A partial-order-based simulation and validation approach for high-level Petri nets

Jörg Desel, Thomas Freytag
Institut AIFB, University of Karlsruhe, Germany
E-mail: {desel | freytag}@aifb.uni-karlsruhe.de

Andreas Oberweis, Torsten Zimmer
Lehrstuhl für Wirtschaftsinformatik II, University of Frankfurt/Main, Germany
E-mail: {oberweis | zimmer}@wiwi.uni-frankfurt.de

Keywords: Petri nets, partial-order-based simulation, validation, visualization

ABSTRACT

This contribution describes a simulation concept for systems modelled by high-level Petri nets based on partial-order semantics. The main advantage of this approach is its ability to represent and visualize causal dependencies and concurrency explicitly and to give the user a much more intuitive insight into the dynamic behaviour of the modelled system. Additionally, the paper describes how system properties can be specified in a graphical way and how these properties can be checked in a very efficient way by evaluating partially ordered simulation runs.

INTRODUCTION

The topic of this paper is part of the work of the project "*Verification of information systems by evaluating partially ordered Petri net runs (VIP)*" that sponsored by the German Research Society (DFG) [4].

Petri Nets have become a widely accepted formalism for modelling, simulation and analysis of complex systems in a variety of application domains. In particular high-level Petri nets are applied because of their flexible and compact structure. There are two general views at the dynamic behaviour of a net model: The first one - called the *sequential semantics* - is based on the set of *firing sequences* of a net. The second one - called the *causal semantics* - considers the set of partially ordered runs (or *processes*) of a net. Whereas the causal semantics is favorized in parts of Petri net theory, the area of applications still is dominated by the sequential semantic. Usually, simulation tools for Petri nets generate totally ordered sequences of transition occurrences. However, - inspired by several recent results in the domain of system verification (e.g. [6]) - the causal semantic view has become a much more suitable approach for practical applications. It combines strong analytical power with an increased structural efficiency and the ability to visualize and validate the dynamic system behaviour in a much more user-oriented way.

Related papers in the context of the *VIP* project have introduced a simulation approach based on the construction of partially ordered runs [1, 5], have shown how this technique contributes to efficiency in several ways [3] and have sketched algorithms to check certain properties by evaluating these runs [1]. The main focus of this contribution is to give an overview how the partial-order-based simulation concepts can be used to visualize dynamic system behaviour and thus validate certain system properties in a much more intuitive and user-friendly way than this could be done using classical sequential simulation methods.

BASIC DEFINITIONS

Predicate/Transition nets

We suppose the reader to have some understanding of Petri nets, in particular with high level nets as introduced e.g. in [8, 9]. For algorithmical reasons, we restrict the class of *Predicate/Transition nets (Pr/T nets)* to finite nets with finite domains containing no transitions with empty preset or postset. We restrict operations on domains to take place inside guard expressions (i.e. all arc labels consist only of multisets of variables). This is no real restriction of the class of Predicate/Transition nets because every net with arc label operations can be transformed to an equivalent net with guard expressions.

