Modeling with history-dependent Petri nets

Modelling with History-Dependent Petri Nets

Kees van Hee, Alexander Serebrenik, Natalia Sidorova,
Marc Voorhoeve, and Jan Martijn van der Werf

Department of Mathematics and Computer Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{k.m.v.hee, a.serebrenik, n.sidorova,
m.voorhoeve, j.m.e.m.v.d.werf}@tue.nl

Abstract. Most information systems that are driven by process models (e.g., workflow management systems) record events in event logs, also known as transaction logs or audit trails. We consider processes that not only keep track of their history in a log, but also make decisions based on this log. Extending our previous work on history-dependent Petri nets we propose and evaluate a methodology for modelling processes by such nets and show how history-dependent nets can combine modelling comfort with analysability.

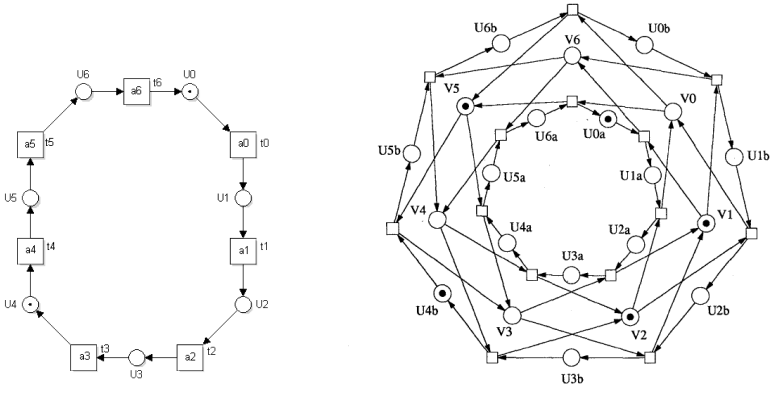
1 Introduction and a Motivating Example

Modern enterprise information systems commonly record information on the ongoing processes as series of events, known as *logs*. Such information might be useful to ensure quality of the processes or of the software, or might even form a legal conformance requirement. Moreover, numerous business processes involve decision making based on previously observed events. For instance, medication should not be ministered if an allergic reaction to a similar medication has been observed in the past.

In classical Petri nets, commonly used to model business processes, the enabling of a transition depends only on the availability of tokens in the input places of the transition. In our previous work we introduced *history-dependent nets* extending the classical model by recording the history of the process and evaluating transition guards with respect to the history [6]. One of the major advantages of history-dependent nets consists in separating the process information from additional constraints imposed to guarantee certain desirable properties of the design. Therefore, the resulting nets are more readable. To illustrate this point consider the following well-known example.

Example 1. The model [5,7] consists of a circular unidirectional railway track of seven sections and two trains a and b. Safety requires that two adjacent sections are never occupied by more than one train. Intuitively, we would like to model the railway track as a set of seven places corresponding to sections, and seven

G. Alonso, P. Dadam, and M. Rosemann (Eds.): BPM 2007, LNCS 4714, pp. 320–327, 2007.



(a) As a history-dependent net

(b) As a classical Petri net [5]

Fig. 1. Simple railway example

transitions corresponding to movements of a train from one section to another. Trains themselves are then represented by tokens (see Figure 1(a)).

Being a classical Petri net, this model, however, does not respect the safety requirement stated. Figure 1(b) presents the original solution as proposed in [5]. For $i = 0, \ldots, 6$ and $z = a, b$, U_{iz} means that section i is occupied by train z, and V_i means that the sections i and $(i+1)$ $mod\,7$ are vacant. Observe that the sole purpose of U_{ib} and V_i is to impose the safety restrictions. We believe that understanding such a model and designing it is a difficult task for a layman.

To ease the modeling task, we use *guards* stating that the transition following U_i fires if U_i has exactly one token, while $U_{(i+1)\ mod\,7}$ and $U_{(i+2)\ mod\,7}$ are empty. It should be noted that U_i has exactly one token if and only if the initial number of tokens at U_i together with the number of firings of the preceding transition exceeds by one the number of firings of the subsequent transition. Similarly, U_i is empty if and only if the initial number of tokens at U_i together with the number of firings of the preceding transition is equal to the number of firings of the subsequent transition. Hence, the guards can be constructed by using the information stored in the history, which is the sequence of firings till the current moment together with the initial marking.

