Systems Specification, Analysis, and Validation by means of Timed Predicate/Transition Nets and Logic Programming

F. Belli
University of Paderborn, Dept. of Electrical and Electronics Engineering, 33095 Paderborn, Germany
Email: bell@uni-paderborn.de

J. Dreyer
dre@omega.uni-paderborn.de

Abstract

In this paper a method will be presented for systems design, specification and analysis based on predicate/transition nets (PrT nets) and logic programming. In order to evaluate and optimize the behavior of systems, the PrT net models are extended with quantitative time. The firing times are assigned to transitions and are given by arithmetic expressions which might contain variables. An approach to transforming requirements driven PrT net models into logic programs is given. The generated logic programs contain the static structure and the dynamic behavior of the PrT net models and can be used for simulation and analysis purposes. Moreover, they can be viewed as a precise and lucid specification of the programs to be implemented.

Three examples are given which illustrate how Prolog goals can be used to validate system model properties. Example 1 describes the transformation steps for simple PrT net models. In Example 2 the generated Prolog program corresponding to a PrT net model containing formulae inside the transitions is described. Example 3 illustrates how time constraints of PrT net models with variable time can be validated.

1 Introduction

During the last years the effort for modelling of system behavior and the analysis of concurrent processes with Petri nets has increased considerably. This development is supported by a growing number of software-tools [9].

Because of the simple structure of their components, Petri nets, e.g. place/transition nets (P/T nets) [21], are excellent means to lucidly exemplify general problems of system organization, as to conflict solving, deadlock avoidance, etc. On the other hand, because of the uniformity of the basic net elements, descriptions of larger systems by P/T nets tend to produce very large graphical representa-

Copyright 1995 by the Institute of Electrical and Electronics Engineers, Inc. Printed in USA. All rights reserved.

68
sions are not necessarily fixed, i.e. the expressions can contain variables.

In Chapter 2 the relevant features of PrT nets are summarized. The third chapter introduces VTT-PrT net models and their corresponding specification and simulation levels. Chapter 4 describes the transformation of a PrT net model into a Prolog program. Chapter 5 demonstrates the approach depicting its strength with three examples. The first example describes the transformation steps and the generated Prolog program corresponding to a simple PrT net model. Besides the model dependent predicates also model independent predicates are listed which allow to validate following system properties: reachability of system states, n-bounded places and M-dead transitions. The second example displays the generated Prolog program corresponding to a PrT net with formulae inside transitions and complex arc labels. Example 3 demonstrates how time constraints of VTT-PrT net models can be validated. In Chapter 6 the shortcomings and advantages of our approach are outlined and performance measurements of the realized simulations are provided using the examples introduced in Chapter 5. Finally, Chapter 7 summarizes the paper and discusses the results.

2 PrT nets

In the following some notions of the Petri net theory used in this paper are summarized. A P/T net model is composed of 4 different graphical elements:

- places (nodes represented by circles): Places can be considered as conditions. A condition is satisfied if the place contains at least as many tokens as specified by the corresponding outgoing arc.
- transitions (nodes represented by rectangles): Transitions can be considered as the active elements of a net. By firing, they change the actual state of the net.
- a flow relation between nodes (represented by arrows): Only nodes of different types can be connected directly by an arrow.
- tokens (represented by dots): A place can contain zero, one or more tokens. Tokens are often used to represent resources.

A simple P/T net model is shown in Fig. 1. Definitions concerning P/T nets are given in Appendix A.1. Compared to the P/T nets the predicate/transition nets (PrT nets) have the following properties [5, 10]:

- Distinguishable items (tuples with individual values) may be used instead of identical, unstructured tokens.
- The conditions for the firing of transitions may additionally be specified by means of some logical or algebraic relations between input and output variables of the transition.

- Each transition of a PrT net may represent an entire class of events. In the same way, a predicate may represent an entire class of elementary places.