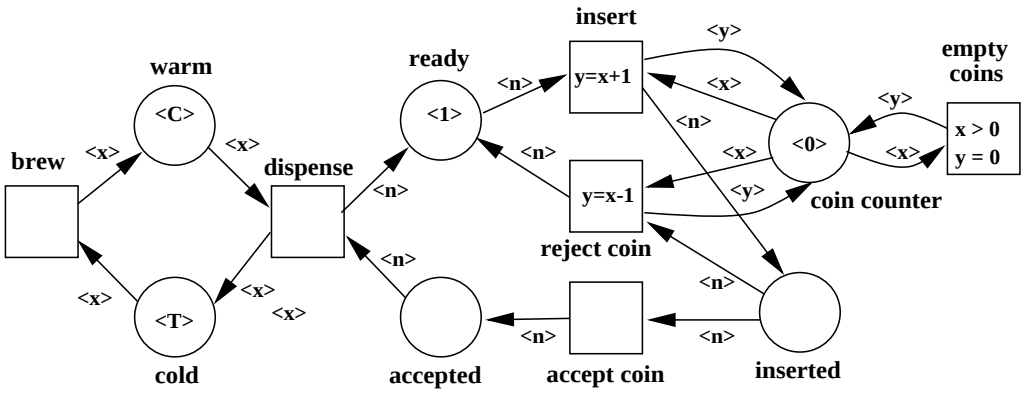

Figure 1: Example for a Predicate/Transition net: A beverage selling machine

An example is given in Figure 1. It models a beverage selling machine that brews a cup of coffee or tea when a coin is inserted. The basic operation loop starts when the transition **insert** occurs moving the token on **ready** to **inserted** (and incrementing the value of **coin counter** by 1). Afterwards the transition **accept coin** can occur moving the token from **inserted** to **accepted**. Another transition **reject coin** is also enabled at this stage modelling the case where the coin is rejected by the machine and the token is moved back to **ready**. As soon as **accepted** holds a token and a beverage has been made available by occurrence of the transition **brew**, the transition **dispense** can occur, giving a beverage[1] to the customer and putting a token on **ready** to signal that a new coin can be inserted. A small subsystem consists of the transition **empty coins**, modelling the operation to remove the coins from the coin storage. The system is initially marked by a token <**C**> on the place **warm** (i.e. a coffee has been brewed), a token <**T**> on the place **cold** (i.e. a tea has not been brewed yet), a token <**1**> on the place **ready** (i.e. the machine is ready for inserting coins) and a token <**0**> on the place **coin counter** (i.e. the coin storage is empty).

Partial orders, causal nets and processes

A Petri net is called a *causal net* if every place has at most one input transition and at most one output transition, every transition has at least one input place and at least one output place and the flow relation has no cycles (i.e. its transitive closure is a partial order[2]). The places of a causal net are called *conditions,*the transitions are called *events,* the underlying partial order is called *causal order.*

[1] Note that this a very simple machine, e.g. it does not allow the customer to choose betwenn tea and coffee - he must take the type of beverage the machine supplies for him

[2] A partial order is a irreflexive, transitive binary relation over a given set (here the set of conditions and events of the causal net)

Shortly, a causal net is an acyclic, place-bordered net with forward and backward unbranched places. The conditions without input transitions are called *minimal elements* and the conditions without output transitions are called *maximal elements* of the causal net.

A process can be described in a straightforward way by a causal net (then called a *process net*):

- Each condition of the causal net represents the existence of a marking tuple (multiset element) on a particular place of the Pr/T net at some stage of the process
- Each event of the causal net represents the occurrence of a transition in the Pr/T net for a particular variable assignment
- Each arc of the causal net represents the flow of marking tuples in the Pr/T net: Whenever a transition consumes a tuple from a place, an arc is drawn from the associated condition to the associated event. And whenever a transition produces a tuple on a place, an arc is drawn from the associated event to the associated condition
- The minimal elements of the causal net are those conditions associated to the initial marking of the Pr/T net

For notational convenience, we label a condition of a process net by the name of the associated place followed by the associated marking tuple put in brackets. For example, a condition named **P1(a,1)** stands for the marking tuple <**a**,**1**> on place **P1**. The events of the process net are labelled by the name of the associated transition followed by the list of variable assignments put in brackets. For example **T1(x=a,y=1)** stands for the transition **T1** occurring for the assignment **x=a** and **y=1**.

Figure 2 shows a process net for the net in Figure 1 (some names are abbreviated for sake of readability). The set of minimal elements consists of the conditions **warm(C)**, **cold(T)**, **coin counter(0)** and **ready(1)**, representing the initial marking of the system net.

