A Logic of Actions to Specify and Verify Process Requirements

Carlo Simon

Institute for Management, University Koblenz-Landau
simon@uni-koblenz.de

Abstract

In this paper, Timestamp Petri nets – a special kind of timed Petri nets - and a Logic of Actions are used to specify process requirements. Petri net implementations of such specifications are used for a better visualization and to prove whether a given realization is sound and complete with respect to a given specification. The theory is applied to the specification of workflow processes. An example shows how to prove that a given workflow fulfills time constraints required by the management of a company.

Keywords:

(Timestamp) Petri nets, Logic of Actions, Workflow Management

Introduction

A *Software Requirements Specification* (SRS), as defined in (Davis, 1993), "is a document containing a complete description of *what* software will do without describing *how* it will do it". An SRS is the result of an early phase within a software project and builds the base for the further development. It is a collection of user needs and technical restrictions that must be considered while the software is being built.

Because of its central importance, many academic and practical work concentrates on the following problems:

- Which formal techniques and languages exist to define requirements such that they can be transformed into a formal specification and later on into an implementation?
- If a given formal requirements definition cannot be transformed into an implementation automatically, is it at least possible to prove whether a realized implementation fulfills this requirement definition?

In the following, we concentrate on special requirement definitions - specifications of processes. We use a Logic of Actions proposed in (Simon, 2001a) to formally define process specifications and realizations. Petri net representations of these definitions are used to illustrate the represented processes and for proving given realizations against given specifications within this theory. They will be explained throughout this paper. An example from Workflow Management Systems will be used to illustrate the application of this logic within a software development process.

Timestamp Petri Nets

Timestamp Petri Nets (Hanisch, Lautenbach, Simon, and Thieme, 1998) are a special kind of Petri net (Peterson, 1981, Reisig, 1985) allowing the representation time information. Each token of a timestamp net carries a *timestamp* designating the moment the token was put on its

place. *Intervals* at the incoming edges of transitions specify their *permeability* with respect to the marking of the adjacent place. If all pre-places of a transition are marked and all its incoming edges are permeable simultaneously, then this transition is able to fire.

Figure 1 shows a timestamp net consisting of five places *p1*, …, *p5* and three transitions *a*, *b*, and *c*. Place *p1* is marked with a token with timestamp three, the token on *p2* has a timestamp seven and the token on *p3* has a timestamp of one. Therefore, the token on *p2* is the youngest and that on *p3* is the eldest shown in the net. All other places of the net are unmarked.

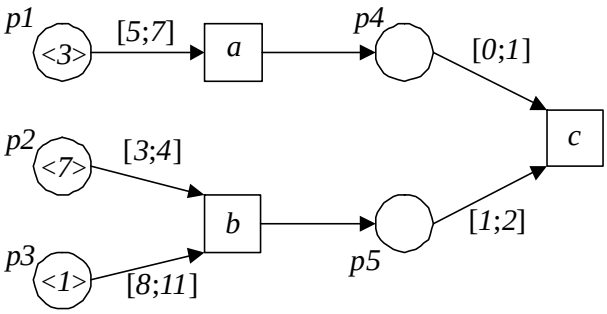

Figure 1: An example of a Timestamp Petri Net

The edge from place *p1* to transition *a* is permeable from eight to ten, i.e. in interval [*5+3*; *7+3*]. Within this interval, transition *a* is able to fire any moment. If doing so, the resulting token on place *p4* gets a timestamp with value eight to ten dependent on the moment transition *a* actually fires.

The edge from *p2* to *b* is permeable from ten to eleven and that from *p3* to *b* from nine to twelve. Since both edges are only permeable simultaneously from ten to eleven, transition *b* is only able to fire within this interval. Afterwards, the resulting token on *p5* will carry a token with a timestamp between ten and eleven.

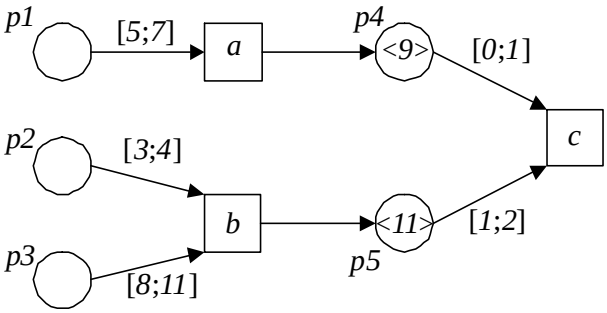

Figure 2: Transition c *is timewise stuck*

Figure 2 shows the follower marking from the situation described above. If transition *a* fires at moment nine and transition *b* fires at moment eleven. Then the edge from place *p4* to transition *c* is permeable from nine to ten while the edge from *p5* to *c* is permeable from twelve to thirteen. As a consequence, transition *c* is not able to fire, since its incoming edges are not permeable simultaneously at any moment although all pre-places of *c* are sufficiently marked. In this situation, we call transition *c timewise stuck*. We use such situations to specify undesired behavior.