A formal description of PrT nets is given in [10]. Following assumptions are made:

- Let the capacity of a predicate denote the maximum number of tuples that the predicate can carry simultaneously.
- Let n denote the arity of a predicate. Then all the items of the predicate are n-tuples. n can vary from predicate to predicate.
- In the following example n-tuples are always delimited by angular brackets. Arcs between the same elements and with the same direction can be replaced by one arc with a symbolic sum containing a linear combination of the tuples corresponding to the replaced arcs as label. For brevity, arc labels are also allowed to contain scalar products. E.g. the label \(<x, y> + <z, z> + <z, z>\) can be replaced by \(<x, y> + 2 <z, z>\).
- P/T nets can be considered as a special class of PrT nets where all predicates (places) are 0-ary, i.e. the marking of the predicates and the labels of the arcs are multiples of the zero-tuple \(<\>\).

In Fig. 2 a PrT net model is presented, demonstrating the firing of the transition T1. Following tuple sums of predicate P1 fulfill the restriction specified by the label of

![Figure 1: P/T net model](image1)

![Figure 2: PrT net before and after firing transition T1](image2)
the arc \((P_1, T_1)\): \(< 3, 4 > + < 3, 5 >, < 3, 5 > + < 3, 4 >, < 8, 6 > + < 8, 7 >, < 8, 7 > + < 8, 6 >\).

Besides this restriction, the expressions inside \(T_1\) have to be valid. The first expression \((x > z)\) is not valid for the first two sums. As the third expression excludes the fourth sum, only the sum \(< 8, 6 > + < 8, 7 >\) is valid. For predicate \(P_2\) only the sum \(< 8 > + 2 < 4 >\) is valid because of the second expression inside \(T_1\).

## 3 PrT nets with variable transition time

Causal, classical Petri nets are not suitable for evaluating the performance of systems since they neither contain information about the duration of the processes nor specify the instant of the initiation of an enabled transition. Performance evaluations can be realize using timed Petri net models, i.e. models with explicit quantitative time.

Timed Petri nets were introduced by Ramchandani [20]. He assigned to each transition a fixed (deterministic) firing time. Merlin [17] assigned to each transition minimum and maximum delays. Sifakis [22] associated the delays with places and proved that his approach is equivalent to the approach with delays assigned to transitions. In Stochastic Petri Net models [18] the firing times are not given by fixed delays, instead the durations are random variables with certain probability distributions.

The most common approach of timed Petri nets is the approach, where the firing delays are assigned to transitions. This paper is also based on that approach because it allows a natural representation of the reality.

Transitions with fixed durations are suitable for processes with constant execution times. E.g. the time required for transporting a component on a conveyor from A to B does not depend on the component. However, fixed durations are not suitable for processes like assemblies where the duration depends directly on the product to be assembled. E.g. it makes a big difference if 2 or 20 screws have to be fixed. Variable firing times are used for instance in [11].

The PrT net models are extended in the present approach by assigning a firing time which is given by an arithmetic expression. An arithmetic expression can contain also variables. Two net levels are distinguished in this paper:

- The *specification level* model is composed of predicates and timed transitions containing constraint inscriptions and the delay time. It does not determine how the model clock is realized.

- The *simulation level* model can be considered as the refinement of the specification level model. It is generated from the specification level model by replacing timed transitions by a nontimed *start-transition*, an *internal predicate* and a nontimed *end-transition* as displayed in Fig. 3 (see also [2]). Additionally a simulation level comprises a special predicate \((P_0)\) that contains an item with the actual time of the central clock. \(P_0\) has incoming (outgoing) arcs from (to) all transitions of the net. Except the central clock a simulation level model is in accordance with the definition of PrT nets.

Following assumptions are made:

- The firing process of a transition cannot be interrupted.
- A transition cannot fire concurrently to itself, i.e. a second firing can only be started after the first one is finished. This restriction can be easily realized by setting for all internal predicates the capacity to 1.
- If a transition changes the state *not enabled* to the state *enabled* at the time \(t_0\) then the transition is obliged to fire at this time (if this transition is not involved in a conflict). This strategy is called *earliest firing rule*.
- Every transition has a firing duration greater than zero.

Suppose the transition \(T_1\) becomes enabled at time \(t_0\) and the delay time of \(T_1\) is \(z_1\). Let \(T_1b\) be the corresponding simulation level *start-transition* and \(T_1e\) the *end-transition*. Then, if \(T_1\) is not in conflict with another transition, \(T_1b\) is fired at \(t_0\). Transition \(T_1e\) is fired at \(t_0 + z_1\), if all postconditions of \(T_1e\) are fulfilled. This means that the item in the inner predicate \(PT_1\) remains at least \(z_1\) time units inside the predicate and the delay time of \(T_1\) specifies the minimum delay time. This model behavior is an adequate representation of the reality, since for instance a machine is occupied as long as the processed component is not unloaded.