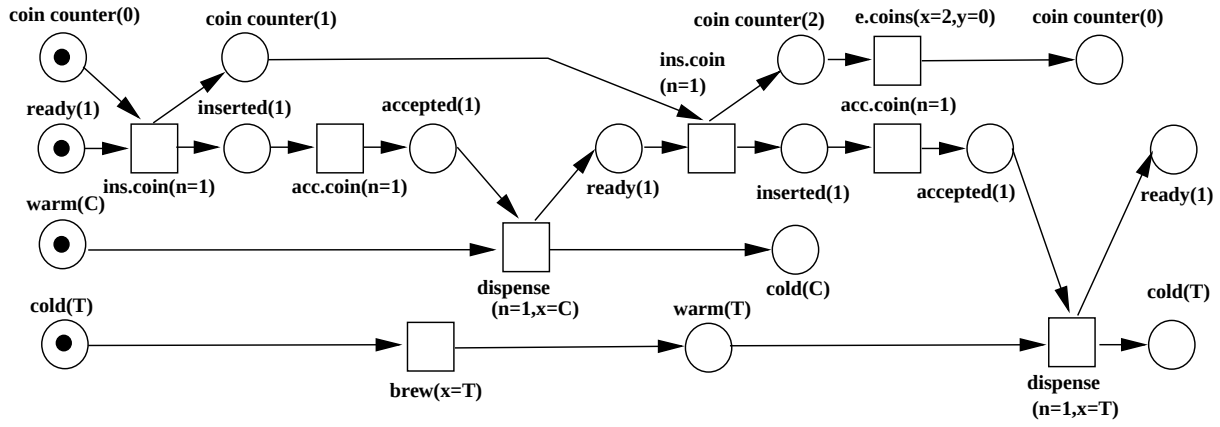

Figure 2: A process net of the net in Figure 1

The process net in Figure 2 is a visualization of the dynamic behaviour of the underlying Petri net system. The upper part of the drawing (the "line" between the condition **coin counter(0)** on the left and the condition **coin counter(0)** on the right) describes what is happening to the coin storage unit of the machine: Initially it is empty, then coins are inserted and eventually removed again. The "line" below shows the behaviour of the machine's "control unit": Coins are inserted, accepted or rejected and the production of beverages is managed from here. The lower two "lines" of the process net represent the "beverage production units": A coffee (tea) is warm, is given out, cold water is brewed again until a new warm coffee (tea) is produced a.s.o.

This representation not only gives a complete overview about the concurrent behaviour of the system, but also preserves a very intuitive view to the system that can easily be understood by a user who has few or even no knowledge about Petri net theory.

In contrast, we consider the representation of the system behaviour using a sequential simulation method. A typical set of occurrence sequences of the net in Figure 1 looks as follows:

(1) i.coin(n=1) - a.coin(n=1) - brew(x=T) - e.cup(n=1,x=C) - i.coin(n=1)...
(2) brew(x=T) - i.coin(n=1) - a.coin(n=1) - e.cup(n=1,x=C) - e.cup(n=1,x=T)...
(3) e.cup(n=1,x=C) - brew(x=T) - i.coin(n=1) - a.coin(n=1)...
(4) ...

Note that all listed occurrence sequences (and in fact even more) are implicitly represented in the process net shown in Figure 2. Apart from their inefficient and non-graphical representation, they are not giving any intuitive and user-friendly view to the system dynamics (such as causal dependencies, concurrency and distributed behaviour).

GRAPHICAL QUERY VALIDATION

We now want to validate some properties of our beverage machine. The processes generated by the partial-order-based simulation can act as a database that can be queried for checking whether certain system properties hold or do not hold.

One important aim of the VIP project is to supply a user-friendly, graphical interface to specify such queries. This idea refers to the well-known concept of *fact transitions* [7] which allows to extend a Petri net by a special type of transitions that have no effect on the net behaviour but rather specify invariant properties that are expected to hold in all reachable states.