Timewise stuckness always occurs when one incoming edge of a transition looses its permeability before another become permeable at all. Hanisch, Lautenbach, Simon, and Thieme (1998) have presented an analyzing technique that can be used to determine whether certain transitions in a timestamp net might get timewise stuck. The same analyzing technique

can be used to determine time parameters within such a net such that certain transitions will never get timewise stuck.

Until now, applications of timestamp nets have been rather technical (Simon, 2001a, Simon, 2001b), but as we will demonstrate later in this paper, non-technical applications can be found as well.

A (Timed) Logic of Actions

In Simon, (2001a), a *Logic of Actions (LA)* is proposed. *Process* is the central term in LA. Processes consist of *actions* which might occur or are forbidden and which are ordered in sequence or might occur coincidentally. Timestamps assigned to actions are used to determine their moment of occurrence. The extension of LA by time is called *Timestamp Logic of Actions (TiLA)*.

Modules, the formulas of LA and TiLA, are used to specify process sets. Before (<), after (>), coincident (=), forbid (¬), and (∧), xor (+), and iteration operators are used to combine sub modules to complex ones.

$$[_{[3;4]}\, a < [_{[6;7]}\, b +_{[5;8]}\, c]]$$

is an example for a TiLA module. It specifies processes where action *a* occurs *3* to *4* time units after process start. Afterwards, either action *b* occurs after *6* to *7* time units or *c* occurs after *5* to *8* units of time and the processes end. A Petri net implementation of this module is shown in figure 3.

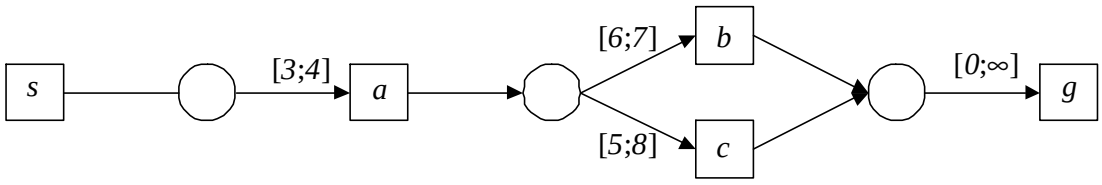

Figure 3: Petri net implementation of $[_{[3;4]}\, a < [_{[6;7]}\, b +_{[5;8]}\, c]]$

In a Petri net *implementation* of a module, the same processes can be realized as are specified by the module. A *process* in such a net is a firing sequence reproducing the empty initial marking and in which the start transition *s* and the goal transition *g*, used to specify the beginning and the end of each process, occur exactly once. Actions are represented by equally named transitions. The symbol ∞ designates that the respective edge never looses its permeability after it has been marked as long as the token stays on the adjacent place.

Petri nets are not only a means to visualize TiLA modules but rather are used to simplify proving within this theory. The aim of *verification* within TiLA is to prove whether a given *realization* of processes fulfills a given process *specification*. A realization can be proven to be *sound* and *complete* with respect to a specification.

Definition: Fulfill

Let *M1*, *M2* be TiLA modules over a set of actions A.

$$M1 \text{ fulfills } M2\ (M1 \rightarrow\!\!\!\!\rightarrow M2) \text{ iff } P[\complement_{M2}(M1)] \subseteq P[\complement_{M1}(M2)]$$

Within this definition, the operator $\complement$ is used to mutually complete the modules in order to make them comparable. In all processes of completed modules each action of this module

either occurs or is forbidden. *M1* fulfills *M2*, if the set of completed processes realized by *M1* is a subset of those specified by *M2*.

A realization is called *sound* if it only realizes specified processes. In terms of our logic, it fulfills its specification. If all specified processes are realized then the realization is *complete*.

Definition: Sound, Complete

Let *S* be a specification given as a TiLA module over a set of actions A, and *R* a realization also given as a TiLA module over the same set of actions A.:

- *R* is sound with respect to *S* iff $R \rightarrow\!\!\!\rightarrow S$
- *R* is complete with respect to *S* iff $S \rightarrow\!\!\!\rightarrow R$

This definition of sound and complete is based on counting the processes specified by modules. However, in the case of TiLA modules the number of specified processes is infinitely large. Therefore, we will consider an alternative approach to proving in the following section.

Proving within the Logic of Actions

Instead of comparing the process sets of a specification *S* and a realization *R*, we do verification by conjoining *S* and *R* and determining whether this rules out processes specified by *R*. In this case, *R* defines processes not specified by *S* and therefore *R* does not fulfill *S*.

Theorem 1

Let *M1*, *M2* be TiLA modules over a set of actions A.

$$M1 \rightarrow\!\!\!\rightarrow M2 \Leftrightarrow P[M1 \wedge M2] = P[\complement_{M2}(M1)]$$

We use theorem 2 instead, if undesirable behavior is specified instead of desired behavior, i.e. if $\neg S$ and not S is given.

Theorem 2

Let *M1*, *M2* be TiLA modules over a set of actions A.

$$M1 \rightarrow\!\!\!\rightarrow M2 \Leftrightarrow |P[M1 \wedge \neg M2]| = 0$$

Both theorems have been proven in (Simon, 2001a). We use theorem 1 for *direct proving* and theorem 2 for *indirect proving*.

