Combining Finite Automata, Parallel Programs and SDL using Petri Nets

Bernd Grahlmann

Institut für Informatik,
Universität Hildesheim,
Marienburger Platz 22,
D-31141 Hildesheim,
bernd@informatik.uni-hildesheim.de,
www.informatik.uni-hildesheim.de/∼bernd

Abstract. This paper introduces a method to combine finite automata, parallel programs and SDL (Specification and Description Language) specifications. We base our approach on M-nets exploiting the rich set of composition operators available in this algebra of high-level Petri nets. In order to be able to combine different modelling techniques, we rely on compatible interfaces. Therefore,

- we extend an existing semantics, namely the M-net semantics for the parallel programming language $B(PN)^2$; and
- we present an M-net semantics for finite automata.

Considering the hybrid modelling of an ARQ (Automatic Repeat reQuest) protocol, we show how the different formalisms fit together as well as the resulting verification possibilities. As a side-effect we describe on-going development of the PEP tool (Programming Environment based on Petri Nets). As a consequence of our approach we are introducing a hierarchical 'programming with nets' method which is currently implemented in the high-level Petri net editor of the tool.

Keywords: $B(PN)^2$, Finite automata, Hybrid system design, M-nets, Parallel programs, PEP, Petri nets, SDL, Verification.

1 Introduction

So far, the PEP* tool (Programming Environment based on Petri Nets [7, 18]) supports a variety of (high-level) modelling techniques:

- $B(PN)^2$ (Basic Petri Net Programming Notation) a parallel programming language [8, 5, 13],
- M-nets (Modular multilabelled nets) an algebra of high-level Petri nets [6],
- PFA (Parallel Finite Automata) [16] a collection of FA (Finite Automata).

* The PEP project is financed by the DFG (German Research Foundation). This work has been partially supported by the HCM Cooperation Network EXPRESS (Expressiveness of Languages for Concurrency)

Moreover, we have given a compositional M-net semantics to SDL (Specification Description Language) [9] covering dynamic process creation and termination as well as procedures [14, 15]. However, a hybrid system design approach which combines one or more of these techniques has not been presented so far. Addressing this problem and proposing a solution is the main purpose of this paper.

In contrast to other (even Petri net) tools, the PEP tool has been designed in such a way that Petri nets are the central method into which different kind of systems are translated in order to use Petri net theory for the simulation, the analysis and the verification. Therefore, it is straightforward to realise also the combination of different formalisms using (preferably high-level) Petri nets.

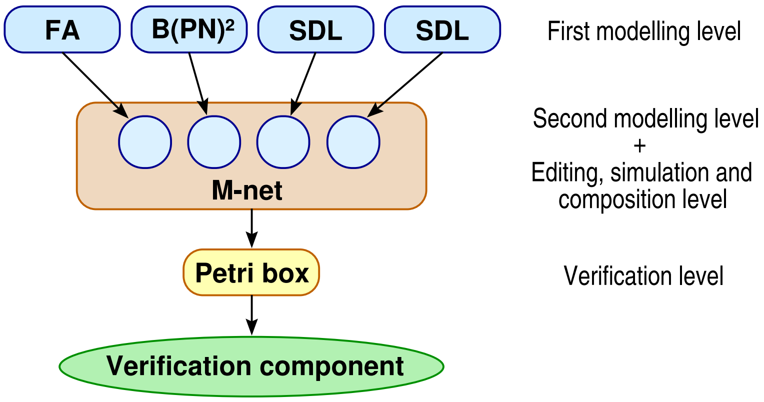

Fig. 1. Abstract presentation of the hybrid modelling.

Fig. 1 shows the abstract idea of our approach. We distinguish two different modelling levels.

1. On the first level, each part (possibly more than one) of the whole system is designed individually using one of the supported modelling formalisms ($B(PN)^2$, FA or SDL).
2. The second level has two purposes:
   (a) A hierarchical description of the system specifies how the individual parts are combined and giving its general functionality.
   (b) It serves as an additional editing and simulation level offering interactive M-net editing, simulation and composition.

The bottom part of the figure shows that the resulting M-nets are unfolded into a special class of low-level Petri nets called *Petri boxes* [4] in order to apply the different verification techniques included in the PEP tool. Currently, PEP includes partial order based model checking [12, 11, 19] and algorithms based on linear programming [23] as well as interfaces to other verification packages [18] such as INA [26], SMV [10, 27] and SPIN [20] providing reduction algorithms based on BDDs, on the stubborn set or sleep set method, and on symmetries.

After choosing a general approach, the next important task is to consider the interfaces in more detail. The interaction facilities of the different formalisms have to be analysed to provide compatible interfaces. This should include:

1. accesses to variables,
2. procedure invocations,
3. process creations,
4. synchronous as well as asynchronous communication via B(PN)^2-like channels, and
5. SDL-like send and receive operations.

Assuming that our approach is successful there are a lot of benefits. Users then have the possibility to model each part of a parallel system using the formalism of his/her choice. This is particularly interesting for teams performing distributed or cooperative system design. Apart from personal preferences, gaining expressiveness is an interesting argument. For instance, SDL provides different communication mechanisms from B(PN)^2. Obviously, it eases the modelling if all of them may be used. Finally, our approach supports reuse of existing components.