Unlike the original solution, our approach allows to separate the process information (trains move along the sections of a circular rail) from the mechanism used to impose the safety requirements (additional transitions and places in Figure 1(b)).

Clearly, the same history-dependent Petri net can be modelled in many different ways. As one extreme, one can consider expressing all dependencies by means of places and transitions. This approach is illustrated by an overly-complex Petri net on Figure 1(b). As another extreme, one can put the entire information

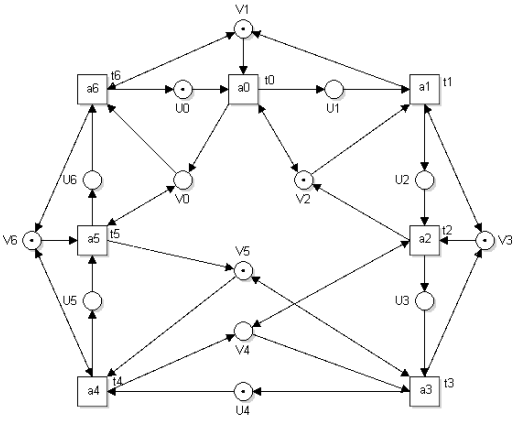


Fig. 2. Railway example: eliminating the guards

in transition guards, i.e., opt for a history-dependent net with just one place, connected to all transitions. We refer to such net as so-called *"flower" net*. Such a net without transition guards can execute transitions in any order and with history-based guards we can restrict this behavior the way we like. The best solution, in our view, is between both extremes: the basic process steps are expressed in the structure of the net, while additional constraints on the execution are imposed by transition guards.

An important aspect of history-dependent nets is the language for expressing transition guards. We developed a language that is powerful enough to express inhibitor arcs, which means that we have a Turing-complete formalism. We considered two subsets of this language, namely the counting formulae and the next-free LTL and showed that by restricting the language to these subsets we can automatically transform history-dependent nets into classical Petri nets (in some cases with inhibitor arcs). Figure 2 shows the net obtained by translating the history-dependent Petri net from Figure 1(a). These nets have more places and transitions than corresponding history-dependent nets and therefore more difficult to read, but they allow for classical analysis methods and for model checking.

In this paper we consider *global history*, which means that any transition may have a guard based on the total history of the net. Access to global history is realistic in many situations, for instance in health care, where all care providers have access to an electronic patient record, or in highly integrated supply chains. The focus of this paper is on the methodology of using history-dependent nets for modelling and analysis of business processes.

The remainder of the paper is organized as follows. In section 2 we describe the methodology for modeling and analysis of history-based nets. In Section 3 we discuss a example from juridical practice. Finally, we review the related work and conclude the paper.

2 Methodology

In this section we describe our approach to modeling with history-dependent nets. We present two different methodologies applicable depending on the project intake: modelling from scratch or re-engineering a data-centered model. A modeling methodology should be seen as a set of guidelines rather than a rigid algorithm.

2.1 Modelling from Scratch

In this subsection we assume that modelling is done from scratch, i.e. a new information system is to be developed. The *first* step in modeling consists in determining the *stages* in the life cycle of the *objects* that play a role in the system. For instance, in Example 1 the objects are trains and the stages are railway tracks. In a hospital care model the objects are patients and the stages can be "blood sample being analysed" or "on medication". Observe that in this case, an object (patient) can be in different stages at the same time: the patient can be X-rayed and at the same time a blood sample can be tested. In general, non-experts should be able to understand what are the objects and what are the stages. In Petri nets the objects are represented by tokens while the stages are modelled as places. It should be noted that *a direct attempt* to model the process as a Petri net will typically result in places representing both process stages and artificial mechanisms needed to express such constructs as choice.

