HYBRID HIGH-LEVEL NETS*

Ralf Wieting

Oldenburger Forschungs- und Entwicklungsinstitut für
Informatik-Werkzeuge- und Systeme (OFFIS)
Escherweg 2, D–26121 Oldenburg, GERMANY

ABSTRACT

This paper presents a new methodology for modeling and simulation of hybrid systems which is called Hybrid High-level Nets (HYNETS). The new methodology integrates three established modeling approaches into one language. High-level Petri Nets represent the basic discrete framework, differential algebraic equations (DAEs) are used to describe continuous systems behavior and object-oriented concepts improve the expressiveness and compactness of models. The paper explains basic ideas of the language and illustrates its usefulness with a small example.

1 INTRODUCTION

Modeling and simulation of hybrid systems, i. e. systems consisting of a mixture of discrete and continuous components, is a research area that becomes more and more interesting. This is due to the fact that most systems of real world applications are not purely discrete nor purely continuous and often both parts influence each other.

Consider for instance the small example in Figure 1: A level of liquid in a tank changes continuously dependent on the output of a pump and the opening degree of a valve. Two level indicators report to a control unit when a certain level is reached. The control unit reacts on these discrete events by sending instructions for regulating the pump and the valve in such a way that the tank level always stays in between level S_1 and level S_2.

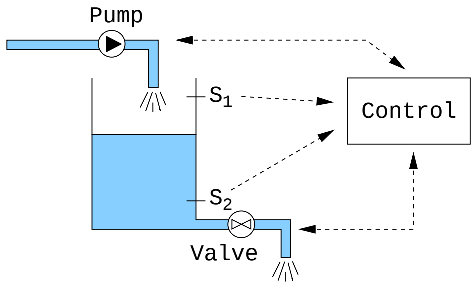


Figure 1: Example of a Hybrid System

Modeling methodologies which are able to describe these kinds of systems in a convenient way and which allow an efficient analyzation or simulation of models to get information about the systems' behavior are very useful in a wide range of application areas; examples are manufacturing systems, production lines, chemical plants, mechatronic systems, process control and ecological systems.

This is the reason why many hybrid modeling methodologies have been and will be proposed. Most of the known approaches base upon continuous simulation languages which are extended by components allowing the specification of discrete actions as well. Cellier (1979) was one of the first who introduced a sound methodology for specifying and simulating hybrid systems. By his approach a systems description is divided into a continuous and a discrete part and the interaction is governed by an interface. Similar interface methods followed (cf. Göllü and Varaiya 1989). The main disadvantage of these approaches is the fact that all continuous systems behavior has to be specified completely by one single system of differential equations. A more recent methodology tries to overcome this shortcoming by using object-oriented concepts (Elmqvist et al. 1993).

In this paper I present a hybrid modeling methodology which, in contrast to the approaches mentioned above, is based upon a discrete language, namely high-level Petri nets. Before I explain the basic ideas of this approach in Section 3 and the evolution of hybrid models in Section 4, I give a brief overview of the different classes of Petri nets in the next section. In Section 5 I illustrate the capabilities of the new language. Finally, in Section 6 I summarize the main results and give some ideas on future objectives.

*In *Proceedings of the 1996 Winter Simulation Conference*, Coronado, California, USA, Dec. 1996.

2 PETRI NETS

Petri nets were originally introduced by C. A. Petri (1962). Since then a lot of variants and extensions of Petri nets have been proposed and investigated. See Murata (1989) for a comprehensive introduction.

Due to their good properties in theoretical analysis, practical modeling, and graphical visualization of concurrent discrete systems, they are in use in a wide range of application areas. Especially high-level Petri nets (Jensen and Rozenberg 1991), which lead to more succinct and manageable models, have shown good suitability.

Nevertheless, Petri nets are basically discrete models. They are not inherently qualified for modeling continuous systems. In order to overcome this drawback but still preserve the good properties of Petri nets several continuous extensions have been proposed. David and Alla (1992) and Trivedi and Kulkarni (1993) independently introduced two formal hybrid modeling approaches on the basis of Place/Transition Petri nets. In both cases basic extensions relate to a new kind of real valued tokens which reside on continuous places. These tokens can be modified continuously by other net elements.