This paper is structured as follows. First, we briefly describe the most relevant aspects of M-nets in section 2. In section 3 we will introduce our running example, a hybrid modelling of an ARQ (Automatic Repeat reQuest) protocol. Thereby, we will give an intuition of the three different modelling formalisms (B(PN)^2, FA and SDL). In section 4 we will exploit the interfaces in more detail presenting the different extensions of the semantics. Section 5 illustrates some of the resulting verification possibilities. Before we conclude in section 7 we will present new features of the PEP tool which are closely related to the presented approach in section 6. Finally, a list of references is given. Some of the papers are available at http://www.informatik.uni-hildesheim.de/~pep.

2 M-nets

The class of M-nets forms an algebra of high-level Petri nets. It was introduced in [6] as an abstract and flexible meta-language for the definition of the semantics of concurrent programming languages. The most distinguishing feature of the M-nets is given by the rich set of composition operators they provide. These allow – as composition operators in process algebras usually do – the compositional construction of complex nets from simple ones, thereby satisfying various algebraic properties.

Two kinds of inscriptions are distinguished. Annotations are responsible for the unfolding into low-level Petri nets while labels are used for the composition operations (parallel and sequential composition, choice and iteration as well as synchronisation and restriction). For our purpose it is interesting that

– Places of M-nets are annotated by a *type*, describing the set of allowed tokens. A type may contain natural numbers, the Boolean values, the usual token $\bullet$, and the special token $\dagger$, or tuples of these. The label distinguishes entry, internal and exit places.

- Transitions are annotated with occurrence conditions and labelled with (multi) sets of action terms. We will give an example how action terms are used in the synchronisation below.
- For our purpose it is sufficient to annotate arcs with sets of expressions.

We will use Fig. 2 to provide the necessary intuition. Let us first consider the subnet consisting of two internal places Pd and Pe, one transition Tc and two arcs. The *transition rule* for M-nets is explained informally on the example: If Pd is marked with a pair $(3, 2)$, the variables in the arc inscriptions (pid and πid) can only be bound to 3 and 2, respectively. Thus, an occurrence of Tc carries the pair $(3, 2)$ to Pe. However, Tc can occur in infinitely many modes, because the action term contains variables ($'X$ and X') which are not sufficiently bound. E.g., $X' = 2 \wedge\ 'X = 3$ is a possible binding.

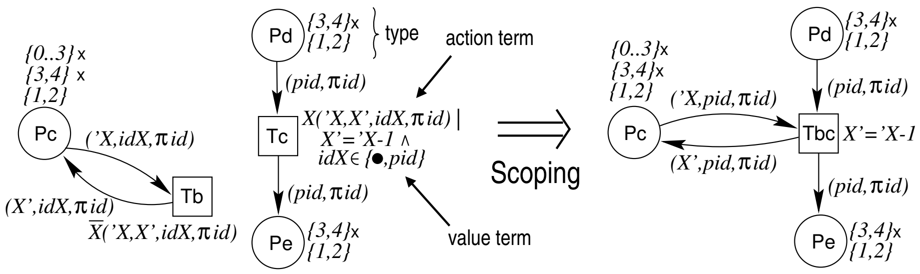

Fig. 2. Scoping example: $N_1 \| N_2$ and $[\ \{X\} :\ (N_1 \| N_2)\ ]$.

Fig. 2 as a whole depicts how the *scoping* mechanism (first synchronisation and then restriction) may restrict these bindings considering a typical example, namely the access to a local variable within a procedure. Tb is the access transition within the variable net and Tc corresponds to the access of the variable within the procedure. The renaming of the variable idX to pid (according to the $idX \in \{\bullet, pid\}$ part of the value term) ensures that the correct instance of X is accessed. Note that the free variables are bound during the scoping.

3 Hybrid Modelling of the ARQ Example

In this section we will introduce our running example. We will use the description of the hybrid modelling of a simple ARQ communication protocol with alternating acknowledgement to give an overview of the main characteristics of the supported modelling formalisms.

The inner part of Fig 3 shows that the whole ARQ system comprises two different processes (Sender and Receiver) communicating via two SDL signal routes (a and d). Furthermore, the initial number of instances as well as the maximal number are specified (e.g., Sender(1,2)) and the type of the signals is given (e.g., signal ack ({0,1})).