The *second* step aims at the events that cause the change from one stage to another. In Petri nets these events are modeled as transitions. For example, in the patient care process the transition from the X-ray stage to the examination stage may be taken only if the blood test stage has been completed. Upon completing this step one usually has a process model that allows too much behavior, so many occurrence sequences allowed in the model are disallowed in practice.

So the *third step* consists in *restricting* the behaviour of the model constructed so far by means of guards on the existing transitions. These guards dependent solely on events happened in the past, i.e., transition firings, event occurrence time and data involved. For instance, the choice of a medication can depend on an evolution of blood pressure as observed in recent measurements. To ensure correctness of the specified behaviour we often have *global constraints*, such as in the railway case where it is forbidden that two trains are in places with a distance smaller than two. Based on these global constraints the model designer should formulate history-dependent guards restricting firings of individual transitions.

Finally, the *fourth step* aims at assessing correctness of the model, e.g., checking whether the constraints are implied by the guards. To this end we make use of the transformations to classical (inhibitor) Petri nets.

2.2 Modelling from an Existing Data-Centered Model

The four steps of the methodology described in Section 2.1 are not applicable if the development starts from a legacy information system. A legacy information

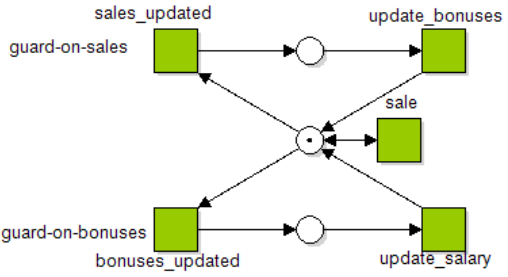


Fig. 3. Active rules as a history-dependent net

system is typically database-centered. Process information is expressed by means of *active rules* [9] that should ensure global constraints [3]. Unfortunately, it is a commonly recognised fact that implicit relations between the rules and unpredictable (non-deterministic) behaviour can jeopardise maintainability of a system. Therefore, we propose an alternative approach, deriving a history based net from an active database.

The *first* step consists in listing all possible basic actions. To illustrate our approach we consider two rules: (1) if the updated sales figure exceed ten units, the bonus of the salesperson is increased; (2) if a salesperson has obtained three bonuses, her salary is increased. In this particular case we have only one basic action, namely, *sell*. We construct the flower net using these basic actions.

The *second* step is constructing a *windmill net* based on the place of the previously constructed flower net. Every vane represents a rule, i.e., consists of a linear Petri net formed by an event and a series of actions (Figure 3). Condition acts as a part of a guard of the transition representing the triggering event. Observe, however, that every event can be handled only once. Therefore, the guard needs to count a number of occurrences of the corresponding action *after* the transition represented by the last action of the rule has been fired for the last time. Therefore, *guard-on-sales* is $\#_{\succ last(update\_bonuses)}\{sale\} > 10$. The guard corresponding to the second rule can be written in a similar way.

3 Example: The "Supply Chain" of Criminal Justice

To illustrate the advantages of history-dependent nets we consider a simplified example of a process of criminal justice. In this process four parties are involved. They form a so-called "supply chain". The participating parties are the police department, the prosecutor's office, the court house and the prison.

At the first stage a person is called "free" and is presumed innocent. If the person is suspected of committing a crime, the police department will try to arrest the suspect, charge him with the crime and either let him go free (with some restrictions like taking his passport), if the suspect is not dangerous and there is no escape risk, or put him in custody, otherwise. Next, the prosecutor's office interrogates the suspect and the witnesses, and, upon the results of the