For given Petri net implementations of modules *M1* and *M2*, we can realize $M1 \wedge M2$ by *joining* ($\times$) the completed implementations $\mathsf{N}(\complement_{M2}(M1))$ and $\mathsf{N}(\complement_{M1}(M2))$. Joining identifies equally named transitions of both participating nets (including *s* and *g*) and fuses them. All places of the participating nets also occur in the result of the join except those which cause redundant structures. Additionally, places are added to prevent transitions representing a certain action and others representing its prohibition from occurring both in processes.

The effect of joining Petri net implementations of modules is that the resulting net realizes only such processes specified by each participating module or its Petri net implementation, respectively. The completion step is responsible for making the participated processes comparable. This observation leads to the following theorem that is central in (Simon, 2001a).

Theorem 3

Let *M1*, *M2* be TiLA modules over a set of actions A, N ($C_{M2}(M1)$) and N ($C_{M1}(M2)$) Petri net implementations of their completions.

$$N(C_{M2}(M1)) \times N(C_{M1}(M2)) \text{ implements } M1 \wedge M2$$

As an example, let us consider a specification

$$S = [_{[3;4]}\, a < [_{[6;7]}\, b +_{[5;8]}\, c]]$$

and a realization

$$R = [[_{[4;4]}\, a <_{[6;7]}\, c] \wedge_{[0;\infty]} \neg b].$$

Specification *S* describes processes where *a* occurs three to four units of time after process start. Afterwards, either *b* occurs six to seven units of time later, or *c* occurs five to eight units of time later. Realization *R* describes processes where *a* occurs exactly after four units of time after process start. Six to seven units of time later, *c* occurs. In addition, *b* is forbidden.

Figure 3 shows an implementation of *S* and figure 4 shows an implementation of *R*.

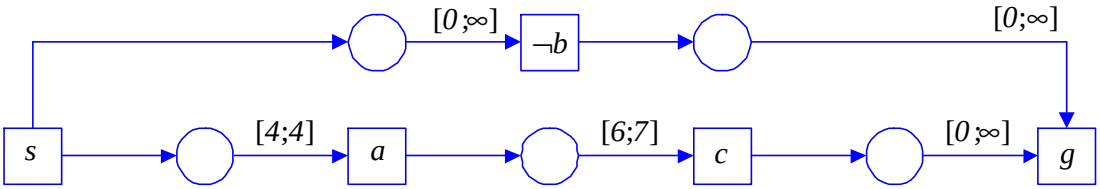

Figure 4: Implementation of R

For proving whether *R* fulfills *S*, we first have to mutually complete their implementations. Figure 5 shows this step with respect to the implementation of *R*. Although the result of this completion looks quite complex this is only a problem of visualization. The operation itself is linear dependent on the number of transitions in the net.

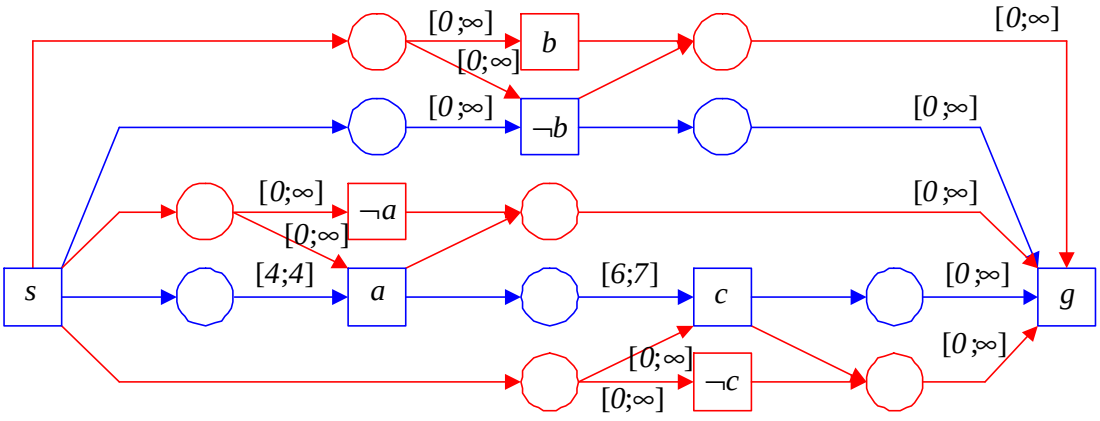

Figure 5: Implementation of C_S(R)

As we mentioned above, completion is responsible for making the process sets of modules comparable. Therefore, this completion step, as demonstrated above, is also important on the Petri net level (cf. Simon, 2002). However, in Petri net implementations we can make an observation that simplifies further calculations: Processes of such a Petri net implementation are firing sequences reproducing the empty initial marking. Thus, only such firing sequences can be realized which are also realizable in the same Petri net without time information. In such a Petri net without time information, such firing sequences reproducing the empty initial marking must be covered by T-invariants. Therefore, each transition that is not included in any T-invariant cannot occur in any realizable process of such a Petri net implementation. In other words: the corresponding action does not occur or is forbidden in any process of the implemented completed module.