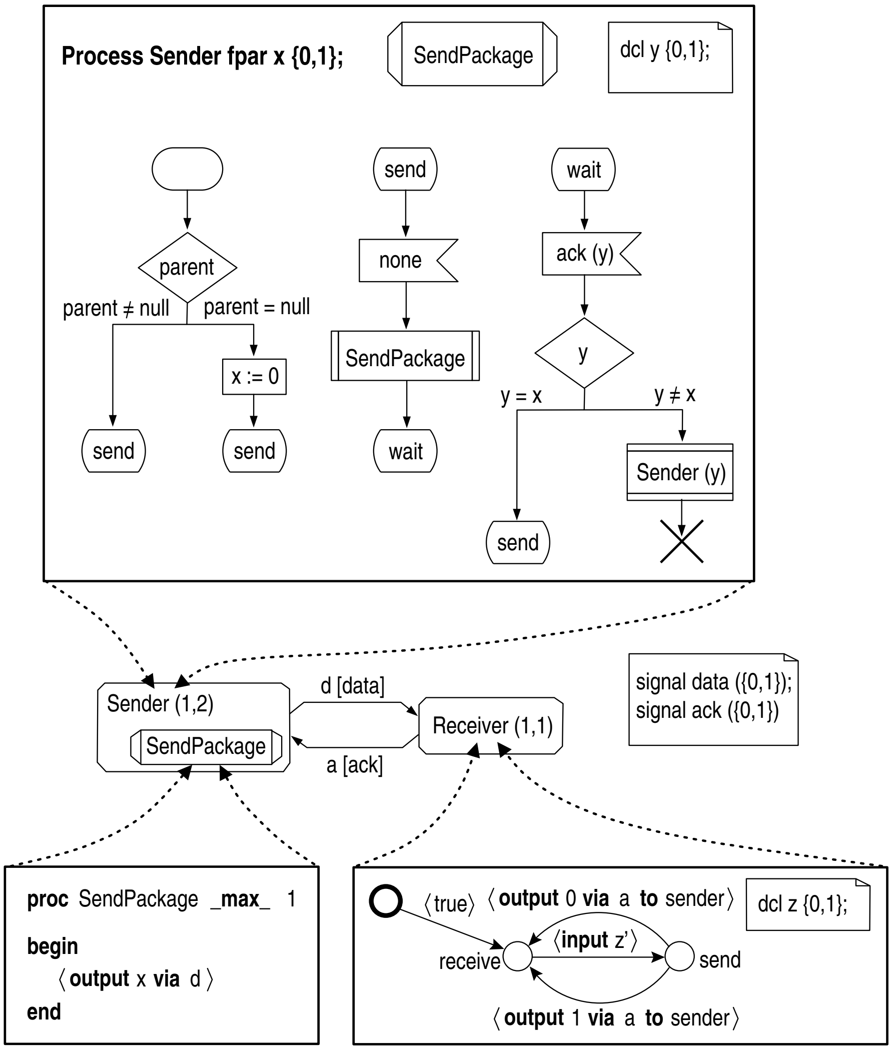

Fig. 3. Hybrid modelling of the ARQ system.

The rest of the figure explains how the three components (one for each process and one for the procedure SendPackage belonging to the Sender process) are modelled using the editors corresponding to the different formalisms.

The purpose of the ARQ system is to transmit an alternating sequence of 0 and 1 (modelled by a signal type data with parameter type $\{0, 1\}$). Each Sender instance is responsible for the correct transmission of exactly one data package with the label x (which is the formal parameter) via the signal route d to the Receiver process. It is intended to introduce failure by letting the Receiver process non-deterministically return random acknowledgements via the signal route a to sender (the sender of the last received data package).

3.1 SDL Part

The Sender process is modelled in the graphical representation of SDL. SDL is a quasi-standard language for the specification of distributed systems, especially in the area of telecommunication. An SDL specification describes a *system* consisting of *processes* which are extended communicating finite state machines.

In SDL variables and procedures are declared within the scope of a process, but they are local to process instances – i.e., different instances do not interfere. In addition to the instances which are created automatically at system start, process instances may be created by any other instance of any process type, but can be terminated only by itself. Each process instance is equipped with an input queue. Signals which are transmitted from an instance via signal routes (without delay) or channels (with delay) to another instance are put in the input queue of the receiver. In contrast to other languages there is always one receiver. Neither broadcast nor multiway communication is available. The receiver may be specified by implicit variables whose values are changed automatically

- self: the instance itself,
- sender: the sender instance of the most recently consumed signal,
- parent: the instance that created this one, and
- offspring: the most recently created instance;

or by the signal route (if this is unique). Furthermore, synchronous communication is not supported because the input queue already introduces a potential delay.

Starting from the initial state () the behaviour of the Sender process depends on the implicit parent variable (parent). The initial instance (parent=null) sets the value of x to 0 before entering the state send. The Sender instance continues to invoke the procedure SendPackage (SendPackage) upon receipt (ack (y)) of the corresponding acknowledgement (y≠x). Then it creates (Sender (y)) the next Sender instance (passing the next label as parameter) and terminates () afterwards.

3.2 B(PN)^2 Part

The SendPackage procedure is modelled in the parallel programming language B(PN)^2 [8]. Atomic actions (enclosed in action brackets $\langle\rangle$) offer much more expressiveness than the corresponding SDL counterparts. For instance, the action $\langle C1? = {}'x \wedge C2! \geq x' - y' \rangle$ is only executable if the current value of the variable x can be read from the channel $C1$. Furthermore, an occurrence of the action has the effect that the values of the variables x and y are set (non-deterministically) in such a way that the difference is less or equal to a value which can be written (and actually is written upon occurrence of the action) to a second channel (or stack) $C2$. This can not be expressed in SDL and it is very difficult to express it in, for instance, C [22], which is the main language for the implementation of verification algorithms.