As the introduction of time is an additional restriction, the set of possible firing sequences of a timed Petri net is a subset of the set of firing sequences of a nontimed Petri net. Although the restriction *earliest firing rule* leads normally
to a considerable complexity reduction, the complexity of large models may still be too high. The search routine was improved in the present approach by storing and actualizing the temporary minimum time required for the execution of a predefined task. If the time required for a part of the task is equal to the temporary minimum, then the search branch is discarded. A drastic reduction of the complexity can be achieved by facts that contain technological knowledge about the process. If for instance components to be processed arrive on a conveyor and the components on the conveyor are represented by items inside the predicate conveyor then it makes sense to introduce FIFO (first in first out) predicates (see also [16]), i.e. predicates where the items have to be extracted in the order they arrived at the predicate. Another approach for reducing the complexity of the simulation of large models is presented in [6]. The main idea of this approach is to apply to hierarchical nets the divide and conquer strategy.

The firing sequence that performs a specified task needing a minimal amount of time units and fulfilling the earliest firing rule is not necessarily the absolute optimum. It cannot be guaranteed that a search routine using only local rules finds the absolute optimum because it may be better to wait for the enabling of other transitions. Because of the complexity of the problem, scheduling algorithms with global search strategies [7] can often not be used for large models.

4 Transforming PrT nets into logic programs

An introduction into logic programming is given in Appendix A.2. The approach presented in this paper is based on results of our previous work [3]. To explain the approach following basic notions are summarized.

Definition 4.1 Let $F$ be a flow relation and $F \subseteq (P \times T) \cup (T \times P)$. For each transition $t \in T$ of a PrT net there exists an input set $\bullet \leftarrow$ and an output set $\bullet \rightarrow$. The sets are defined as follows [21]: $\bullet \leftarrow := \{ p \in P \mid (p, t) \in F \}$ and $\bullet \rightarrow := \{ p \in P \mid (t, p) \in F \}$.

Definition 4.2 (see also [21]) Let $L(x, y)$ denote the label of the arc $(x, y) \in F$. The incidence matrix of a PrT net consisting of $m$ predicates (places) and $n$ transitions is an $n \times m$ matrix $I$. The elements of $I$ are defined by

$$I[t, p] := \begin{cases} L(t, p), & \text{if } p \in t \bullet \leftarrow \\
-L(p, t), & \text{if } p \in t \bullet \rightarrow \\
L(t, p) - L(p, t), & \text{if } p \in t \bullet \rightarrow \\
0, & \text{otherwise} \end{cases}$$

To achieve the aim of simulating the dynamic behavior of a system by means of a Prolog program, in addition to the static structure of the PrT net model the state-transformation caused by firing a transition has to be considered.

4.1 Transformation steps

- **Step 1:** Generation of the incidence matrix of the PrT net model.
  For each $t \in T$ perform the following steps:
- **Step 2:** Generation of Prolog body goal $b_1$.
  $b_1$ is satisfied if for all negative entries of the incidence matrix belonging to $t$ the items specified by the arc labels are available in the corresponding input predicates.
- **Step 3:** Generation of Prolog body goal $b_2$.
  $b_2$ is satisfied if for all positive matrix entries belonging to $t$ the firing of $t$ does not violate the upper limit of identical items in the output predicates.
- **Step 4:** Generation of body goal $b_3$. $b_3$ is satisfied if the formula in $t$ is valid.
- **Step 5:** Generation of body goal $b_4$. $b_4$ transforms $M$ into $M'$ (firing of $t$).
- **Step 6:** Generation of a Prolog clause which has the following form:
  $$c(M_0, M_1) : = b_1(...), b_2(...), b_3(...), b_4(...)$$
where $M_0$ is the actual marking and $M_1$ is the marking of the net after firing $t$.

