

Editor: Daniel Moldt

Proceedings of the
International Workshop on
Modeling and
Business
Environments
ModBE'13

University of Hamburg
Department of Informatics

These proceedings are published online by the editor as Volume 989 at

CEUR Workshop Proceedings

ISSN 1613-0073

<http://ceur-ws.org/Vol-989>

Copyright for the individual papers is held by the papers' authors. Copying is permitted only for private and academic purposes. This volume is published and copyrighted by its editors.

Preface

These are the proceedings of the International Workshop on *Modeling and Business Environments* (ModBE'13) in Milano, Italy, June 24, 2013. It is a co-located event of *Petri Nets 2013*, the 34th international conference on Applications and Theory of Petri Nets and Concurrency.

More information about the workshop can be found at

<http://www.informatik.uni-hamburg.de/TGI/events/modbe13/>

Business environments are a central application domain for modeling approaches. Basic paradigms of these approaches correspond to their central concepts, such as processes, objects, components, agents, services or organizations. Their inherent properties allow an adequate Business/IT-Alignment. Within the models and systems of this alignment several principle notions need to be incorporated, such as distribution, concurrency, correctness and adaptability. In this workshop modeling approaches will be discussed from various perspectives with several means.

While ModBE'13 (Modeling and Business Environments) will take place as a satellite event of Petri Nets 2013 other modeling techniques than Petri nets and their means are explicitly welcome. Furthermore, experts from the application domain will challenge the technical and conceptual solutions. ModBE'13 shall provide a forum for researchers from interested communities to investigate, experience, compare, contrast and discuss solutions for modeling in business environments. During the workshop a part of the available time is reserved for a group wise discussion of challenging questions.

The program committee consists of:

Bernhard Bauer (Germany)
Olivier Boissier (France)
Fabian Büttner (France)
Jean-Michel Bruel (France)
Christine Choppy (France)
Ernesto Damiani (Italy)
Patrick Delfmann (Germany)
Susanna Donatelli (Italy)
Joaquín Ezpeleta Mateo (Spain)
Walid Fdhila (Austria)
Michael Felderer (Austria)
Luciano García-Bañuelos (Estonia)
Holger Giese (Germany)
Paolo Giogini (Italy)
Vincent Hilaire (France)
Lom Messan Hillah (France)
Viviana Mascardi (Italy)
Maristella Matera (Italy)

Florian Matthes (Germany)
Jan Mendling (Austria)
Daniel Moldt (Germany) (Chair)
Ambra Molesini (Italy)
Berndt Müller (United Kingdom)
Andreas Oberweis (Germany)
Andrea Omicini (Italy)
Sietse Overbeek (Germany)
Alexei Sharpanskykh (The Netherlands)
Christophe Sibertin-Blanc (France)
Carla Simone (Italy)
Ingo Timm (Germany)
Ferucio Laurentiu Tiplea (Rumania)
Adeline Uhrmacher (Germany)
Ulrich Ultes-Nitsche (Switzerland)
Wamberto Vasconcelos (United Kingdom)
Jan Martijn van der Werf (The Netherlands)
Mathias Weske (Germany)
Manuel Wimmer (Austria)

We received five high-quality contributions for which at least four reviews were made. In addition we received two posters. The program committee has accepted three of them for full presentation. Furthermore the committee accepted one papers as short presentations. Two more contributions were accepted as posters.

Furthermore, we would like to thank our colleagues in the local organization team at the University of Milano, Italy, for their support.

Without the enormous efforts of authors, reviewers, PC members and the organizational team this workshop wouldn't provide such an interesting booklet.

Thanks!

