediction component that uses probabilities to forecast the users´ goals and means, and a calibration component that controls the prediction mechanism and adapts it to the current user.

1. Introduction

Intelligent Tutoring Systems (ITS) are knowledge based systems that shall support a learner by help, explanations, exercises and recommendations in a competent and careful way. To do this, they must be capable of adapting to the individual learning style of each student. This requires a model of the user´s current knowledge state, which is called a student model in this context (Kass 1991; McCalla, & Greer 1992). Using a student model to realize the student´s needs for offering appropriate help can be compared to plan recognition: Observing a sequence of actions and inferring the intended goal. Presenting adequate help is similar to plan generation: Offering action steps, that lead to the student´s intended goal. Using approaches from plan recognition in the context of ITS (Bauer et. al. 1991) requires knowledge about the domain being taught and being planned in. In this paper, an approach is presented that realizes the ideas of planning in the domain of ITSs which lack an explicit design theory about the domain to be taught, but make use of an oracle (in the sense of (Valiant 1984)) that classifies the users´ solutions to exercises.

2. User modeling and planning

Plan recognition means inferring possible (planning) goals from observed sequences of actions. Planning means transforming a current state (of problem solving) to or in the direction of an known goal state by using possible operators in a pre known state space (Allen et. al 1991). This scenario is very similar when using student models in ITSs as a basis for offering the right help information at the right time. Students face a task they shall solve in order to get familiar with the domain the task is taken from. Normally there is not a single solution but the number of correct solutions is manifold, as well as the different ways to achieve them. Locating the student´s state in that search space is the task of the student model. By doing so the student model provides information to generate help that supports the student to reach his goal (Möbus, Pitschke, & Schröder 1992).

3. Domain knowledge in user modeling and planning

The student model usually describes and represents student knowledge in relation to expert knowledge (Kass 1991). The user knowledge is inferred from the observation of his behaviour. Interpreting his actions with respect to the expert knowledge is a prerequisite for locating his knowledge state in the knowledge space. This interpretation is usually done by a diagnosis component that uses a domain theory to try to reconstruct the student´s solution proposal for finding out whether it is correct or not, and to construct a continuation of his problem solving proposal towards a solution (Möbus, Schröder, & Thole 1992). Someone who plans, uses his knowledge to achieve a certain goal. If no knowledge about the planning domain or the subject being taught is available, usually it is not possible to either recognize a plan or the student´s knowledge state, or to generate a plan or a solution proposal for the student. The domain of Petri Nets, the subject being taught in our system Petri-Help (Möbus et. al. 1993), does not provide a formal design theory to construct petri nets out of a given specification. That is why the standard approaches from planning and student modeling are not applicable.

[1]This research was supported by the Stiftung Volkswagenwerk (Az. 210-70631/9-13-14/89).

4. Petri-Help

Petri-Help is a help system that supports users in modeling with condition event petri nets (Pict. 2). A sequence of tasks is presented for which the student shall develop a solution. During the problem solving session, the student can ask the system to judge his solution sketch proposal or to suggest a completion proposal. The judgement of solution sketches implies a formal specification of the task to be solved, and an oracle to classify the sketches with respect to the specification. In the petri net domain it is unusual to present a formal specification of the scenario to be modeled (Reisig 1992). Instead of this nets are presented as solutions and due to their self explaining property everyone understands the problem being tackled. An exception is (Olderog 1991), where petri nets are used as formal semantics for process terms, which are formally derivable from a trace logic specification.
In Petri-Help, we decided to present the tasks to be solved as a set of temporal logic formulae, which describes the properties of the intended solution (see Pict. 1).