Both approaches have already shown good results for investigating small hybrid systems (cf. Petterson and Lennartson 1995). Thus, in order to improve the expressiveness and compactness of hybrid models it is obvious to integrate similar extensions into high-level Petri nets. First ideas from this conclusion have been presented in Wieting and Sonnenschein (1995). A more detailed description of the new approach is presented in Section 3.

A slightly different approach which is also based on a class of high-level Petri nets is presented in (Brielmann 1995). In this approach extended Predicate/Transition Nets serve as a basis and first order differential equations are used to describe continuous systems behavior. But in contrast to my proposal the differential equations are transformed into according difference equations before they are modeled with elements of the discrete Predicate/Transition Nets.

3 HYBRID HIGH-LEVEL NETS

In this section I will introduce the new modeling methodology of **Hy**brid High-level **Net**s (HyNets).

The basic framework of HyNets is represented by a special class of high-level Petri nets called **T**imed **H**ierarchical **O**bject-**R**elated **N**ets (Thorns) (cf. Schöf, Sonnenschein, and Wieting 1995). These nets already offer several reasonable features for modeling complex discrete systems. Two different timing concepts (delay times and firing durations) related to transitions allow the description of many different temporal processes and dependencies. Due to an appropriate hierarchy concept even large models can be structured into clear peaces. Finally, an object-oriented programming language, namely C++, is used to define complex objects and to inscribe transitions with activation conditions and firing actions, etc. In this way even complex discrete systems can be described in a compact and clear way.

In the following sections I explain how the concepts of Thorns can be enhanced to describe the behavior of continuous systems as well. Since most continuous systems can be described by differential algebraic equations (DAEs), it should be possible to specify ordinary first order differential equations in explicit form ($x' = f(x, y, t)$) and algebraic equations ($x = f(y, t)$) within the new language. Since C++ does not provide constructs for the specification of DAEs, this objective has to be achieved by the introduction of an enhanced inscription language.

3.1 Inscription Language

As pointed out above, the originally used inscription language of Thorns had to be enhanced to be able to specify DAEs. For this purpose a new **H**ybrid **o**bject-oriented **La**nguage called HoLa was developed (Majchszak 1996). The roots of HoLa are basically C++. Thus, the syntax is very similar and the basic object-oriented concepts like inheritance, polymorphism and dynamic binding are still present.

In order to distinguish continuously changing state variables from other variables in a HyNet model, HoLa offers the new elementary class `Real` next to standard classes like `Int`, `Double`, etc. Only objects or object attributes of type `Real` can be changed in a continuous way. Hence, only variables of this type may appear on the left hand side of a DAE.

Let us consider the example of Figure 1 again to illustrate, how a user can define a complex object type. Assuming the relevant attributes of the tank are its area A and the level l and density d of the liquid filled into it, a HoLa definition of the tank may look as follows:

```
class Tank {            Double Tank :: v(){
  Float    A;             return( A * l );
  Double   d;           }
  Real     l;           Double Tank :: p(){
  Double   v();           return( d * l );
  Double   p();         }
};
```

The definition shows a complex object type with the attributes mentioned above and two methods $v()$

and $p()$ where $v()$ calculates the liquid volume and $p()$ calculates the pressure on the bottom of the tank. Since the attribute l is of type `Real`, it can be modified by differential equations (see Section 5 for more details).

3.2 HyNet Structure

Following the customary notation for defining Petri Nets, I define the structure of a HyNet as a triple (P, T, A) satisfying the requirements below:

1. P is a final set of *places*.
2. T is a final set of *transitions* partitioned into two disjoint sets of discrete transitions T_D and continuous transitions T_C.
3. $T \cap P = \emptyset$ and $T \cup P \neq \emptyset$.
4. A is a final set of *arcs* partitioned into two disjoint sets of discrete directed arcs A_D and continuous undirected arcs $A_C \subseteq (P \times T_C)$. Furthermore, discrete arcs are partitioned into three disjoint subsets of standard arcs $A_S \subseteq (P \times T_D) \cup (T_D \times P)$, enabling arcs $A_E \subseteq (P \times T)$ and inhibitor arcs $A_I \subseteq (P \times T)$.