4.2 Net simulation

The analysis of PrT net models can be divided into structural analysis and dynamic analysis. Structural analysis includes properties like whether the model contains isolated predicates, isolated transitions or whether the model is self-loop-free. Structural model properties can be computed by analysing the generated Prolog program. For instance it can be simply checked if a model is self-loop-free by examining for each transition if a predicate belongs to the set of input predicates as well as to the set of output predicates.

The dynamic behavior of PrT net models can be simulated by means of queries to the corresponding Prolog program. The simulation results of a modeled manufacturing system can be used in order to determine if the dynamic properties of the system model are in accordance with the expected behavior of the system, i.e. if the manufactured products reach the output predicates and possess the expected features. In order to determine whether a state $M_1$ can be reached from an initial state $M_0$ a query of the form $q(M_0, M_1)$ can be entered. $M_1$ is reachable if the system answers "yes".

During the design phase of a manufacturing system, simulation can be employed for instance for testing if a buffer is big enough or if the buffer is a bottleneck that has to be removed by modifying the model.
5 Examples

The first example depicts a PrT net model of a simple manufacturing system. It contains no formulae inside transitions, no arc labels with formal sums and demonstrates how a PrT net model can be represented by a Prolog program and how the model properties and the dynamic behavior of the model can be tested. Example 2 describes how PrT net models, containing formulae inside transitions and arc labels with formal sums, can be represented by a corresponding Prolog program. The third example displays a VTT-PrT net model and outlines how for a given goal the optimal solution (minimal time) can be computed for this model.

5.1 Example 1

The manufacturing system represented in Fig. 4 is composed of a pallets pool, a components store, a load station, an assembly cell and a conveyor. The PrT net model in Fig. 5 illustrates the dynamic behavior of the manufacturing system.

- At least one order (o) in the order storage (P1) is the precondition for the activation of the production. First a request for the transport (r) and a request for the components (c) are generated (T1).
- If the pallets pool is not empty, a pallet (p) is moved to the load station (T2) according to the transport request. In the load station components (c) which have been previously brought from the store are placed on the pallet. After the pallet has been loaded and moved to the assembly cell, this pallet will be unloaded.
- The assembly cell contains two concurrent working robots (r1 and r2). The robots are closely located and have overlapping workspaces. In order to avoid collisions between r1 and r2 only one robot is allowed to be in the common area at a given time point (P17). In the assembly cell the components will be taken by r1 or r2 from the buffer placed in the common area. The products (pr) are assembled and finally placed in the common area on the conveyor.

In this example, all predicates have infinite capacity and no transition has a logic or algebraic formula. Therefore, the transformation steps 3 and 4 and the body goals b, b2 can be omitted (see also 4.1). As all predicates of the example contain only items of one type, the body goal b1 is implemented as follows:

\[ \text{precondPrT(State_list, Weight_list)} \]

where the ith element of the State_list specifies the actual amount of items in the ith predicate (pi) and the ith element of Weight_list specifies the weight of the arc (pi, t). If the arc (pi, t) does not exist, the weight value is zero. Let len be the length of the lists (number of predicates). b1 is satisfied if for all i, \(1 \leq i \leq \text{len}\), ith element of State_list \(\geq\) ith element of Weight_list.
The predicate pre/2 can be used to ask if a predicate (place) is n-bounded for a given initial marking.

**Definition 5.1** A place is n-bounded, if the number of items in the place is always less than n + 1.

In order to determine whether the load station components buffer which is able to store up to 3 components has always sufficient room, the following query can be entered:

?- pre([3,4,0,0,200,0,0,0,0,0,0,0,0,1], [0,0,0,0,0,0,0,2,0,1,0,0,0,0,0,1]).

If the system answers this query with \( \rho \) then place \( P7 \) is 3-bounded, otherwise it is not 3-bounded because a state can be reached where \( P7 \) contains more than 3 tokens. Predicate pre/2 can also be used to test if, for a given initial state (first argument of the predicate), the transition specified by the second argument is an \( M \)-dead transition.

**Definition 5.2** A transition \( T \) is called \( M \)-dead if starting from state \( M \) there exists no firing sequence that leads to a state where \( T \) is enabled.

If \( pre/2 \) is not satisfied then the specified transition is \( M \)-dead. Transition \( T7 \) can be tested with the following query:

?- pre(M, [0,0,0,0,0,0,0,2,0,1,0,0,0,0,0,1]).