Description of Places:

Ws	Waiter is sleeping
Wro	Waiter is ready to accept order
Wrs	Waiter is ready to serve
K	Kitchen got the order
P	Meal gets prepared
R	Meal is ready

Starting Condition: Ws

Temporal Logic Formulae:

$O(Wro \rightarrow \Diamond(Ws \wedge K))$

$O(K \rightarrow \Diamond P)$

$O(P \rightarrow \Diamond R)$

$O(R \wedge Ws \rightarrow \Diamond Wrs)$

$O(R \wedge Wro \rightarrow \Diamond Wrs)$

$O(Wrs \rightarrow \Diamond Ws)$

$O(Ws \rightarrow \Diamond Wro)$

$O(\neg(Ws \wedge Wro))$

$O(\neg(Ws \wedge Wrs))$

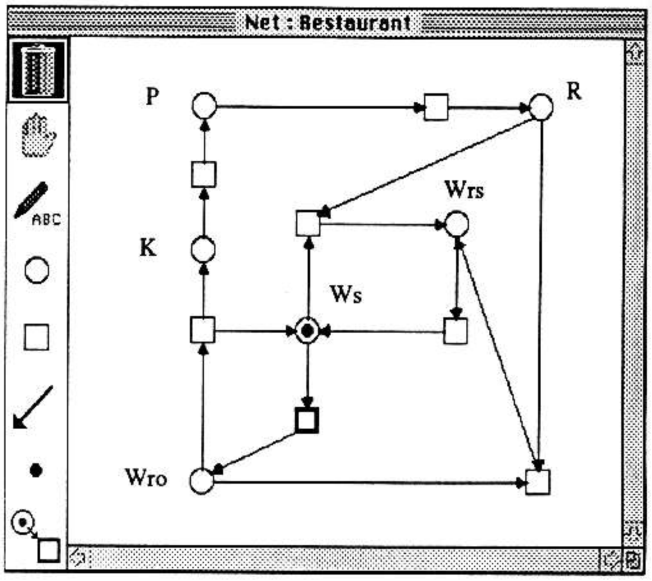
$O(\neg(Wrs \wedge Wro))$

$O(Ws \vee Wrs \vee Wro)$

O means "always", ◊ means "eventually"

Picture 1: Temporal Logic Description of the "Restaurant" Modelling Task

We use a simple propositional logic which is enriched by the temporal predicates "nexttime", "eventually" and "always". Our logic allows branching time and uses step semantics. This way of specification enables us to verify solution proposals or parts of them by model checking (Josko 1990). Model checking is used to test which subset of the specification formulae the current petri net is a model for, or equivalently which part of the specification is fulfilled.



Picture 2: A user's solution proposal for the "Restaurant" Modelling Task

Due to our restriction on condition event petri nets, these models are always finite and often cyclic. Unfortunately, the sequence of nets the student runs through while developing a solution, is nonmonotonous with respect to the specification formulae being fulfilled in the nets. That means, a formula once having been proven, may be falsified in a later state of the problem solving process, although this state is necessary to obtain a correct solution. That is why it is not possible to glue together net fragments fulfilling single formulae to achieve a net that fulfills the conjunction of these formulae. In spite of this nonmonotonous property, the number of formulae fulfilled increases during problem solving. At the beginning, the empty net fulfills no formula, at the end, the correct solution is a model for the whole set of specification formulae. So there must be nets detectable during problem solving, where some formula has been fulfilled additionally. We call each net on the way to a solution a state, and each such state, where the set of fulfilled formulae increases, a safe state. Safe states are defined inductively: The empty net is a safe state. Every net that is a model for a superset of those formulae being fulfilled in the last safe state, is called safe. These safe states are used by a learning component to acquire design rules which are used to generate help proposals for the user. The system was used by a course of graduate students. The verification of formulae was widely used and accepted. The help proposals were criticized to be not fine grained enough for the several impasse situations. This problem arises, because no user model is yet integrated in the system that could guide the generation of help information towards offering appropriate help.

5. An adequate approach to user modeling

As mentioned above, in our domain (petri nets) it is not possible to build up a state space a priori. Instead of this, the search space is learnt by the system during problem solving sessions with all users. In this state space, later users are identified by their observable behaviour, and the most probable next action or goal can be predicted with respect to their history (Pitschke 1993). Thus, we don´t have a real user model, but a usage model (Grunst, Oppermann, & Thomas 1993), and the identification of a single user is done by probabilities about his behaviour. An approach based on probabilities as well is described by (Becker 1993) for the acquisition of stereotypes.
In the following section, it is described how the state spaces are acquired by observing the users´ behaviour. It is shown, in which way a new student is identified within these problem spaces and how the prediction mechanism works in detail. After that, it is described how the prediction control adapts to the current user by comparing predictions with the observed actions. At the end, it is described how to use this mechanism to offer appropriate help and to validate existing design heuristics about users´ behaviour.