The semantics of these net elements is described in Section 4. The graphical representation is shown in Figure 2. As usual, places are represented by circles and transitions by rectangles. In order to distinguish continuous transitions from discrete ones, continuous transitions have two border lines.

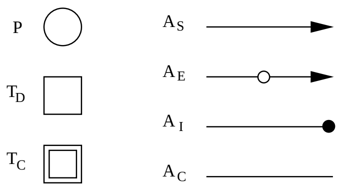
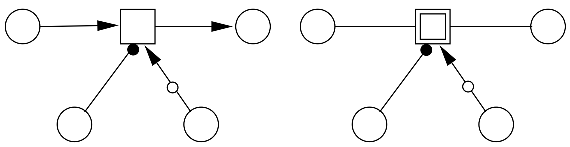


Figure 2: Graphical Representation

Figure 3 shows the allowed connections between places and transitions according to the definition above.



(a) (b)

Figure 3: Possible Connections between Places and Discrete (a) and Continuous (b) Transitions

3.3 HyNet Inscriptions

The following inscriptions can be assigned to the net elements of a HyNet.

Places are labeled by a *place type* and a *place capacity*. The type of a place defines the kind of objects which may reside on a place. This type is either an elementary HoLa-class (`Bool`, `Char`, `String`, `Int`, `Long`, `Float`, `Double`, `Real` or `Token`) or a user defined complex HoLa-class e. g. `Tank`. In the case that a place type is defined by a complex class, objects of derived classes are allowed to reside on that place, too.

The place capacity specifies the maximum number of objects held by a place at each moment. It is defined by a positive integer or `Omega` where `Omega` represents infinite capacity.

Arcs have an *arc weight* and a *variable name*. The arc weight of a standard arc and an enabling arc is a positive integer which defines how many object have to reside on the incident place to enable the incident transition. The weight of inhibitor and continuous arcs is one per default.

The variable name of an arc is used to reference objects of a place from a transition. Together with the weight of an arc a set of variables is defined. For example, if an arc is labeled with the weight 2 and the variable name x, two variables $x.at(1)$ and $x.at(2)$ are defined. If the arc weight is one the variable is defined directly by the variable name. Variables may be used by the inscriptions of a transition (see below). Inhibitor arcs don't need a variable name.

Transitions have the most comprehensive inscriptions:

- The *firing capacity* of a transition is a natural positive number or `Omega`. It specifies how often a transition can fire in parallel with itself. Since a continuous transition modifies all objects on surrounding places at each time, it has an infinite firing capacity per default.
- Every transition can be labeled with an *activation condition*. This condition is used to make further requirements to objects enabling that transition. It has to be specified as a boolean HoLa expression without side effects over variables of incoming arcs. In order to avoid name clashes of variable names two arcs incident to a transition must not have the same name.
- Another important inscription of a transition is the *firing action*. In case of a discrete transition the action is specified in form of a sequence of HoLa statements and in case of a continuous transition it is defined by a system of algebraic and first order

differential equations respecting HoLa syntax. In both cases variables of incoming and outgoing arcs can be used, but it has to be assured that only variables of type `Real` appear on the left hand side of equations.

- Finally, discrete transitions have two more inscriptions defining their behavior in time: a *delay time* and a *firing time*. Both inscriptions are specified as HoLa expressions without side effects over variables of incoming arcs. The results obtained by an evaluation of these expressions have to be of type `Double`.

Now, the structure of a HyNet and all possible inscriptions are introduced. In the following I will describe the semantics of the different net elements and their interactions.