If we consider the Petri net in figure 5 without time information, we only find one T-invariant covering transitions *s*, *a*, ¬*b*, *c*, and *g*. No one of the transitions added throughout the completion step is participated in any process. Therefore, we can immediately role back our completion step and can conclude that in our realization already each specified process must have been complete.

Completing the Petri net implementation of the specification *S* and reducing this result with the aid of the T-invariant method described above results in the implementation shown in figure 6.

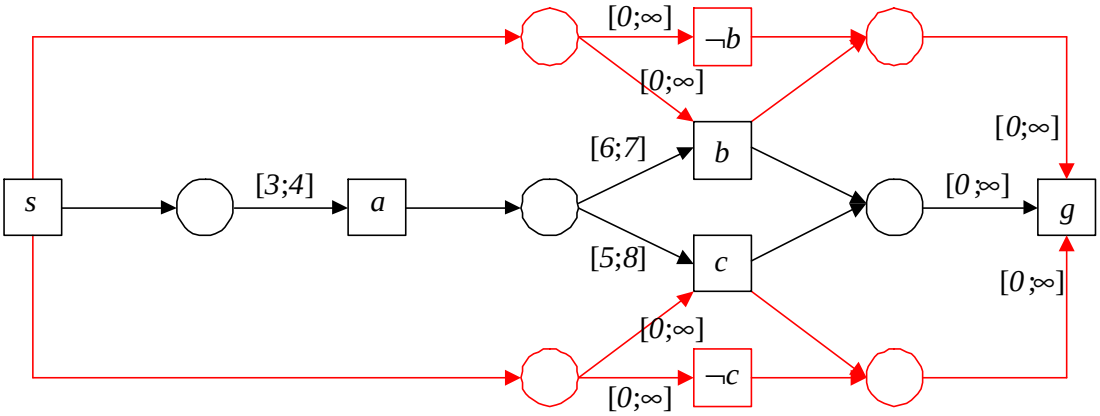

Figure 6: Reduced implementation of C_R(S)

Figure 7 shows the result of joining these intermediary results.

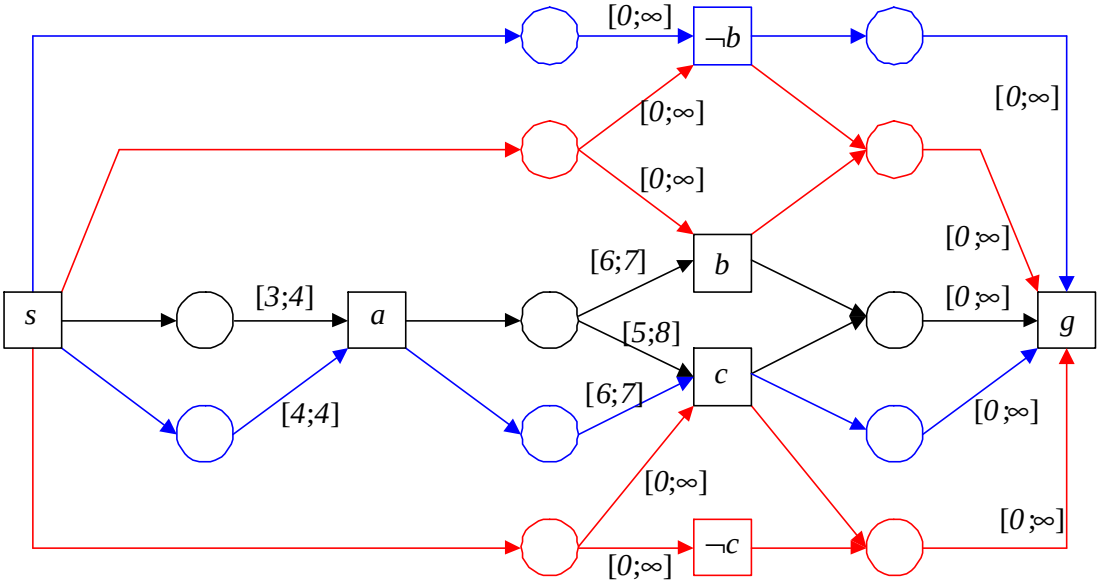

Figure 7: Join of completed specification and realization

Comparable to our reductions of the completed modules, we can also simplify this result using the T-invariant method. This time, we can rule out transitions $\neg a$, b, and $\neg c$ from occurring in any process. This leads to the Petri net implementation of $S \wedge R$ as shown in figure 8.