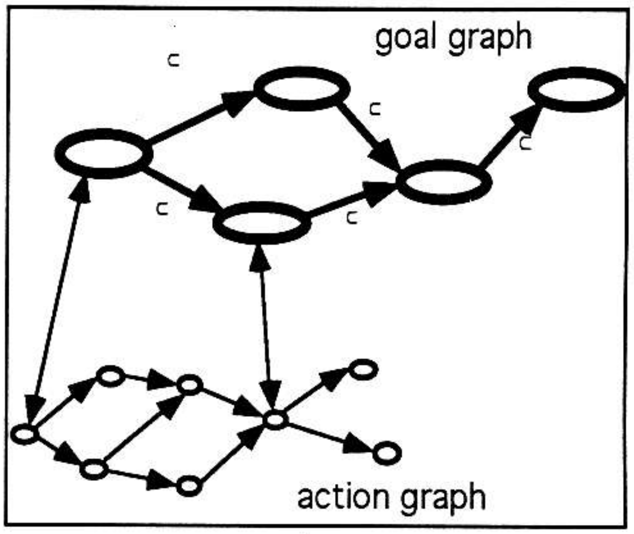
5.1 Representing users´ goals and actions

The information we can get about a user is restricted to the dialog between him and the system, and its interpretation according to the current context. The system acquires this information at two levels: A goal level and an action level. The goal for every user is to solve each task presented by the system, which consists of a set of temporal logic formulae. A sequence of possible subgoals is every sequence of increasing subsets of the set of formulae specifying the task. So every time the user tests a subset of formulae to be fulfilled in his current net, we assume his (sub-)goal was to construct a petri net that is a model of that very set. Out of these subgoals, a goal graph is constructed, which is enhanced by every new observation of a new user. The relations that hold between these states are, first, the subset relation. Every successor node represents a superset of the formulae of its predecessor. Secondly, an arc between two states indicates, that there was a user who chose these two goals as successive subgoals.
In parallel, all the users´ actions they perform to reach a goal are noticed. These actions form the action graph. Its nodes are petri nets which the user developed during his problem solving session. Every editing action results in a new node. The only relation between the nodes of the action graph is that a student, once having developed a net, represented in a node, after his next observable net manipulating action, reached the successor state.

Goal graph and action graph (Pict. 3) are connected by links. Every goal node is connected to all the nets in the action graph that were tested by some user to be a model of these formulae represented in the corresponding goal node. Every node, in the goal graph as well as in the action graph, which has at least one successor, is enriched by probabilities. These probabilities represent the relative frequency of decisions for a possible successor node on condition that a certain history has been observed. The conditional probabilities are represented in every node as:

p(j I i, h) where: i denotes the current node j denotes the successor node and h is the history.

The histories consist of all user´s decisions in the two graphs and also include the time, they needed to traverse the graphs on their way to a solution. These probabilities, together with a certain criterion vector, individual to the current user, will be the basis for the prediction process.



Picture 3: The two graphs to represent the behaviour of several users

5.2 Predicting the behaviour

Predicting the user´s behaviour means finding the most likely continuation with respect to his former behaviour and to the observed behaviours of all former users.
For predicting the user´s next action or goal in a certain state the goal graph and the action graph are searched for a similar behaviour of a former user. For this purpose it is checked whether a conditional probability is stored in the actual node having the current user´s history as its condition. If such conditional probabilities exist the successor node is chosen which has the highest probability. If the user´s history is not found as a condition in a conditional probability it has to be generalized in order to match on an other history leading to the current node. The criteria for

generalization are stored in a criterion vector specific to the current user. The criteria are used to filter the information stored in the two graphs to retrieve a history that is possibly similar to that of the current user. This filter operation is realized in two steps.
First of all, a heuristic generalizes the current user´s history according to the criterion vector until it matches on at least one history.
Secondly, a conflict resolution strategy finds out which of the possible successor nodes is most likely. After the prediction process, the prognosis is compared to the user´s real behaviour, and as a result, the criterion vector and the heuristic strategy are calibrated.