4 EVOLUTION OF HYNETS

Evolution of a HyNet means changing the state of a net. This is done by the firing of transitions or more precisely by the firing of occurrence elements. Before I explain which conditions have to be fulfilled to enable transitions and what happens if a transition fires I have to describe the terms of a state and an occurrence element.

An *occurrence element* is a transition combined with a set of objects that is bound to variables of the transition's ingoing arcs. It is denoted as follows: $[t : < v_1 = o_1, \ldots, v_n = o_n >]$ where t is a transition, v_i are variables of ingoing arcs and o_i are objects from preset places bound to the variables.

The *state* of a HyNet is given by the current time of the state, the markings of places and information about firing occurrence elements. The latter is discussed in more detail in Section 4.4.

A marking of a place consists of the set of objects residing on a place. All objects are distinguished by an unambiguous identification number — that's the reason why we can speak of sets of objects instead of multi-sets of objects. Furthermore, objects are labeled with a time-stamp indicating the time of their instantiation. This time-stamp may be greater than the current time of the state if the object will be produced by a discrete transition in the future (see Section 4.1). In this case an object is called invisible. With respect to these additional labels an object is completely described by its value, its identifier and its time-stamp. I omit identifier and time-stamp if they are not necessary in some context.

4.1 Discrete Transitions

Since the definition of HyNets bases upon the definition of Thorns, the activation and firing of discrete transitions is quite similar to that of Thorn transitions (cf. Schöf, Sonnenschein, and Wieting 1995). Thus, I only summarize some essentials for firing discrete transitions and illustrate the firing with a small example.

Figure 4 shows a small net with a discrete transition and four places. The inscription of the transition is given on the right side of the figure. Place $p1$ has the type Int, an infinite capacity and a marking consisting of three objects (identifiers and time-stamps are not shown).

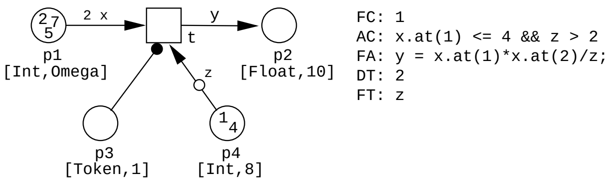


Figure 4: Discrete Transition and some Places

An occurrence element of a discrete transition will be delayed if the *precondition* of the transition is fulfilled. Therefore all places connected by inhibitor arcs have to be empty, every variable of an ingoing arc has to be bound by an object, and the activation condition has to be fulfilled by the bound objects.

Considering the example above, we only have two occurrence elements that fulfill the precondition: $[t : < x_1 = 2, x_2 = 5, z = 4 >]$ and $[t : < x_1 = 2, x_2 = 7, z = 4 >]$. These elements will be delayed.

The delay time of an occurrence element is determined by the according expression of the transition. If an occurrence element has been delayed for the whole delay time without interruption it is called completely delayed. Completely delayed occurrence elements begin to fire as soon as they are enabled. A discrete occurrence element is *enabled* if the precondition is fulfilled, the transition has free firing capacity, and places in the postset of the transition can receive the amount of objects produced by the firing of the transition.

Both occurrence elements of the example are completely delayed after two time units, but only one of them is enabled, because the transition t only has a firing capacity of one. This situation is called a conflict. Conflicts between discrete occurrence elements are solved by a nondeterministic choice — for more details about conflicts see Section 4.3. Let us assume that in this case $[t : < x_1 = 2, x_2 = 5, z = 4 >]$ is chosen.

At the *beginning of a firing* of an occurrence element all objects bound by variables of ingoing standard arcs are consumed from the corresponding preset places. Objects bound via enabling arcs are not consumed — these objects are only read by the transition. The firing capacity is decremented, the firing time is determined by the according expression, the firing action is executed, and produced objects are put on postset places. These objects get an unambiguous identifier and a time-stamp calculated by the current time plus the firing time.

In the example objects 2 and 5 are consumed from place $p1$ and a new object 2.5 is produced on place $p2$. The time-stamp of this object is calculated according to the current time. Notice: although this object is invisible in the current state, it already uses capacity of place $p2$.