5.2 Example 2

Now consider the PrT net model of Fig. 2 which contains a formula inside the transition and arc labels with formal sums. Part of the listing of the corresponding Prolog program is shown in Fig. 7. The complete listing can be found in [3]. Except predicates transition/2 and state/1 all other predicates are model independent. The Prolog predicate transition/2 contains the following body goals:

- for each input predicate of \( T1 \) a body goal precond-PxT which is fulfilled if the input predicate contains a sufficient number of appropriate items,
- for each expression inside \( T1 \) an equivalent body goal,
- postcond-PxT which is fulfilled if the firing of \( T1 \) does not violate the maximum capacity (third argument) of any output predicate,
- for each input predicate of \( T1 \) a body goal firingRem which removes the items specified by the corresponding arc label from the predicate list,
- for each output predicate of \( T1 \) a body goal firingAdd which adds the items specified by the corresponding arc label to the predicate list.

The state of the PrT net after firing Transition \( T1 \) can be computed by entering the following query:

?- m(State),transition(State,New_state).
5.3 Example 3

The VTT-PtT net model of a manufacturing system displayed in Fig. 8 is composed of several conveyors and three robots. The inscriptions of the transitions T2, T3 and T4 contain following expressions:

- The first expression assures that the maximum load capacity of the robot is not exceeded.
- The second expression assures that the robot is able to grip the components, i.e., it checks if the width of the component is not greater than the maximum distance between the fingers of the gripper.
- The third expression specifies the (variable) firing time.

The inscriptions indicate that the robots have different machine parameters. Robot 3 is able to handle products with weights up to 15 kg whereas the maximum load capacity of robot 1 is 4 kg. The gripper of robot 1 can open up to 50 mm. Robot 3 needs 2 seconds for inserting and fastening one screw and 10 seconds for finishing the assembly (fixed product-independent time) of a product. For assembling a product with 5 screws robot 3 needs therefore 20 seconds.

A tuple inside predicate P1 specifies a product to be assembled by one of the available robots. The first element of a tuple can be considered as a product identifier, the second element specifies the required type of the A-component and the third element specifies the required type of the B-component. A tuple inside predicate P2 specifies an A-component. The first tuple-element is the subtype of the component, the second specifies the weight of the component and the third specifies the number of screws to be inserted and fastened by the robot. A tuple inside predicate P3 corresponds to a B-component. The first tuple-element indicates the subtype, the second the weight of the component and the third the width of the component. Transition T1 selects the component(s) required by the order and moves them into the buffer. The components belonging to one product are assembled by a robot that is able to perform this task and moved on a conveyor to the products buffer.

It should be noticed that the product with the identifier 1 can only be assembled by robot 3 and the products 2 and 4 can be assembled in two different ways since two A-com-
ponents are of subtype 1. The goal state is a state where predicate P8 contains 3 tuples and the predicate containing the clock time contains the minimal time.

The simulation is performed in the following way: First all end-transitions that are enabled at the actual time are fired, next all enabled start-transitions are fired. If no transition belonging to the net model is enabled, then the clock transition is fired. These steps are performed until the goal state is reached with a minimum amount of time units. According to the earliest firing rule only transitions that are in a conflict with other transitions are backtrack during the search process. All other transitions have to fire when they became enabled.

The minimal time for assembling the three specified products is 29 seconds. The corresponding assembly solution is shown in Fig. 9.

6 Discussion

6.1 Advantages of the approach

The combination of PrT nets and logic programming and the presented transformation procedure leads to a programming environment with the following advantages:

- It is easy for everyone to build an automatic translator, giving easy access to simulation and automatic verification.
- PrT nets are very suitable for modeling systems because of their simple structure, because they can represent concurrency and conflicts and because of their mathematical basis for analysis. This means that apart from being a modelling tool, PrT net models also provide a tool for the evaluation of system properties.
- PrT nets are high level programming language that allows the expression of problems in declarative manner by describing what has to be done instead of determining, like in conventional imperative programming languages, how it has to be done. As Prolog programs are executable, the problem description is at the same time the problem solution.
- The PrT net model is transformed automatically into a Prolog program. The transformation preserves the dynamic behavior of the PrT net model. A Prolog program derived from a PrT net model can be used to simulate the dynamic behavior of the modeled system. Thus Prolog serves as a simulation language about net properties. A feature that makes Prolog attractive as simulation language is that it allows rapid prototyping [14].