5.3 The criterion vector

To show, how the prediction process works, we have to take a look at the criterion vector and the matching process. A history, leading to a specific node, consists at the moment of a sequence of nodes, having been traversed to reach the current node, and the amounts of time needed to step from one node to the next. It can be easily enriched by further information, deducible from the dialog, like user´s mistakes or demands for help.
The components of the criterion vector show how strictly the user´s history should be used in order to find a corresponding history in the graphs. The criterion vector is generated after the user´s first two goals or actions were observed. So, in the first node it is known which history the user processed up to this node, and the way he chose to go on. According to the user´s step from the first to the second node an optimal history up to the first node is generated that would have led to the prediction of that very step. After that a least common generalization is found out of this optimal history and the user´s actual behaviour. The criterion vector is set up with the parameters that would generalize the user´s history (up to the first node) to the least general generalization mentioned above.
Consider, for instance, the user´s way traversing the nodes a, b, c and d to reach node e (no difference whether in the goal, or in the action graph). One criterion of the user´s vector could state to just consider the last three nodes of his way, in that example c, d and e. So, in the current node e, this criterion would generalize the user´s way to *cde, where * matches to any sequence of nodes, starting with the empty net/goal and leading to node c. An extended example can be found in (Jordan 1994).

5.4 The heuristic

Imagine there is no matching history after applying the criterion vector to the user´s current history. In this case, the criteria of the criterion vector have to be weakened, to get a more general description. For this purpose, the heuristic is designed as a family of functions, one for each component of the criterion vector. Each function weakens its specific component according to the frequency of success in matching of that criterion and according to a factor, that indicates the step size by which the criterion is weakened. This factor is also subject to the process of adapting the heuristic.
The conflict resolution strategy is involved if the matching process ends with more than one history fitting the current criterion vector. So one history has to be selected that serves as the condition for the conditional probabilities for choosing a specific node being in the current node.

5.5 Adapting the criterion vector and the heuristic strategy

The adaptive aspect of the prediction mechanism is initiated by predicting the behaviour in every state the user reaches, although he didn´t ask for help. After his next step, the result of the prediction is compared to the observed action, and the criterion vector and the heuristic are adapted. Adapting the criterion vector means finding a vector which generalizes by the matching process to a history, that delivers the very node that has been observed as a prediction. In fact we need a criterion vector for a least general generalization in order to predict correctly.
Adapting the heuristic is based on the difference vector between the old and the adapted criterion vector. The factor for step size in each component´s heuristic function is changed to a value that makes the heuristic return the adapted criterion vector (see above) in one step, when getting the old criterion vector as its input.

6. Help generation

Students ask for help if they are in an impasse situation (Möbus, Pitschke, & Schröder 1992). That means they don´t know how to proceed towards a correct solution. Help information should be appropriate to the student´s current situation, should offer as little information as possible, so that the problem solver is urged to leave the impasse by his own activities. Furthermore information should cause only little surprise and not a new impasse. Empirical studies with about 40 students indicated that using Petri Help, impasse situations usually don´t arise at the beginning of a problem solving session. So the prediction mechanism as described above could adapt itself to the current user´s problem solving strategy. Help information containing the prediction of the student´s next step satisfies the criteria stated above because it is the most likely continuation of his previous work. Compared to the help facility in Petri-Help the information offered is much more fine grained because not only the actions leading to safe states are offered as help.

7. Validating design heuristics

As mentioned above, in the petri net domain we have no explicit design theory. Nevertheless, watching human

modelers indicates that they use several design heuristics to transform the temporal logic problem description to a petri net (Möbus et. al 1993). They take a single formula and create a net fragment to fulfill it, without taking care of the nonmonotonous property. Of course, only model checking can prove the correctness of the application of these heuristics. Validating a design heuristic means showing that it has been applied frequently and successfully.
The goal and the action graph provide all the information for finding the correspondence between a heuristic´s precondition and its conclusion. The difference of two successive goals in the goal graph includes the heuristic´s precondition (a temporal logic formula). The difference between the petri nets in the nodes of the action graph that correspond to the goal nodes includes the conclusion (the net fragment being a model for the formula).
A first version of the validation is already implemented to work on the database learnt by Petri-Help (see above). It indicates a great plausibility of the heuristics, used by most human problem solvers.