At the *end of a firing* of an occurrence element only the firing capacity of the transition has to be incremented again. All produced objects get visible automatically, because the time of state equals the time-stamps of the objects now.

After four time units of firing, transition t of the example could fire again if there would be enough objects residing on place $p1$ then.

4.2 Continuous Transitions

The idea of continuous transitions is to change values of objects residing on places adjacent to the transition continuously. Only objects or object attributes of type `Real` can be modified. I explain the activation condition and the firing rule of a continuous transition on a small example, too.

Figure 5 shows a small net consisting of a continuous transition and four places together with the net inscriptions.

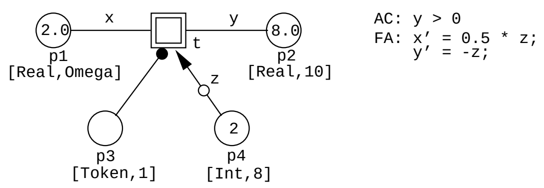


Figure 5: Continuous Transition and some Places

A continuous occurrence element is *enabled* under the same conditions as a discrete one. A precondition is not necessary because continuous transitions are not delayed. Thus, a continuous occurrence element is enabled if all places connected via inhibitor arcs are empty, every variable of an incident arc can be bound by an object, and the activation condition is fulfilled by the bound objects.

In the example of Figure 5 the occurrence element $[\,t : < x = 2.0, y = 8.0, z = 2 >]$ is enabled.

An enabled continuous occurrence element *fires* as long as it is enabled. Firing of a continuous occurrence element means continuously changing the values of bound objects according to the equations of the transitions firing action — algebraic equations assign values to objects and differential equations change object values. The firing does not influence the object identification numbers and time-stamps, since only object values are changed and no new objects are created. There are only two reasons why a continuous occurrence element becomes disabled:

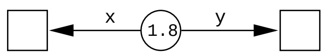
(1) the activation condition is no longer fulfilled or

(2) a discrete transition consumes an object bound by the occurrence element or produces an object on a place connected via an inhibitor arc.

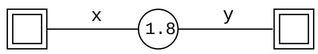
Considering the example above the continuous occurrence element fires exactly for four time units. Then, the object 8.0 bound to the variable y becomes zero and the activation condition is no longer true. The marking of place $p1$ at this moment is 6.0.

4.3 Conflict Handling

Before I describe the firing rule of a HyNet, I will explain the handling and resolution of conflicts between occurrence elements. I omit reflections on enabling and inhibitor arcs because they are not relevant in this context. There are five basic cases of conflicts:



Case 1: Two or more enabled discrete occurrence elements want to consume the same object. This is the classical conflict. It is solved by a nondeterministic choice: only one transition can fire.



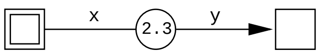
Case 2: Two or more enabled continuous occurrence elements want to change the same object or object attribute, respectively. In this case we have to distinguish three subcases depending on the firing actions of the transitions:

(a) Both transitions want to assign different values to the object by algebraic equations, e. g. $x = 3$ and $y = sin\, t$. This conflict is solved by a nondeterministic choice as in case 1.

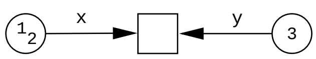
(b) One transition wants to set the object value by an algebraic equation, e. g. $x = 3t$, and the other one wants to change the object value by a differential equation, e. g. $y' = 0.5\,y$. Here we give higher priority to algebraic equations.

(c) Both transitions want to change the object value by differential equations, e. g. $x' = 3$ and $y' = sin\, t$. In this case the object will be shared by the transitions. The new object value is defined by the sum of changes. The advantage of this definition is shown with the example of Section 5.

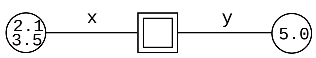
Since subcases (a) and (b) lead to states which are not well defined, a modeler should avoid such situations.