- Whereas there exist already powerful CASE (computer aided software engineering) tools for imperative programming languages which support the programmer during the software engineering process, no comparable tools exist for logic programming. The combination of PrT nets and logic programming presented in this paper outlines how this gap can be closed. The automatically generated Prolog programs can be considered as the second step of the software engineering process as shown in Fig. 10. In the following steps these programs are refined and parallelized manually (because of efficiency reasons) until executable programs - here parallel CS Prolog programs [15] - are created.

- Unlike almost all other parallel programming languages, parallel logic programming languages inherently express parallelism. As the parallelisms expressed by a PrT net model are also contained in the derived Prolog program, the program can be easily converted into a parallel Prolog program (see also [8]).
- There exist sophisticated tools for simulating the behavior of Petri net models. The languages of these tools are specially developed for simulation and animation purposes. However, they are not as powerful as Prolog queries and they are not suitable for implementing the modeled systems.

6.2 Shortcomings of the approach

The simulation of a PrT net model can have exponential complexity for some models. As this is a consequence of the application and not of Prolog, current simulators have the same problem.

In Prolog no iterative constructs exist as such. Instead, recursion is used to specify both recursive and iterative algorithms. The principal advantage of iterative algorithms over recursive algorithms is in efficiency, mostly space efficiency. Despite that, there exists a class of recursive Prolog programs using tail recursion that corresponds quite closely to conventional iterative programs. Such recursive Programs can almost be as efficient as iterative programs implemented in a conventional language.

A problem arises when traversing a graph with cycles. In this case the predicate net/2 might never terminate. Infinite loops can be avoided effectively by not allowing the same state to be visited twice. This can be achieved by adding an accumulator of reached states and by checking before an edge is traversed whether the next state is contained in the accumulator.

Standard Prolog systems use an execution mechanism
which always selects the leftmost atom in a goal together with a depth-first search strategy. The depth-first search strategy finds each success branch of the search tree when traversing any finite tree or finite directed acyclic graph. A serious problem arises if the search tree contains an infinite branch. In this case Prolog may fail to find a solution to a goal even if the success branch is finite. An infinite search can be avoided by specifying a maximum search depth. If the solution is at a depth beyond the specified maximum depth then Prolog fails to find the solution.

6.3 Performance measurements

The following Prolog programs were executed on a Sun SPARCclassic workstation using the SWI-Prolog interpreter version 1.9 [24] running under the operating system Solaris 1.1.1 and X-Windows.

- Measurements performed using Example 1:
  In Example 1 the manufacturing of a product is finished when it arrives at place P16. In the 4 executed simulations the amount of tokens in place P16 is used as goal state. In case 1000 products are given as goal state the following query has to be entered:

```
?- time(net(
    [1000, 4, 0, 0, 2000, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1],
    [1000, 4, 0, 0, 2000, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1])).
```

The measured CPU-times are shown in Fig. 11. Including the transition T5, that moves the paletts back to the paletts pool, 8 transitions have to be fired in order to generate one product. This means that in the last case 8000 firing processes were performed in in 36.52 seconds.

- Measurement performed using Example 2:
  The calculation of the new net state that is reached after firing transition T7 took 0.02 seconds.

- Measurement performed using Example 3:
  The calculation of the optimal assembly solution for the 3 ordered products with predicate P1 as FIFO predicate took 2.05 seconds.

It should be noticed that there exist qualitative differences between the selected examples not only due to different net complexities. Whereas in Example 1 and Example 2 the program terminates when the goal state is reached, in Example 3 the search procedure terminates when the optimal solution (minimal time) is computed. On the other hand the introduction of time and the earliest firing rule can be considered as an additional restriction which reduces the search space.

7 Comparison with related work and conclusions

The presented approach concentrates on the transformation of PRT nets into logic programs. In order to allow quantitative evaluations the PRT net models are extended in this approach by assigning to the transitions firing durations which can be fixed or variable. VTT-PRT net models were presented as customer requirements driven formal specifications. The net models are translated automatically into high level, implementation oriented Prolog programs by a transformation procedure that preserves the static and dynamic properties of the net model. This algorithm is valid for most cases in practice, as demonstrated by means of three examples, having different complexity levels.