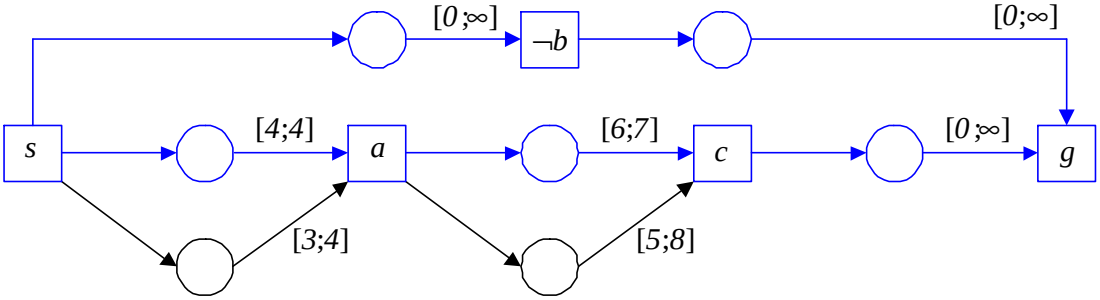

Figure 8: Reduction of the join result with the aid of T-invariants

In all firing sequences beginning with start transition s, both pre-places of transition a get marked at the same moment. Three to four time units later the first incoming edge of a is permeable. The second one is permeable exactly after four time units. Since transition a can only fire if all its incoming edges are permeable simultaneously, the permeability of both edges results from the intersection of interval $[3;4]$ and $[4;4]$. As a result, we get the net shown in figure 9.

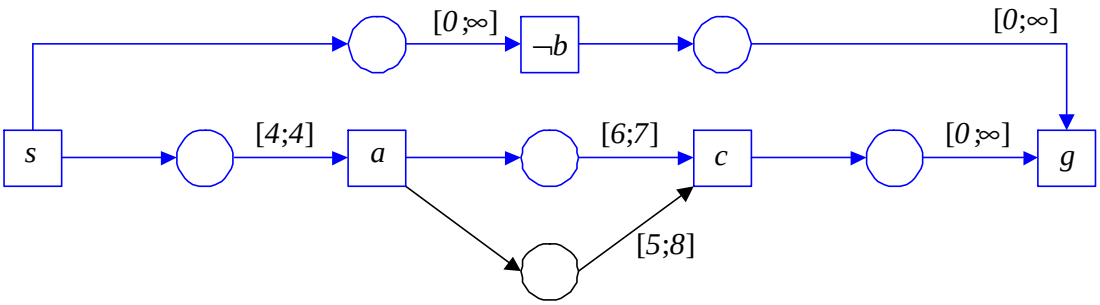

Figure 9: First reduction based on time considerations

In the same way we can proceed with the incoming edges of transition c. The result is the Petri net implementation of our realization R as shown in figure 4. With the aid of theorems 1 and 3 we conclude that the realization fulfills the specification.

In the following sections, we show an application of this theory to the field of Workflow Management Systems.

Workflow Management Systems

The development of *Workflow Management Systems* (WfMC, 1996) arose with *Electronic Document Management Systems*. Offices appeared to be drowning in paper. However, merely changing a paper representation of a business issue into an electronic representation was no solution (Koulopulos, Frappaolo, 1995). The main benefit of Electronic Document Management Systems arises from integrating them into a Workflow Management System identifying the documents needed for conducting a certain task and retrieving them from document archive.

The expression *workflow* is used to describe human work and the basic terms on this area are defined by the *Workflow Management Coalition (WfMC)*. A workflow is "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules". A Workflow Management System "defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications" (WfMC, 1996).

These fundamental definitions by the WfMC highlight the most important parts of workflow systems from the point of view of automation:

- *Workflow participants* who perform the work. The term workflow participant "is normally applied to a human resource but it could conceptually include machine based resources such as an intelligent agent" (WfMC, 1996).

  An *organizational role* specifies a group of employees "exhibiting a specific set of attributes, qualifications and/or skills" (WfMC, 1996). Instead of "workflow participant" the term "role player" is used. This highlights the fact that when a workflow is planned one does not think of individuals, but of tasks and of positions in an organizational structure which are implemented to fulfill this task. The assignment of a certain task to a specific individual is done while a process instance is executed.

- *Activities* are definitions of the elementary pieces of work of which a workflow is composed. The structure of a workflow specifies sequences, alternatives, and parallel occurrences of activities.

  An activity can be performed manually (by humans) or automatically (by the Workflow Management System or by elementary tools for business automation).

  A *process instance* is a workflow in execution, while an *activity instance* is an activity in execution.

The diagram in figure 10 illustrates the relationships between these elementary terms according to the WfMC (WfMC, 1996).