Case 3: An enabled discrete occurrence element wants to consume an object which is part of two or more enabled continuous occurrence elements. Here we give higher priority to the discrete occurrence element, i. e. the discrete transition can fire with the bound object and the according continuous occurrence element becomes disabled.



Case 4: A discrete transition is enabled by more than one occurrence element but it has not enough firing capacity to fire all of them at the same time. This internal conflict is solved in the same nondeterministic way as case 1.



Case 5: A continuous transition is enabled by more than one occurrence element. Since continuous transitions have infinite firing capacity no conflicts result from lacking firing capacity. Here, a conflict can only arise if two or more occurrence elements want to assign different values to the same object by an algebraic equation, e. g. if the firing action is $y = x$ such a situation occurs. This kind of conflicts is solved as in case 2 (a). Thus, in the example above, no assumptions can be made on the value of the object bound to y during the firing of the continuous transition (it may be 2.1 or 3.5).

These five cases represent the basic kinds of conflicts. More complex situations can be reduced to basic conflicts. Then, they are solved according to the specified rules.

The idea of sharing objects between continuous occurrence elements and giving discrete transitions higher priority over continuous ones was previously proposed by Le Bail, Alla, and David (1991) for hybrid Petri nets.

4.4 Firing Rule

As already mentioned at the beginning of this section, evolution of a HyNet means changing the state over time. This is done by the firing of discrete and continuous occurrence elements. The firing of a discrete occurrence element can be split into two discrete events, namely the beginning and the end of a firing. At these times, the state is changed by some discrete actions including the consumption and production of objects (cf. Section 4.1). Between such discrete events the state is modified continuously by enabled continuous occurrence elements (cf. Section 4.2). Within time periods where only continuous occurrence elements fire, no objects are consumed or produced, only object values are changed.

If we take the conflict handling of Section 4.3 into account the evolution of a HyNet can be formulated by the following algorithm:

1. Execute all actions belonging to discrete occurrence elements at the current time. This is done by interleaving actions which belong to enabled occurrence elements that start firing and occurrence elements that stop firing at this moment until no such element exists. Notice that the execution of these actions may enable or disable other (conflicting) occurrence elements and that time does not proceed in this step.
2. Determine all enabled continuous occurrence elements and change the values of bound objects according to equations specified by the firing actions until
   - a discrete occurrence element becomes enabled or finishes its firing or
   - a continuous occurrence element becomes enabled or disabled.
3. Continue with 1.

This algorithm implements a firing rule that always fires a maximum conflict free set of occurrence elements.

If in step two no continuous occurrence element is enabled no object values will be changed. Time elapses without any actions until one of the listed events occur. If none of these events will occur and no continuous occurrence element is enabled the algorithm does not proceed. The net is dead.

4.5 Evolution Graph

The evolution of a Petri net is usually represented by a reachability graph or coverability graph respectively (cf. Murata 1989). The nodes of these graphs represent markings and the edges relate to fired transitions responsible for the change of a marking.

A similar term, called evolution graph, was introduced by Le Bail, Alla, and David (1991) for hybrid Petri nets. Here, the nodes of a graph not only represent the marking of a net but also an invariant speed vector for continuous transitions and a vector of reserved marks. The edges between these nodes are labeled with transitions as for ordinary Petri nets.

The ideas of Le Bail et al. can be applied to an evolution graph for HyNets. With respect to the algorithm of Section 4.4 a HyNet evolution graph can be constructed as follows.

Every node of the graph represents a HyNet-state which is completely defined by

- the current time of state,
- the markings of all places including visible and invisible objects,
- a set of firing discrete occurrence elements together with the time specifying the end of their firing, and
- a set of firing continuous transitions.

As long as the firing rule algorithm stays in step two, a state (or a node of the evolution graph) is valid: time elapses and object values are changed continuously by the firing continuous transitions. A state changes to another state

- if the set of firing discrete occurrence elements changes, i. e. if a discrete occurrence element finishes firing or a new one starts firing, or
- if the set of firing continuous occurrence elements changes, i. e. if a continuous occurrence element becomes enabled or disabled.