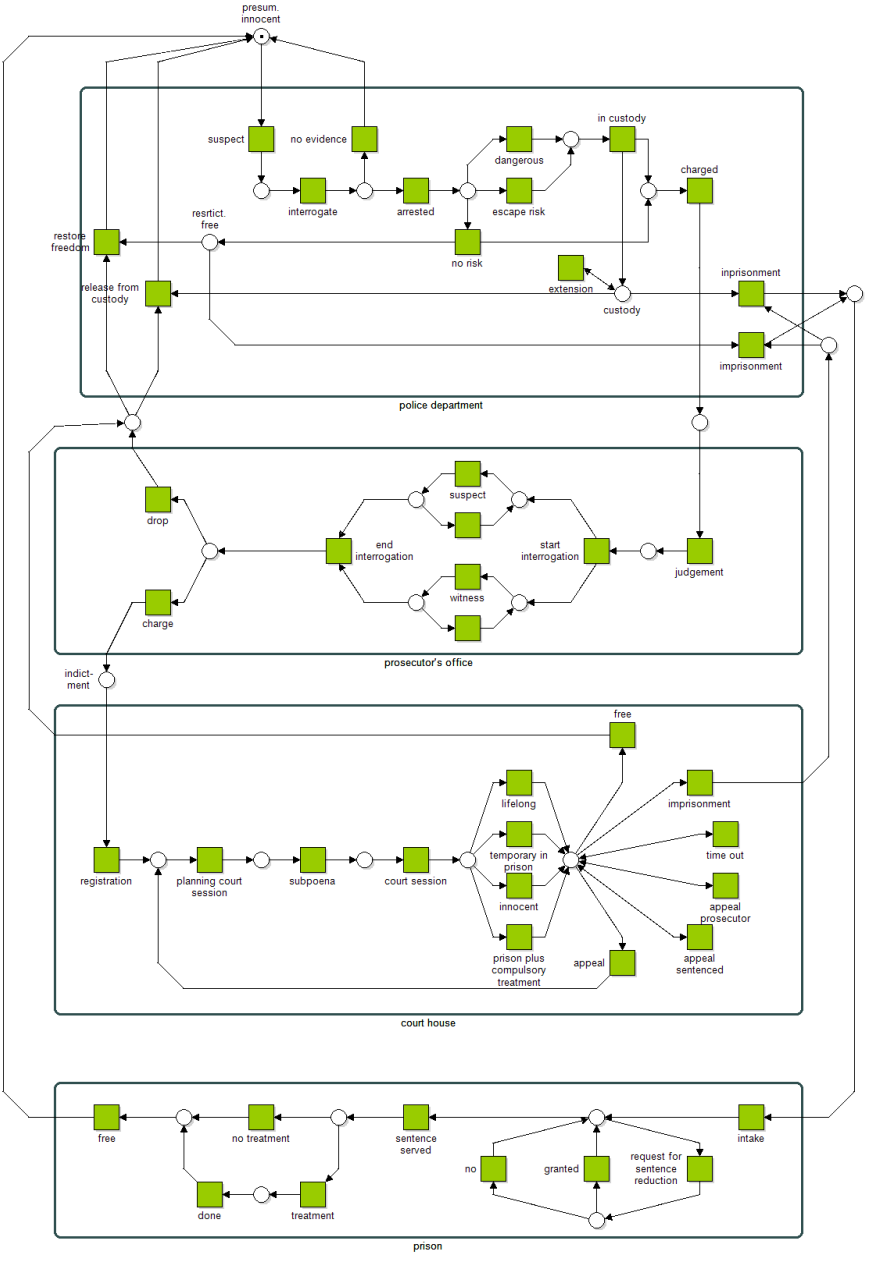


Fig. 4. Criminal justice

interrogation, decides either to drop the charges (in which case the suspect is either released from the custody or his freedom is restored) or to proceed with the charges leading to an indictment.

The process of the court house involves deciding whether the suspect is guilty and what kind of punishment should be carried out. Depending on the court's decision, the suspect or his attorney can decide to submit an appeal. In this case the process is repeated. If the suspect has been convicted and no appeal has been submitted, he is imprisoned. During his stay at the prison the convict can apply for a sentence reduction. Depending on the court decision, the person might need to undergo a special treatment upon servicing the sentence. If no such treatment is needed or the treatment has been done, the person is freed and the entire process restarts.

Similarly to Example 1 we make a clear separation between the four basic parts of the process presented in Figure 4 and additional constraints existing between the steps of each part and between the parts. To model these and similar constraints we use the guards, which are expressions over the history. The following table illustrates a number of constraints and their formalisation as formulae of our history logic [6]. (Note that $\lambda(e)$ denotes the label of event e from the history log.)