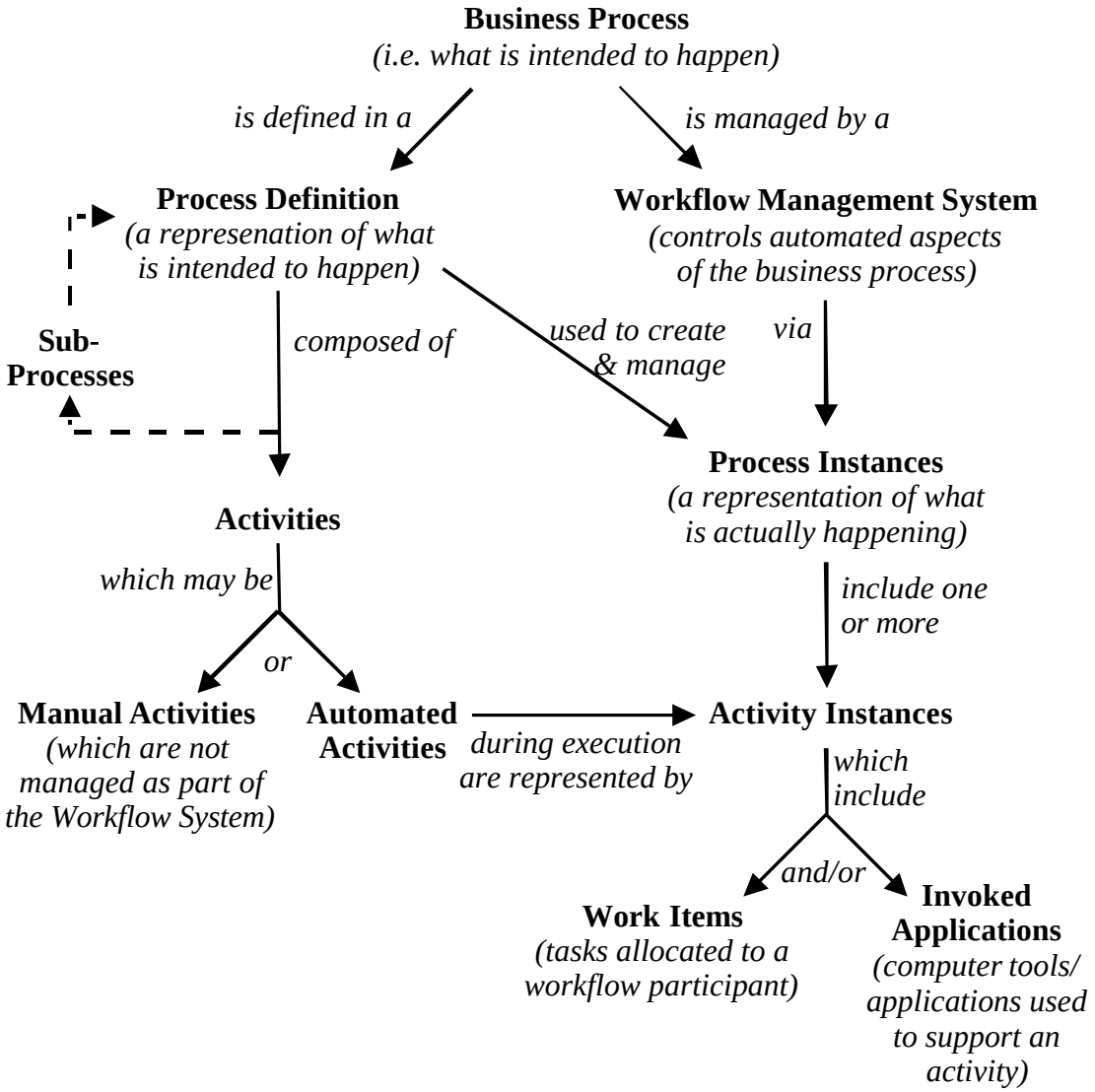

Figure 10: Relationships between basic workflow terminologies

Obviously, processes are central in the definition of Workflow Management Systems. They are detailed in nature because they cover both, the usual business process as well as exceptions from this process in the case of faults. However, a first specification of such a business process might come along without such details. In the following section we consider the question how to prove whether a workflow definition fulfills such a previously made specification.

Applying TiLA to Workflow Management System

In the following, we use TiLA to answer the question: *Is every process of a complex realization conforming a given specification?*

To demonstrate this, we consider an exemplary workflow described in (Aalst, Hee, 2002). They use Petri nets to specify workflow models (Aalst, 1998). As a use case, they consider an insurance company. Figure 11 shows the process "handle complaint". Numbers in transitions are used to indicate the average processing time per task. We leave out assessment cycles, because in the following we want to concentrate on a typical process.

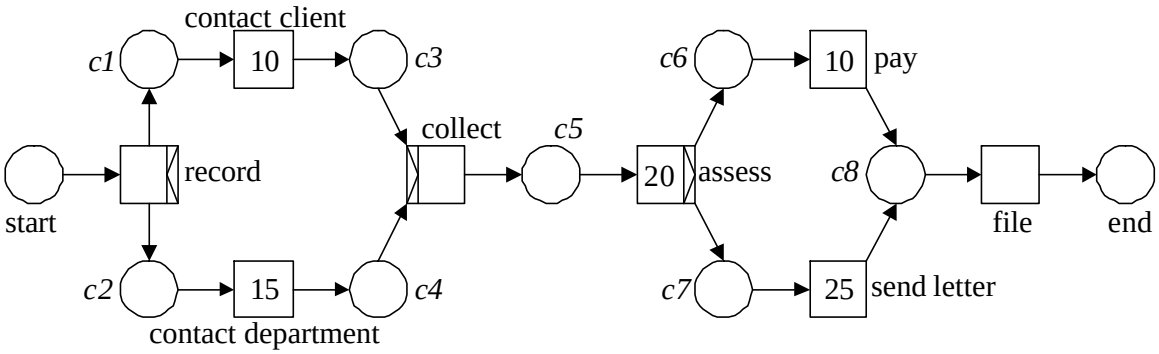

Figure 11: Process "handle complaint"