As mentioned above, the communication mechanism is also very different. The given action may, for instance, synchronise with an action a_1 of a process Π_1 (via C1) and with b_2 of a process Π_2 (via C2). But, as well also with an action c_1 of the process Π_1 (via C1) and with f_3 of another process Π_3 (via C2).

In [13] the original M-net semantics [5] has been extended by (also recursive) procedures covering severall kind of parameters (value, value-result and reference). In our example, the procedure is declared without any parameter. The maximal number of concurrent instances is restricted to '1' (**_max_** 1). The procedure only performs an SDL-like send operation (⟨ **output** x **via** d ⟩) of the value of the (w.r.t the procedure) global variable x via d (which may be a signal route or a channel). The details of this SDL-like communication mechanism will be explained in section 4.2.

3.3 FA Part

The Receiver process is modelled as an FA [21]. In [16] a special class of $B(PN)^2$ specific FA has been introduced. These FA may consist of

1. a start node (representing the initial state of a process),
2. a set of local nodes,
3. a set of exit nodes (representing that the process has terminated),
4. a set of edges between these nodes, and
5. a labelling function that annotates each edge with a $B(PN)^2$ action.

So far, the PEP tool includes a structure preserving compiler which translates a collection of FA into a $B(PN)^2$ program. This offers an additional nice graphical interface, but it implies a restriction, because not every control structure of an FA may be compiled into a control structure of a $B(PN)^2$ program. In [24] an extension of $B(PN)^2$ (mainly by $GOTO$s) has been proposed which overcomes this restriction. Its implementation would extend the existing possibilities to combine $B(PN)^2$ with FA. In this paper we propose to translate FA directly into M-nets which allows us to use arbitrary control structures without extending other formalisms and without introducing any $B(PN)^2$ specific overhead. Thus, in addition to the nice graphical appearance of FA we may benefit from the extended possibilities to specify control structures.

In our example the FA is very simple. It only consists of one start node, two internal nodes, four arcs and an SDL-like variable declaration (dcl z ({0,1})). Starting from the initial state the process first enters the state receive. Then, the process may infinitely often receive (⟨ **input** z' ⟩) a signal (writing the value to the variable z) and afterwards send (e.g., ⟨ **output** 0 **via** a **to** sender ⟩) randomly either a '1' or a '0' via a (which may be from the point of view of the FA a signal route or a channel) to the sender of the last received signal. This obviously implies that the FA has to be translated in such a way that the handling of the implicit variables (such as sender) as well as of the input queue is done in the same way as in SDL.

4 Extensions of the Semantics

In this section we will first analyse the potential interfaces between the different formalisms in detail. Afterwards, we will give the main ideas for successful extensions. In particular, we will describe different ways to translate an FA or a $B(PN)^2$ program into an M-net and we will extend the language $B(PN)^2$ (as well as its semantics) by the introduction of a new communication mechanism.

It is straightforward to compare the existing semantics

1. the original M-net semantics for $B(PN)^2$ without procedures [5],
2. the M-net semantics for $B(PN)^2$ with procedures [13], and
3. the M-net semantics for SDL covering dynamic creation and termination of processes as well as procedures [14, 15]

in order to analyse the problems related to interfaces between different formalisms. The most striking difference is the kind of control flow tokens which directly implies differences concerning the types of the places as well as the number of parameters (arity) of the action terms.

In the first semantics it is sufficient to use black tokens ($\bullet$). As a consequence, nearly all places have the type $\{\bullet\}$. Only those places holding the value of a variable have the corresponding type of that variable (e.g., *set1*) as a type. To be precise, this is an abbreviation for a type $set1 \times \{\bullet\}$. Furthermore, action terms have small arities. For instance, the access to a variable x is performed using an action term *x('x,x')* containing only two parameters, namely for the pre- and the post-value to the variable x.

In the second semantics it is necessary to distinguish different instances of a procedure. As a consequence, everything is extended in order to handle procedure instance identifiers (*pid*s):

- *pid*s (which may be bound to $\bullet$ in the global parts) rather than black tokens are passed in the control flow.
- The types of places have one component for the *pid*s which may either be $\{\bullet\}$ for global parts or a set of *pid*s (*pid_set*) for parts belonging to a procedure.
- Every action term is extended by one parameter for a *pid*. For instance, *x('x,x',idx)* is involved in the access to a variable x.

In the third semantics, an additional extension enables the handling of process instance identifiers (πids):

- Tuples (pid, πid) are passed in the control flow. Once more, each of them (pid and πid) may be bound to $\bullet$ in the global parts (outside of a procedure or outside of a process).
- The types of places are extended by one component for the πids (πid_set).

– Every action term is extended by one parameter for a πid. For instance, *x('x,x',idx,πid)* is involved in the access to a variable x.

In summary, this means that:

– As long as we only want to combine FA and $B(PN)^2$ programs without procedures we can choose the first approach.
– If $B(PN)^2$ programs with procedures are involved, we have to translate each component in such a way that it is (at least) compatible with the second approach.
– As soon as SDL is involved we have to switch to the third approach.