A transition of a PRT net model can be considered as an activity that depends only on the markings of the places that are connected via arcs with the transition. According to this all conditions and actions related with a transition are contained in a single Prolog unit and are not split like in [1]. Another difference between the approach presented in this paper and that presented in [1] is that the other approach only considers the initial marking and possible firing sequences starting from the initial marking. Intermediate states of the net and the goal state are not considered. Therefore it cannot be computed if a state $M'$ can be reached from an initial state $M$.

There exist already sophisticated tools for simulating the behavior of petri net models. However, their use for software engineering purposes is limited, because their simulation languages are not suitable for implementing the modeled systems.

By combining PRT nets and logic programming a basis for graphical specification, analysis, simulation and implementation of systems is provided. The advantage of a Prolog implementation over an implementation in a conventional language is that it is more "high level" and allows to enter nontrivial queries in a compact way.
Acknowledgment

This work has been supported by Mitsubishi Electric Europe, project Modelling and Implementation of a Flexible Robot Cell.

References


A Appendix

A.1 P/T nets, PrT nets

Definition A.1. A sixuple \( N = (P,T;F,G,W,M_0) \) is a place/transition net (P/T net) iff (if and only if) [21]:

- \( (P,T;F) \) is a net where \( P \) is a non-empty set of places, \( T \) is a non-empty set of transitions and \( F \) is the flow relation.
- \( C : P \rightarrow N^\ast \cup \{\infty\} \) is a capacity function.
- \( W : P \rightarrow N^\ast \) is an arc weight function.
- \( M_0 : P \rightarrow N \) is an initial marking which satisfies for all places \( p \) in \( P \) the condition \( M_0(p) \leq C(p) \).

Definition A.2. Let \( W(x,y) \) denote the weight of arc \( (x,y) \). A transition is enabled:

- if for all \( p \in \ast t \), \( p \) contains at least the number of tokens specified by \( W(p,t) \).
- if for all \( p \in \ast t \), \( M(p) \leq C(p) - W(t,p) \).

An enabled \( t \in T \) may fire.

Definition A.3. Let \( M'(p) \) denote the marking of \( p \) after firing \( t \). \( M'(p) \) is defined as follows (see also [21]):

\[
M'(p) = \begin{cases} 
M(p) + W(t,p), & \text{if } p \in \ast t \setminus t \\
M(p) - W(p,t), & \text{if } p \in \ast t \cap t \\
M(p) - W(t,p), & \text{if } p \in \ast t \cup t \\
M(p), & \text{otherwise}
\end{cases}
\]

A.2 Logic programming

Logic programming differs fundamentally from conventional imperative programming. Logic programs specify the "logical" structure of the problem, i.e., aspects that are relevant to describe the problem rather than defining the steps how to solve the problem. Characteristic for logic programming is that an algorithm is viewed as the composition of two disjoint components: Logic as means of a specification which describes the problem to be solved (declarative knowledge) and control information which describes how the problem is solved (procedural knowledge) [13].

A Prolog program is a set of Horn clauses. Three forms of clauses are distinguished: facts, rules and goals (also called queries). The following notation of a rule is used in general (a fact is a special form of a rule with an empty right part):

\[
C(A_1,\ldots,A_n) :- B_1(A_1,\ldots,A_1),\ldots,B_m(A_m,\ldots,A_1)
\]

with \( C \) being the functor (name) and \( A_i \) being the arguments of the Prolog clause. \( C(A_1,\ldots,A_n) \) is the head or conclusion of the Horn clause,

\[
B_1(A_1,\ldots,A_1),\ldots,B_m(A_m,\ldots,A_m)
\]

is the body and the \( B_i(\ldots) \) are the body goals.

Queries are rules with an empty left part. A query can be preceded by: "Does there exist a substitution of variables in the body goals of the query such that this substituted query is a logical consequence of the program?" A query leading to an answer not being "no" is successful or satisfied, a query leading to the answer "no" is failed, binding.

A predicate is the set of all clauses having the same functor (name) with the same number of arguments, i.e. the same arity. A predicate \( g \) (name) of arity \( n \) is called \( g^n \).