We want to introduce the following abbreviations: $r \cong$ record, $cc \cong$ contact client, $cd \cong$ contact department, $c \cong$ collect, $a \cong$ assess, $p \cong$ pay, $sl \cong$ send letter, and $f \cong$ file. The following module describes the workflow shown in figure 11 where the time information is interpreted as the moment the respective task ends:

$$[_{[0]}\, r < [_{[0]}\, cc \wedge {}_{[0]}\, cd] < {}_{[0]}\, c < {}_{[0]}\, a < [_{[0]}\, p + {}_{[0]}\, sl] < {}_{[0]}\, f]$$

We abbreviate intervals specifying a single moment like [*15*;*15*] by [*15*]. Figure 12 shows a Petri net implementation of this module.

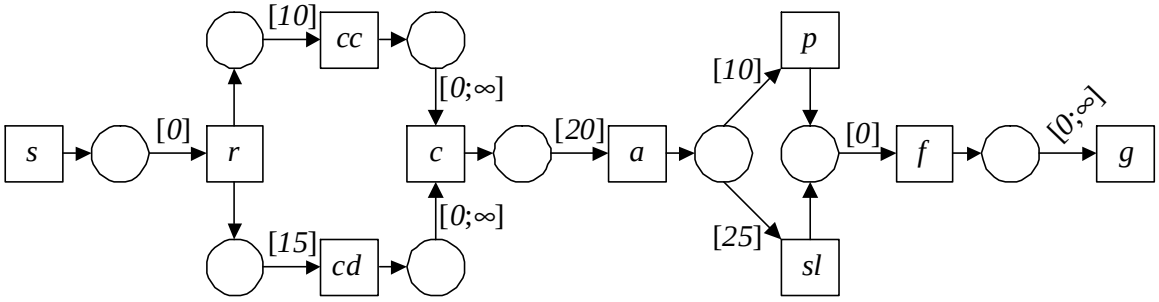

Figure 12: Petri net implementation of "handle complaint"

As one can imagine, the module only specifies a very simple workflow. However, for a manager of an insurance company this already might be too complex. S/he might mostly be interested in the maximal duration of the entire process. A typical question s/he could ask for is: Does it take no longer than 55 units of time after process start until all process data and all used documents are filed?

In the Logic of Action, the formulation of this question is rather simple. $[_{[55]}\, f]$ is a specification of the question. Figure 13 shows the implementation of this module.

Figure 13: Petri net implementation of $[_{[55]}\, f]$

For verifying whether our business process fulfills this specification, we join the implementations of both modules. Figure 14 shows the result of this join.