4.1 Translating FA

Intuitively, the translation (which is similar to the construction of the semantics of a $B(PN)^2$ program [13]) of an FA into an M-net involves several steps:

1. The FA itself (without any local variable declarations) is compiled into an M-net. This translation is parameterised with the chosen approach as well as with (depending on the chosen approach) a *pid_set* and a πid_set.
2. All the local declarations are translated (in the same parameterised way) into special variable nets. At the same time initialisation and termination parts are added to the net for the FA itself.
3. The parallel composition of the net for the FA and the variable nets is scoped w.r.t. initialisation, termination and access actions.
4. Depending on the chosen approach the result is integrated into
   (a) the control flow of another net,
   (b) a $B(PN)^2$ or SDL procedure net, or
   (c) an SDL process net.

Let us now consider the first and the last step in some more detail. In principal, the first step translates:

– each node into a place and
– each arc into a transition which is connected via arcs to the places corresponding to the input and the output node, respectively.

All resulting places have the same type which is determined by the parameters of the translation. The inscription of the transition (as well as the arc inscriptions) are constructed in the same way as described in [5, 13, 14], respectively.

Fig. 4 gives two simple examples. In the left part the second approach with a parameter *pid_set* is chosen. In the right part nearly the same FA is translated using the third approach with parameters *pid_set* and πid_set. We additionally allowed SDL specific node inscriptions specifying the corresponding states which are compiled into an action term (*q(*state1,state2,$\bullet$,π*id)*) dealing with the change of the implicit state variable (q).

The application of the steps two and three results in nets which may be characterised by Fig. 5. Two things are important for the subsequent integration into another surrounding net:

1. all input and output places have the same type (pid_set or $pid\_set \times \pi id\_set$, respectively) and
2. all remaining action terms (in our example only $q($state1,state2,$\bullet$,$\pi id))$ are compatible with the surrounding net.

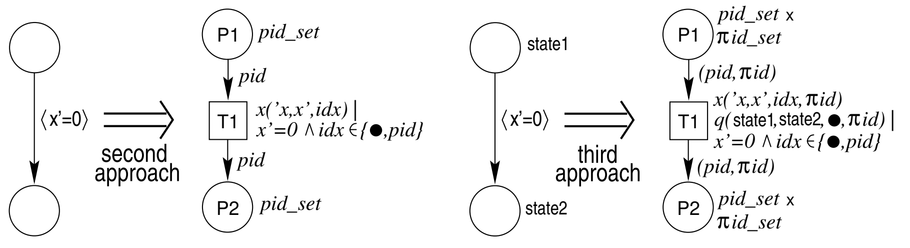

Fig. 4. FA translation example.

Thus, the left net may be inserted in the semantics of a B(PN)^2 program (which has been constructed using the second approach) instead of a block. Therefore, it may, for instance, act as the body of a procedure or just as an arbitrary part of the control flow.

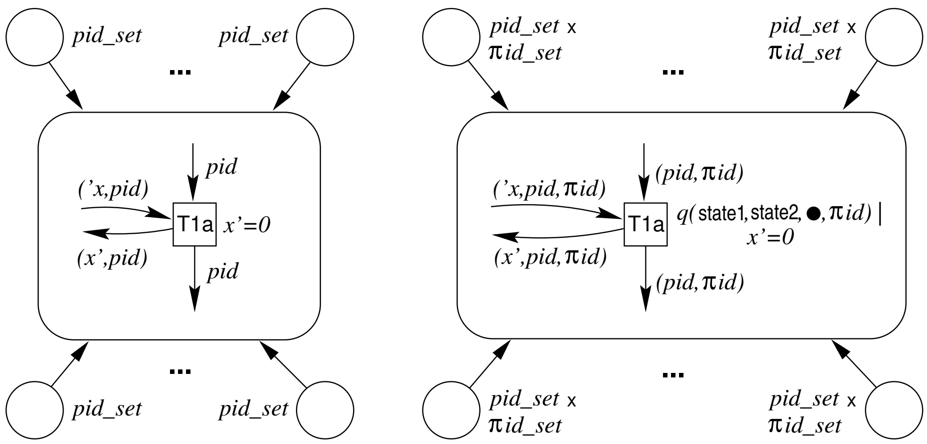

Fig. 5. Simplified representation of the results after the third step.

Due to the fact that an SDL specific FA also performs changes of the implicit state variable (specified by the kind of node and its inscription) the integration of such a net (like the right one) has to obey some further restrictions. Otherwise the surrounding net may, for instance, produce a state state2 while the FA is specified in such a way that it starts in the initial state. This would cause a deadlock. Nevertheless, if the FA starts in the initial state (expressed by an initial node) and ends in a termination state (expressed by an exit node which may be omitted with the effect that an isolated exit place is added to the net), then the resulting net may be inserted as the body of an SDL process net (as in our running example). An abstract representation of the result of this simple transition substitution (cf. [14]) is shown in Fig. 6.