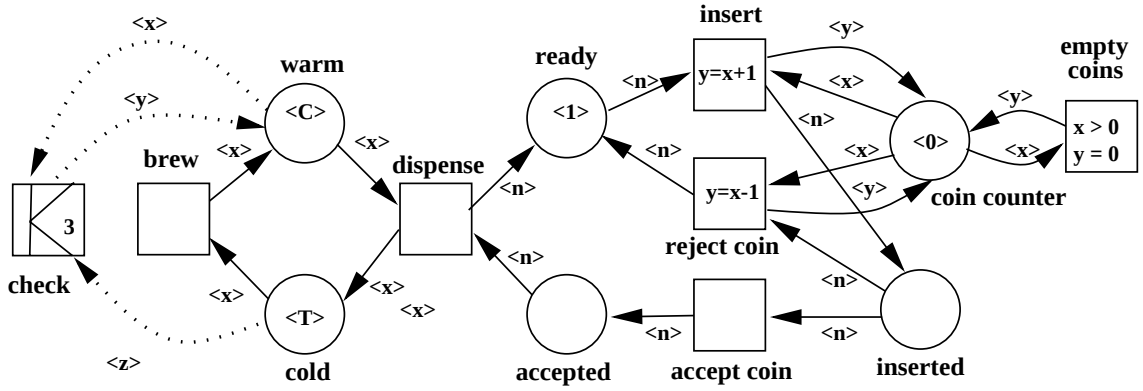

Figure 3: A graphical query to the net in Figure 1

As an example we introduce a fact-like concept called a *causal chain.* Its semantic is as follows: For some set of preconditions that hold in an execution of the Petri net, there exists a causal path (a "chain") of a fixed given length that leads from at least one precondition to a some given postcondition. To illustrate this, we want to check the validity of the following property:

- *Whenever a warm beverage exists, it is dispensed before a new beverage is brewed*

Figure 3 contains a graphical represenation of this query: The causal chain transition **check** (drawn with an inscribed 'K' for the german word "Kausalkette") has been added to the net of Figure 1. It has the input places **cold** and **warm** and the output place **warm**. Arcs are drawn

in dotted style in order to signal their special meaning. The transition is inscribed with the number 3 specifying the required causal path length.

This query can be translated as follows: If both places **cold** and **warm** are marked with a token, we want to know whether there exists an execution where the transition **brew** occurs. In terms of partial order this implies that there exists no process net containing two unordered conditions **cold(...)** and **warm(...)**, either of them being the beginning of a path of length 3 that leads to a condition labelled **warm(...)**[3].

In order to answer this query, it is transformed to a search pattern as shown in Figure 4, being the input for a pattern seach operation in the set of processes. Whenever the search is successful (i.e. at least one process contains such a pattern), we know that the property holds in at least one execution. Once again, this example shows how partial-order semantics contributes to a very intuitive system validation.

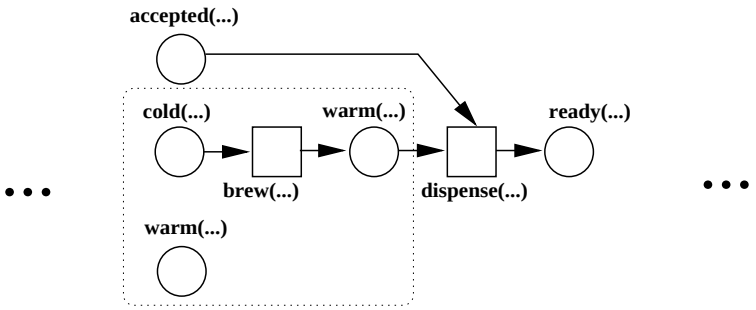

Figure 4: A search pattern to validate the query in Figure 3

The causal chain property is just one example for a class of properties that are in the scope of the VIP project. We state that the partial-order approach is also suitable to handle other property classes in the same comfortable way, in particular progress and reachability properties.

SOFTWARE IMPLEMENTATION

In order to prove the practial applicability of our ideas, we have implemented ***VIPtool,*** a software product that supplies the following functionality (cf. Figure 5):

- An graphical editor for high-level Petri nets
- A graphical query language for specifying system properties
- A simulation and query components based on partial-order semantics
- A process browser with a sophisticated graphical placement policy (based on the Sugiyama algorithm [10])
- A process database for storing and re-using simulation results
- Powerful visualization and validation functions

VIPtool is implemented in PYTHON, a freely-available script language. The graphical interface is based on Tcl/Tk. Supplied hardware platforms will be LINUX, SunOS and Windows 95/NT. A first release of is planned for the summer of 1997, for further information refer to the VIP project's WWW site [2].