The event responsible for a change of state can be used as a label of an edge in the evolution graph.

5 EXAMPLE

In this section I illustrate some features of HyNets with a small example. Figure 6 shows the HyNet model of the tank system introduced in Section 1. The elements on the left side describe the continuous behavior of the tank while the elements on the right model the discrete control unit.

The type of the place *Tank* on the left hand side of the figure is defined by the HoLa class presented in Section 3.1. Currently it holds a tank object with an area A of 8.0. The tank can be filled with liquid of density $d = 0.9$. The current liquid level of the tank is $l = 0.0$. The measures for these values have to be set properly according to physical laws.

In a similar way a complex object type `Valve` can be defined. A valve object has an attribute f for its maximum throughput, a real valued attribute o for its opening degree ($0 \leq o \leq 1$) and a method $t()$ which calculates the actual throughput ($f * o$). The place

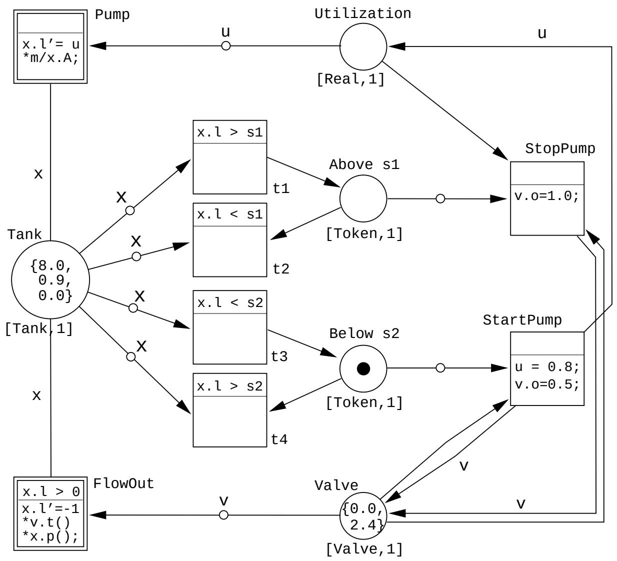


Figure 6: HyNet of the Example of Figure 1

Valve currently holds a closed valve with a maximum throughput of 2.4.

In the depicted situation only the discrete transition *StartPump* is enabled. Because it has a zero valued delay and firing time — like all other discrete transitions in this model — it fires immediately. Thereby the opening degree of the valve is set to 50% and an object 0.8 is produced on the place *Utilization*.

This place describes the power utilization of the pump. If it is marked, the continuous transition *Pump* can fire. Thus, in this state the tank level $x.l$ is changed continuously according to the differential equation $x.l' = u * m/x.A$ specified by the firing action of *Pump* (m describes the maximum output of the pump).

At the same time, the liquid level of the tank rises, the other continuous transition *FlowOut* is enabled. Then it fires continuously according to the equation $x.l' = -v.t() * x.p()$ where $x.p()$ specifies the pressure on the valve (cf. Section 3.1).

In this situation both continuous transitions change the liquid level of the tank concurrently. The level only rises if the output of the pump is larger than the throughput of the valve — let us assume this case.

The next event occurs when the level reaches the value $s2$. The discrete sensor transition $t4$ removes the token from the place *Below* $s2$. This has no further effects. Only when the level reaches $s1$ and the sensor transition $t1$ fires, the pump will be stopped and the valve will be opened to 100% by the firing of transition *StopPump*.

The following evolution of the net is straight forward. The sensor transitions $t1$ to $t4$ report values of the liquid level of the tank to the control transitions *StopPump* or *StartPump* and these transitions regulate the flow into and out of the tank. They do this in such a way, that, after an initialization phase, the liquid level always stays between $s1$ and $s2$.

6 CONCLUSION

In this paper, I have presented a new modeling methodology in which three established modeling approaches are combined.