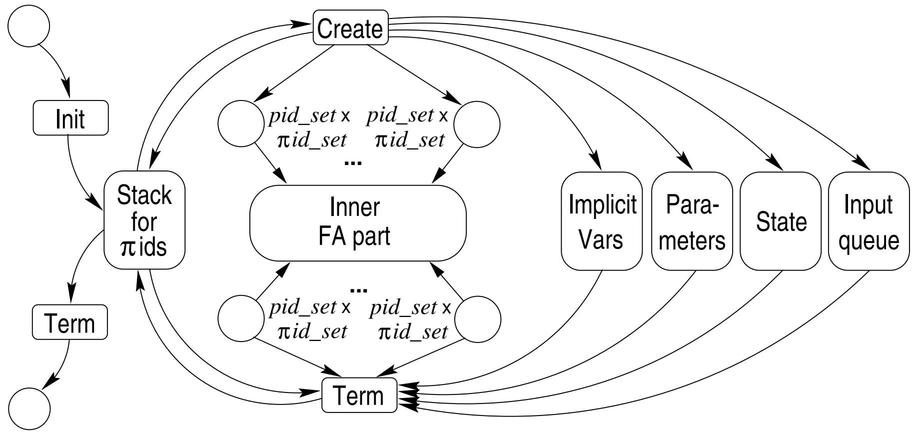

Fig. 6. Abstract view of the FA semantics integration into an SDL process net.

4.2 B(PN)^2 Extensions

So far, we have only dealt with the first (variable accesses) of the five possible interfaces between the different formalisms mentioned in the introduction. And in addition we will only consider the fifth interface (SDL-like send and receive operations) in more detail because the other three do not demand as sophisticated extensions of the already described semantics as the fifth one. They can be handled in the same way as described in [5, 13, 14], respectively.

As described above, the communication mechanisms provided in SDL and in B(PN)^2 are completely different. Regarding the fact that SDL is a widely used and standardised [9] language we have chosen to extend B(PN)^2 (and thus also our FA) by the introduction of SDL-like communication mechanisms rather than extending SDL. To be precise, we basically allow the (standardised) phrase representation of the SDL `input` and `output` construct with some additional possibilities.

The receive action (for instance, $\langle$ **input** z' $\rangle$) reads the first value from the input queue into the variable z. Furthermore, the input may be restricted either by a constant (for instance, $\langle$ **input** 0 $\rangle$) or by an additional side condition (for instance, $\langle$ (**input** z') $\wedge$ (z' $\leq$ 'z) $\rangle$).

There are three variants of the send operation specifying the receiver either by its πid (for instance, ⟨ **output** x **to** sender ⟩), or by a channel or signal route (for instance, ⟨ **output** 0 **via** a ⟩), or by both (for instance, ⟨ **output** 0 **via** a **to** sender ⟩). Furthermore, the output may also be restricted or have side effects (for instance, ⟨ (**output** z' **to** sender) $\wedge$ (z' $\leq$ 'z) ⟩).

The M-net semantics of these new constructs has to take the effects on the implicit variables and the input queue into account which also implies that the resulting M-net has to be combined with an SDL process net. Fig. 7 gives the semantics of the FA part of our running example after the first translation step (the B(PN)^2 action is compiled analogously). Without going too much into the details, we would like to mention that

- T3 corresponds to the receive arc,
- T4 and T5 correspond to the send arcs,
- P4 is the isolated exit place mentioned above,
- *in?(sig,sender',πid)* accesses the input queue (the first parameter is the signal, the second the sender and the third the receiver),
- *a!(0,πid,sender')* accesses the signal route a (with the same parameters).

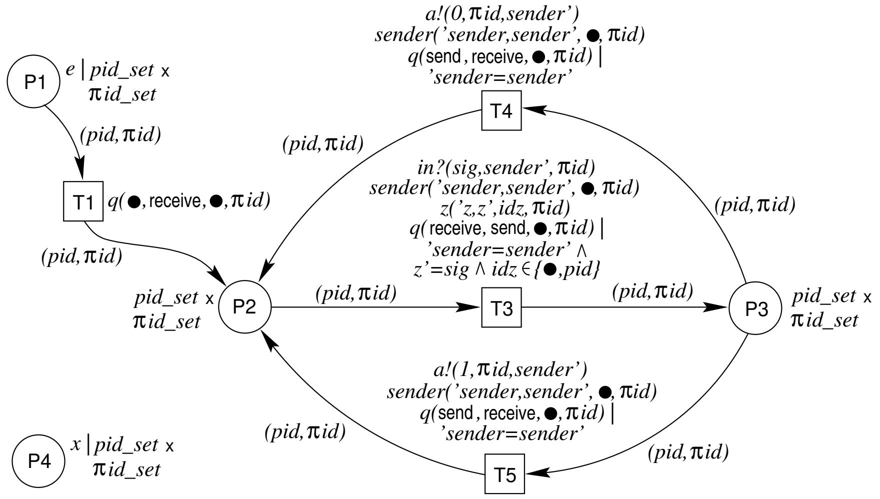

Fig. 7. M-net semantics of the FA part.

5 Verification Results

In [15] we have verified the ARQ protocol applying all kinds of verification techniques which are available in the PEP tool as well as compositional and interactive methods. In contrast to this paper in [15] the whole ARQ system has been modelled in SDL. In order to see the influences of a hybrid modelling using also $B(PN)^2$ and FA parts we have performed the same verifications.

It turned out that we were able to make the same verifications which was our minimal aim. Moreover, the different verifications have been speeded up by 10–30 % which is a nice secondary benefit. It is very likely that this is a result of the reduction of SDL specific overhead. For instance, the semantics of the Receiver process modelled as an FA is smaller than its semantics using SDL. In the rest of this section we will briefly summarise some of the results.