8. Conclusion

It was shown, how a simple adaptive control mechanism, based on a probabilistic approach, can be used to predict student´s behaviour while interacting with an Intelligent Tutoring System. It was pointed out, how close the relation is between plan recognition and user modeling on one hand, and planning and predicting behaviour on the other. The whole system is not based on explicit domain and background knowledge, but on a learning and a verification component. The idea, originally developed to enhance Petri-Help, could be adapted to any planning domain, fulfilling the above mentioned prerequisites. Petri-Help is fully implemented on Macintosh™ computers using MacProlog™. It is widely used by graduate students. The validation component is implemented as a prototype. The rest of the system is currently under implementation by students of an advanced practical training course in Intelligent Tutoring Systems.

9. Acknowledgement

The author would like to thank Prof. Dr. C. Möbus and Dr. O. Schröder for their support of this work and many productive discussions, and J. Folckers for implementing Petri-Help. Also thanks to the reviewers for their valuable and encouraging comments.

10. References

Allen, J., Kautz, H., Pelavin, R., & Tenenberg, J. (1991). *Reasoning about Plans.* San Mateo, Ca.: Morgan Kaufmann.

Bauer, M., Biundo, S., Dengler, D., Hecking, M., Koehler, J., & Merziger, G., (1991). Integrated Plan Generation and Recognition - A Logic Based Approach, DFKI Research Report RR-91-26, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Kaiserslautern/ Saarbrücken

Becker, A., (1993). Automatically Acquired Knowledge Levels as a Basis for User Modeling, in: (Kobsa, & Pohl 1993)

Grunst, G., Oppermann, R., & Thomas, C. (1993): User Modeling in Context-Sensitive Help and Adaptive System Design, in: (Kobsa, & Pohl 1993)

Jordan, O., (1994): A Method for Predicting Behaviour, Study Thesis, University of Oldenburg (in german)

Josko, B. (1990). Verifying the correctness of AADL modules using model checking. In J. W. de Bakker, W. P. de Roever, G. Rozenberg (eds.): Proceedings REX-Workshop on stepwise refinement of distributed systems: models, formalisms, correctness. Berlin: Springer, LNCS 430, 386-400.

Kass, R. (1991): Student Modeling in Intelligent Tutoring Systems, in: (Kobsa, & Wahlster 1991)

Kobsa, A. , Pohl, W. (eds.) (1993): Workshop 'Adaptivity and User Modeling in interactive Software Systems', held on the German Conference on Artificial Intelligence, Berlin, 13.-15.9.93. WIS-Memo 7, September 1993, AG Knowledge-Based Information Systems, University of Konstanz, (in german).

Kobsa, A., Wahlster, W. (eds.) (1991): *User Models in Dialog Systems*, Berlin: Springer.

McCalla, G., Greer, J., (eds.) (1992): *Journal of Artificial Intelligence in Education,* Special Issue on Student Modelling, 3(4) 1992

Möbus, C., Pitschke, K., & Schröder, O. (1992) . Towards the theory-guided design of help systems for programming and modelling tasks. In C. Frasson, G. Gauthier & G. I. McCalla (eds.): Intelligent tutoring systems, Proceedings ITS 92. Berlin: Springer, LNCS 608, 294-301.

Möbus, C., Pitschke, K., Schröder, O., Folckers, J., & Göhler, H. (1993): PETRI-HELP - Intelligent Support for Petri Net Modellers, Internal Report, Department of Computational Science, University of Oldenburg

Möbus, C., Schröder, O., & Thole, H.-J. (1992): A Model of the Acquisition and Improvement of Domain Knowledge for Functional Programming, in: (McCalla, & Greer 1992)

Olderog, E.-R. (1991). *Nets, terms, and formulas.* Cambridge: Cambridge University Press.

Pitschke, K. (1993). Goal Identification and Planning in Intelligent Tutoring Systems; In: (Kobsa, & Pohl 93)

Reisig, W. (1992). *A primer in Petri net design*, Berlin: Springer, 1992.

Valiant, L. (1984). A Theory of the Learnable. *Communications of the ACM*, 27, 1134-1142