[3]The dots inside the brackets denote that the tuple can either be a <T> (tea) or a <C> (coffee)

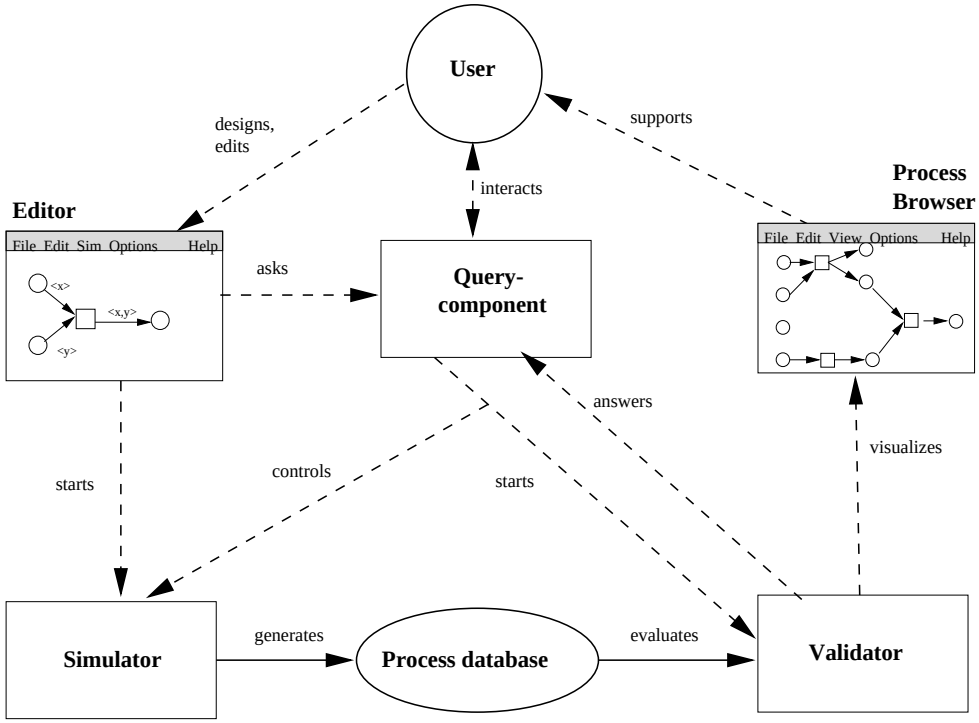

Figure 5: Structure of the software product *VIPtool*

REFERENCES

[1] J. Desel, T. Freytag and A. Oberweis. *Prozesse, Simulation, Eigenschaften netzmodellierter Systeme*. in: Proceedings of Entwurf komplexer Automatisierungssysteme (EKA 97). University of Braunschweig, 1997.

[2] J. Desel, T. Freytag, A. Oberweis and T. Zimmer. *The VIP project homepage*. World Wide Web, http://www.aifb.uni-karlsruhe.de/InfoSys/VIP/overview/vip.html.

[3] J. Desel, T. Freytag and A. Oberweis. *Causal-semantic-based simulation and validation of high-level Petri nets*. in: Proceedings of European Simulation Multiconference (ESM 97). Istanbul, 1997.

[4] J. Desel and A. Oberweis. *Verifikation von Informationssystemen durch Auswertung halbgeordneter Petrinetz-Abläufe*. Technical Report 324, AIFB, Uni Karlsruhe, 1995.

[5] J. Desel, A. Oberweis and T. Zimmer. *Simulation-based analysis of distributed information system behaviour*. 8th European Simulation Symposium ESS 96, Genova, 1996.

[6] J. Esparza. *Model checking using net unfoldings*. Science of Computer Programming, (23):151–195, 1994.

[7] H. Genrich and G. Thieler-Mevissen. *The Calculus of Facts*. Mathematical Foundations of Computer Science, 588–595. Springer-Verlag, 1976.

[8] K. Jensen. *Coloured Petri Nets*. Springer-Verlag, 1992.

[9] W. Reisig. *Petri nets - an Introduction*. Springer-Verlag, 2^{nd} Ed., 1986.

[10] K. Sugiyama, S. Tagawa and M. Toda. *Methods for visual understanding of hierarchical system structures*. IEEE Transaction on Systems, Man and Cybernetics, SMC-11(2): 109–125, 1981.