After unfolding the M-net semantics of the whole ARQ system into a low-level net, we have been able to apply a variety of (not only state space based) verification techniques:

1. Partial order based model checking [12, 11, 19],
2. Sleep set and bitstate techniques [20],
3. Reachability graph based model checking (using 'stubborn set' and 'symmetric' reduction) [26],
4. BDD based model checking [10, 27], and
5. Linear programming based analysis [23].

Using these techniques, we verified

- the deadlock-freeness of the resulting net,
- reachability properties (such as reachability of all the SDL states),
- safety properties (such as *none of the input queues ever contains more than one signal*),
- liveness properties (such as *it is always possible to reach a certain state*),
- progress properties (such as *a send must not be eventually acknowledged*).

Note that we have been able to specify these properties without referring directly to the Petri net, by using an extension of the reference scheme introduced in [17].

Although the main purpose was to show that in general (i.e., also for more complex** systems) such a wide variety of verification techniques may be applied using our approach, we want to provide an intuition of the efficiency. Therefore, we mention that most of the tests took less than one second (using the most appropriate verification method) on a relatively slow 40 MHz SUN SPARC 10, that 32 MB main memory had always been sufficient, and that the resulting low-level net had approximately 3.500 states.

6 New Features of the PEP Tool

The integration of the presented hybrid modelling approach into the PEP tool has been planned for a long time. A couple of steps towards a smooth integration have already been implemented.

** For the time being, this means systems with a complex control flow but without complex data types. The development of high-level net verification methods may solve this restriction.

1. The compiler from $B(PN)^2$ into M-nets optionally uses the first or the second approach for the generation of the semantics. Furthermore, a flexible macro concept has been introduced which allows to choose, for instance, variable or procedure nets from a library of parameterised macro nets [1].
2. The scoping mechanism has been realised as an external program allowing the different necessary scoping operations.
3. A compiler from FA into M-nets supporting the different approaches has been implemented.
4. The high-level net editor has been extended in several ways:
   (a) The concept of hierarchies has been extended towards the additional handling of special purpose parameterised macro nets.
   (b) The editor allows different views of the whole net or of its parts.
   (c) A component for the interactive composition of nets has been added.

The main task which still remains is the implementation of a convenient graphical top-level modelling support which allows the user to specify the interplay of the different components. This includes specifications such as '$B(PN)^2$ program P_1 is inserted as a procedure for the SDL process Π_1' as well as the automatic calculation of the parameters for the corresponding translations and the automatic generation of a net for the global control flow dealing, e.g., with the initialisation, termination and creation of initial process instances.

Moreover, the support for the 'programming with nets' approach will be improved. The user will have the possibility to construct a system in a compositional way using parameterised macro nets for $B(PN)^2$ as well as SDL parts. In order to insert an SDL process net, (s)he will only have to specify some parameters (such as the number of initial instances or the names and types of the formal parameters). Afterwards (s)he may specify the body of the SDL process as well as global variables, for instance. Furthermore, the level of abstraction (whether the real M-net or just an abstract Icon is displayed) may be chosen by the user as well. We will adapt ideas from the COOs [25] and the METAFrame [2] approach.

7 Conclusion

We have presented a new approach to combine different modelling techniques (finite automata, parallel programs and SDL specifications) which allow the user to profit at the same time from the advantages of all these formalisms. These are in particular,

1. powerful and efficient modelling of the control flow using finite automata,
2. high expressiveness (including, for instance, non-determinism, multiway synchronous as well as asynchronous communication and change of multiple variables at a time) of atomic actions in $B(PN)^2$,
3. and nice graphical appearance as well as a complementary communication mechanism in SDL.

Considering an ARQ protocol, we have shown that our approach enables the hybrid modelling of parallel systems and the subsequent application of the rich set of verification methods included in the PEP tool [3, 7, 18]. The fact that our approach is based on M-nets (an algebra of high-level Petri nets) at the one hand enabled the composition operations as well as the verification, but on the other hand does not imply that the user has to be familiar with the technicalities of M-nets because they may be hidden using an extension of the reference scheme presented in [17].

As a side effect we have described on-going development of the PEP tool which will not only result in a smooth integration of the presented approach, but also support a new 'programming with nets' technique.

Acknowledgement:
I would like to thank Eike Best and anonymous referees for their comments; Martin Ackermann, Ulf Fildebrandt, Michael Kater, and Stefan Schwoon for their implementations; and Hans Fleischhack for his suggestion to work on the hybrid modelling approach.