- High-level Petri Nets represent the basic framework. They have a formal basis with a well-defined semantics and they provide an attractive graphical visualization of system models.
- Object-oriented concepts extend the expressive power of the methodology and lead to more succinct and manageable models.
- Differential algebraic equations allow the specification of continuous system behavior. Therefore many different continuous processes can be described conveniently.

Together, these concepts form a comprehensive and comfortable modeling approach capable of modeling complex hybrid systems in a wide range of application areas.

In the future, I want to show the suitability of HyNets by several larger case studies. Another important objective is the development of efficient simulation strategies which allow an extensive investigation of HyNets models by computer tools.

REFERENCES

Brielmann, M. 1995. Modelling Differential Equations by Basic Information Technology Means. Cadlab Rep. 7/95, Cadlab, Paderborn, Germany.

Cellier, F. E. 1979. Combined Continuous/Discrete System Simulation by Use of Digital Computers: Techniques and Tools. Dissertation, Swiss Federal Institute of Technology Zurich, Switzerland.

David, R., and H. Alla. 1992. *Petri Nets & Grafcet — Tools for modelling discrete event systems.* New York: Prentice Hall.

Elmqvist, H., F. E. Cellier, and M. Otter. 1993. Object-Oriented Modeling of Hybrid Systems. In *Proceedings of the European Simulation Symposium — ESS '93*, 31–41. Delft, The Netherlands.

Göllü, A., and P. Varaiya. 1989. Hybrid Dynamical Systems. In *Proceedings of the 28th Conference on Decision and Control*, 2708–2712. Tampa, Florida.

Jensen, K., and G. Rozenberg. 1991. *High-level Petri Nets — Theory and Application.* Berlin: Springer-Verlag.

Le Bail, B., H. Alla, and R. David. 1991. Hybrid Petri Nets. In *Proceedings of the European Control Conference*, 1472–1477. Grenoble, France.

Majchszak, R. 1996. Eine objektorientierte Beschriftungssprache höherer Petrinetze für hybride Modelle. Master's thesis, Fachbereich Informatik, Universität Oldenburg, Oldenburg, Germany.

Murata, T. 1989. Petri Nets: Properties, Analysis and Application. In *Proceedings of the IEEE* 77 (4): 541–580.

Petri, C. A. 1962. Kommunikation mit Automaten. Schriften des IIM Nr. 2, Institut für Instrumentelle Mathematik, Bonn, Germany. Also, English translation: 1966. Communication with Automata. Technical Report RADC-TR-65-377, Vol. 1, Suppl. 1, Griffiss Air Force Base, New York.

Pettersson, S., and B. Lennartson. 1995. Hybrid Modelling focused on Hybrid Petri Nets. In *Proceedings of the 2nd European Workshop on Real-time and Hybrid Systems.* Grenoble, France.

Schöf, S., M. Sonnenschein, and R. Wieting. 1995. Efficient Simulation of Thor Nets. In *Proceedings of the 16th International Conference on Application and Theory of Petri Nets*, ed. G. De Michelis and M. Diaz, volume 935 of *Lecture Notes in Computer Science*, 412–431. Turin, Italy.

Trivedi, K. S., and V. G. Kulkarni. 1993. FSPNs: Fluid Stochastic Petri Nets. In *Proceedings of the 14th International Conference on Application and Theory of Petri Nets*, ed. M. Ajmone Marsan, volume 691 of *Lecture Notes in Computer Science*, 24–31. Chicago, Illinois.

Wieting, R., and M. Sonnenschein. 1995. Extending High-level Petri Nets for Modeling Hybrid Systems. In *Proceedings of the IMACS Symposium on Systems Analysis and Simulation*, ed. A. Sydow, 259–262. Berlin, Germany.

AUTHOR BIOGRAPHY

RALF WIETING is a research assistant in the Department of Systems Modeling at the Oldenburger Forschungs- und Entwicklungsinstitut für Informatik-Werkzeuge- und Systeme (Offis). He received a diploma degree in computer science from the University of Oldenburg in 1992. His research interests include modeling and simulation of hybrid systems using high-level Petri Nets.