Constraint	*Transition*	*Guard*
Both the prosecutor and the sentenced may submit an appeal once after the first court session and once after the second court session. Appeal should be submitted before the time out.	*appeal*	$\#_{\succ last(\{registration\})}\{court\ session\} \leq 2$ $\wedge \forall_{\succ last(\{court\ session\})} u :$ $\neg(\lambda(u) \in \{time\ out, appeal\})$ $\wedge \exists_{\succ last(\{court\ session\})} v :$ $\lambda(v) \in \{appeal\ sentenced,$ $appeal\ persecutor\}$
A prisoner may apply only three times for reduction of punishment and if the application is granted once, no further requests are allowed. If a lifelong sentence has been proclaimed no requests are possible.	*request for sentence reduction*	$\#_{\succ last(\{court\ session\})}\{lifelong\} = 0$ $\wedge \#_{\succ last(\{court\ session\})}\{request$ $for\ sentence\ reduction\} \leq 2$ $\wedge \#_{\succ last(\{court\ session\})}\{granted\} = 0$
If somebody has a lifelong sentence the *sentence served* transition may not fire.	*sentence served*	$\#_{\succ last(\{court\ session\})}\{lifelong\} = 0$

4 Conclusions and Related Work

In this paper we have presented a modelling methodology based on history-dependent nets. We have seen that history-dependent nets improve the modelling comfort by allowing a clear separation between the graphically represented

process information on the one hand, and the logically represented information on the additional constraints on the other. Moreover, in many practical cases history-dependent nets can be automatically translated to bisimilar classical Petri nets, which accounts for their analysability and verifiability.

Histories and related notions such as event systems [10] and pomsets [4,2] have been used in the past to provide causality-preserving semantics for Petri nets. Baldan *et al.* [1] use two different notions of history. Unlike our approach, none of these works aims at restricting the firings by means of history-dependent guards. *History-dependent automata* [8] extend states and transitions of an automaton with sets of local names: each transition can refer to the names associated to its source state but can also generate new names which can then appear in the destination state. This notion of history implies that one cannot refer to firings of other transitions but by means of shared names. We believe that the ability to express dependencies on previous firings explicitly is the principal advantage of our approach.

References

1. Baldan, P., Busi, N., Corradini, A., Pinna, G.M.: Domain and event structure semantics for Petri nets with read and inhibitor arcs. Theoretical Computer Science 323(1-3), 129–189 (2004)
2. Best, E., Devillers, R.R.: Sequential and concurrent behaviour in petri net theory. Theoretical Computer Science 55(1), 87–136 (1987)
3. Ceri, S., Widom, J.: Deriving production rules for constraint maintainance. In: 16th International Conference on Very Large Data Bases, August 13-16, 1990, Brisbane, Queensland, Australia, pp. 566–577 (1990)
4. Goltz, U., Reisig, W.: The non-sequential behavior of Petri nets. Information and Control 57(2/3), 125–147 (1983)
5. Hartmann, J.G.: Predicate/transition nets. In: Advances in Petri Nets, pp. 207–247 (1986)
6. van Hee, K.M., Serebrenik, A., Sidorova, N., van der Aalst, W.M.P.: History-dependent Petri nets. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007, Springer, Heidelberg (2007)
7. Junttila, T.A.: New canonical representative marking algorithms for place/transition-nets. In: Cortadella, J., Reisig, W. (eds.) ICATPN 2004. LNCS, vol. 3099, pp. 258–277. Springer, Heidelberg (2004)
8. Montanari, U., Pistore, M.: History-dependent automata: An introduction. In: SFM, pp. 1–28 (2005)
9. Widom, J., Ceri, S. (eds.): Active Database Systems: Triggers and Rules For Advanced Database Processing. Morgan Kaufmann, San Francisco (1996)
10. Winskel, G.: Event structures. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) Advances in Petri Nets. LNCS, vol. 255, pp. 325–392. Springer, Heidelberg (1987)