References

1. M. Ackermann. *Konzeption eines Compilers für eine parallele Programmiersprache mit Prozeduren.* Diploma thesis, Universität Hildesheim, 1997.
2. M. v. d. Beeck, V. Braun, A. Claßen, A. Dannecker, C. Friedrich, D. Koschützki, T. Margaria, F. Schreiber, and B. Steffen. Graphs in METAFrame: The Unifying Power of Polymorphism, In E. Brinksma, editor, *Proc. of TACAS'97, Enschede*, *LNCS 1217*, 112–129, Springer, 1997.
3. E. Best. Partial Order Verification with PEP. In G. Holzmann, D. Peled, and V. Pratt, editors, *Proc. of POMIV'96, Princeton*, 305–328. Am. Math. Soc., 1996.
4. E. Best, R. Devillers, and J. G. Hall. The Box Calculus: a New Causal Algebra with Multi-Label Communication. In G. Rozenberg, editor, *Advances in Petri Nets 92*, *LNCS* 609, 21–69. Springer, 1992.
5. E. Best, H. Fleischhack, W. Frączak, R. P. Hopkins, H. Klaudel, and E. Pelz. An M-Net Semantics of $\mathrm{B(PN)}^2$. In J. Desel, editor, *Proc. of STRICT*, Workshops in Computing, 85–100, Springer, 1995.
6. E. Best, H. Fleischhack, W. Frączak, R. P. Hopkins, H. Klaudel, and E. Pelz. A Class of Composable High Level Petri Nets. G. De Michelis and M. Diaz, editors, *Proc. of ATPN'95, Torino*, *LNCS* 935, 103–118. Springer, 1995.
7. E. Best and B. Grahlmann. *PEP: Documentation and User Guide.* Universität Hildesheim. Available together with the tool via: http://www.informatik.uni-hildesheim.de/~pep.
8. E. Best and R. P. Hopkins. $\mathrm{B(PN)}^2$ – a Basic Petri Net Programming Notation. A. Bode, M. Reeve, and G. Wolf, editors, *Proc. of PARLE*, *LNCS* 694, 379–390, Springer, 1993.
9. CCITT. *Specification and Description Language*, CCITT Z.100, Geneva, 1992.
10. E. Clarke, K. McMillan, S. Campos, and V. Hartonas-Garmhausen. Symbolic Model Checking. In R. Alur and T. A. Henzinger, editors, *Proc. of CAV'96, New Brunswick*, *LNCS* 1102, 419–422, Springer, 1996.

11. J. Esparza, S. Römer, and W. Vogler. An Improvement of McMillan's Unfolding Algorithm. In T. Margaria and B. Steffen, editors, *Proc. of TACAS'96, Passau, LNCS* 1055, 87–106, Springer, 1996.
12. J. Esparza. *Model Checking Using Net Unfoldings.* In Number 23 in *Science of Computer Programming*, 151–195, Elsevier, 1994.
13. H. Fleischhack and B. Grahlmann. *A Petri Net Semantics for $B(PN)^2$ with Procedures. Proc. of PDSE'97*, 15–27, Boston, IEEE Comp. Soc. Press, 1997.
14. H. Fleischhack and B. Grahlmann. *A Compositional Petri Net Semantics for SDL*, Technical report, HIB 18/97, Universität Hildesheim, 1997.
15. H. Fleischhack and B. Grahlmann. *Towards Compositional Verification of SDL Systems. Proc. of 31st HICSS – Software Technology Track*, 404–414, IEEE Computer Society Press. 1998.
16. B. Grahlmann, M. Moeller, and U. Anhalt. A New Interface for the PEP Tool – Parallel Finite Automata. *Proc. of 2. Workshop Algorithmen und Werkzeuge für Petrinetze*, AIS 22, 21–26. FB Informatik Universität Oldenburg, 1995.
17. B. Grahlmann. The Reference Component of PEP. In E. Brinksma, editor, *Proc. of TACAS'97, Enschede, LNCS 1217*, 65–80, Springer, 1997.
18. B. Grahlmann. The PEP Tool. In O. Grumberg, editor, *Proc. of CAV'97, Haifa, LNCS 1254*, 440–443, Springer, 1997.
19. B. Graves. Computing Reachability Properties Hidden in Finite Net Unfoldings. *Proc. of FST&TCS'97. LNCS*, Springer. 1997.
20. G. Holzmann and D. Peled. The State of SPIN. In R. Alur and T. A. Henzinger, editors, *Proc. of CAV'96, New Brunswick, LNCS 1102*, 385–389. Springer, 1996.
21. J. E. Hopcraft and J. D. Ullmann. *Introduction to Automata Theory, and Languages, and Computation.* Addison Wesley, 1994.
22. B. W. Kernighan and D. M. Ritchie. *The C Programming Language* Prentice Hall, 1988.
23. S. Melzer and J. Esparza. Checking System Properties via Integer Programming. In H. R. Nielson, editor, *Proc. of ESOP'96, LNCS 1058*, 250–264, Springer, 1996.
24. S. Melzer and S. Römer. Synchronisierende Automaten in PEP. *Proc. of 3. Workshop Algorithmen und Werkzeuge für Petrinetze*, Technical Report 341, 52–59. AIFB Universität Karlsruhe, 1996.
25. C. Sibertin-Blanc. Cooperative Nets. In R. Valette, editor, *Proc. of ATPN'94, LNCS 815*, 471–490, Springer, 1994.
26. P. H. Starke. *INA: Integrated Net Analyzer.* Handbuch, cf. http://www.informatik.hu-berlin.de/~starke/ina.html.
27. G. Wimmel. A BDD-based Model Checker for the PEP Tool. *Technical Report, University of Newcastle upon Tyne*, 1997.