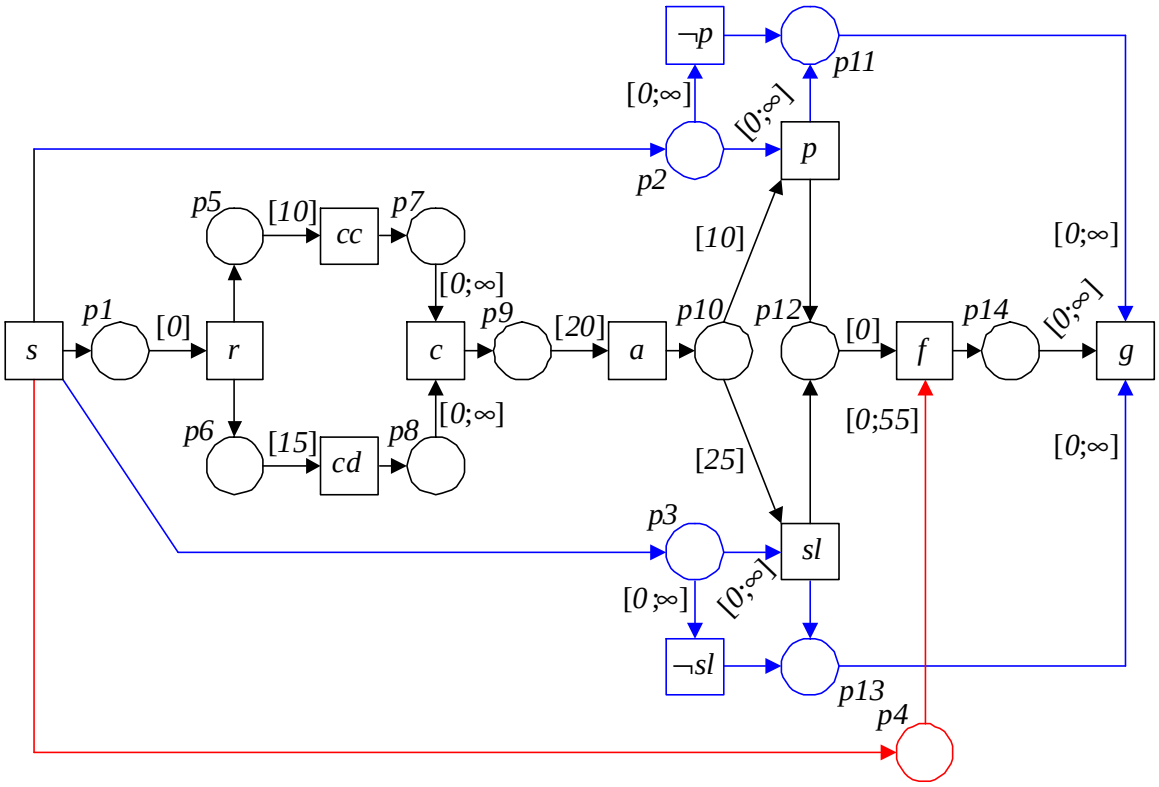

Figure 14: Join of the workflow and its specification

Now we have to examine whether the net in figure 14 represents the same processes as the net in figure 12. If it represents less, this can only be caused by transition *f* getting timewise stuck. We use symbolic analysis to verify this.

Firing transition *s* produces tokens with timestamp α on *p1*, *p2*, *p3*, and *p4*. After firing transition *r*, places *p2*, *p3*, *p4*, *p5*, and *p6* are marked. The tokens on *p2*, *p3*, and *p4* have timestamp α, those on *p5* and *p6* have timestamp $\beta = \alpha + 0 = \alpha$. By firing *cc* and *cd*, places *p5* and *p6* get unmarked, *p7* gets a token with timestamp $\gamma = \beta + 10 = \alpha + 10$, and *p8* gets a token with timestamp $\delta = \beta + 15 = \alpha + 15$. The tokens on places *p2*, *p3*, and *p4* still have timestamp α. Assuming transition *c* fires as soon as possible, *p7* and *p8* get unmarked and *p9* gets a token with timestamp $\varepsilon = min\{\gamma, \delta\} = \alpha + 15$. Firing transition *a* causes a marking where *p10* has a token with timestamp $\zeta = \varepsilon + 20 = \alpha + 35$, and where *p2*, *p3*, and *p4* are still marked by tokens with timestamp α. Now, we have the possibility either to fire transition *p* or to fire transition *sl*.

- Firing transition *p* marks *p11* and *p12* by tokens with timestamp $\theta = \zeta + 10 = \alpha + 45$. Under this situation, transition *f* does not get timewise stuck and can fire at moment θ. Independent from the moment transition ¬*sl* fires, we can finish the process afterwards by firing *g*.
- Firing transition *sl* marks *p12* and *p13* by tokens with timestamp $\lambda = \zeta + 25 = \alpha + 60$. In the same moment, the edge from *p12* to *f* is permeable. However, the other incoming edge of *f* (from place *p4* to *f*) has already lost its permeability. Therefore, transition *f* is timewise stuck and the process cannot be finished.

Consequently, the realization of our business process does not fulfill the manager's specification. Moreover, we observe that we have to optimize our business process such that it

is *5* units of time faster. This can be achieved by optimizing action *contact department*, *assess*, or *send letter*.

Conclusion

In this paper, we have used a Logic of Actions to specify process requirements. Petri net implementations of the formulas of our logic are used for visualization and for proving. As an application, we chose the verification of workflows. We demonstrated our approach with the aid of an example.

Our models and proving techniques are rather formal. As a consequence, they allow precise descriptions of the systems under examination. However, especially in a business environment these descriptions must be substituted by less formal ones, i.e. our mathematical methodology must be hidden from a possible user. Our future work will focus on this problem.

References

Davis, A. M. (1993): *Software Requirements - Revision*, Prentice Hall, Englewood Cliffs

Hanisch, H.-M., Lautenbach, K., Simon, C., and Thieme, J. (1998): *Timestamp Nets in Technical Applications*, In *IEEE International Workshop on Discrete Event Systems*, San Diego, CA; USA

Koulopulos, T. M. and Frappaolo, C (1995): *Electronic Document Management Systems – A Protable Consulting*, McGraw-Hill, New York

Peterson, J. L. (1981): *Petri Net Theory and the Modeling of Systems*, Englewood Cliffs: Prentice Hall

Reisig, W. (1985): *Petri Nets: An Introduction*, volume 4 of EATCS Monographs in Theoretical Computer Science, Springer-Verlag

Simon, C. (2001a): *A Logic of Actions and Its Application to the Development of Programmable Controllers*, Verlag Fölbach, Koblenz, Germany

Simon, C. (2001b): *Developing Software Controllers with Petri Nets and a Logic of Actions*, in *IEEE International Conference on Robotics and Automation*, ICRA 2001, Seoul, Korea

van der Aalst, W., van Hee, K. (2002): *Workflow Management – Models, Methods, and Systems*, The MIT Press, Cambridge, Massachusetts

van der Aalst, W. (1998): *The Application of Petri Nets to Workflow Management*, The Journal of Circuits, Systems and Computers

Workflow Management Coalition, WfMC (1996): *Terminology & Glossary*, issue 2.0, http://www.wfmc.org/