Daniel Moldt

Hamburg, June 2013

ModBE'13 Proceedings

Part VI ModBE'13: Invited Talk

- Knowledge and Business Intelligence Technologies in Cross-Enterprise Environments for Italian Advanced Mechanical Industry**
Ernesto Damiani and Paolo Ceravolo 271
-

Part VII ModBE'13: Long Presentations

- Optimizing Algebraic Petri Net Model Checking by Slicing**
Yasir Imtiaz Khan and Matteo Risoldi 275
- A Proposal for the Modeling of Organizational Structures and Agent Knowledge in MAS**
Lawrence Cabac, David Mosteller, Matthias Wester-Ebbinghaus 295
- Mining Declarative Models Using Time Intervals**
Jan Martijn van der Werf, Ronny Mans and Wil van der Aalst 313
-

Part VIII ModBE'13: Short Presentation

- Improving Emergency Department Processes Using Coloured Petri Nets**
Khodakaram Salimifard, Seyed Yaghoub Hosseini and Mohammad Sadegh Moradi 335
-

Part IX ModBE'13: Poster Abstracts

- Advantages of a Full Integration between Agents and Workflows**
Thomas Wagner and Lawrence Cabac 353
- Cloud Transition for QoS Modeling of Inter-Organizational Workflows**
Sofiane Bendoukha and Lawrence Cabac 355

ModBE'13: Invited Talk

Knowledge and Business Intelligence Technologies in Cross-Enterprise Environments for Italian Advanced Mechanical Industry

Ernesto Damiani and Paolo Ceravolo

SESAR Lab, Department of Computer Science,
Università degli Studi di Milano, Italy
<http://sesar.dti.unimi.it>

Abstract. Today's industry is pushed by the competitive pressure to revise the business model by opening the organizational boundaries to suppliers, clients and partners. As a side effect, knowledge sharing within the market increases and organizations may lose control on strategical knowledge that can be exploited by competitors. For this reason process monitoring today cannot fail in controlling the collaboration activities established inside and outside the organization.

KITE.it is a project, founded by the italian Ministry of Economic Development, aimed at proposing a methodological and technological framework to support the italian mechanical industry in adopting advanced business network approached.¹ The Kite framework is aimed at driving the management process in the identification of the business values creating the network and in supporting the strategical analysis by monitoring both the operational and collaborative processes.

The metric system was designed to integrate in a unified analysis metrics insisting on the strategical, operational and collaborative level. From the technological point of view this is achieved by decoupling the monitoring format from the execution logs that can be integrated in the Kite model from heterogeneous data sources.

¹ Website: http://www.kite-project.it/en_GB/home

ModBE'13: Long Presentations

Optimizing Algebraic Petri Net Model Checking by Slicing

Yasir Imtiaz Khan and Matteo Risoldi

University of Luxembourg, Laboratory of Advanced Software Systems
6, rue R. Coudenhove-Kalergi, Luxembourg
{yasir.khan,matteo.risoldi}@uni.lu

Abstract. High-level Petri nets make models more concise and readable as compared to low-level Petri nets. However, usual verification techniques such as state space analysis remain an open challenge for both because of state space explosion. The contribution of this paper is to propose an approach for property based reduction of the state space of Algebraic Petri nets (a variant of high-level Petri nets). To achieve the objective, we propose a slicing algorithm for Algebraic Petri nets (*APNSlicing*). The proposed algorithm can alleviate state space even for certain strongly connected nets. By construction, it is guaranteed that the state space of sliced net is at most as big as the original net. We exemplify our technique through the running case study of car crash management system.

Key words: High-level Petri nets, Model checking, Slicing

1 Introduction

Petri nets (PNs) are a well-known low-level formalism for modeling concurrent and distributed systems. Various evolutions of PNs have been created, among others *High-level Petri nets* (HLPNs), that raise the level of abstraction of PNs by using complex structured data [14]. However, HLPN can be *unfolded* (i.e., translated) into a behaviourally-equivalent low-level PN.

For the analysis of concurrent and distributed systems (including those modeled using PNs or HLPNs) model checking is a common approach, consisting in verifying a property against all possible states of a system. A typical drawback of model checking is its limits with respect to the *state space explosion* problem: as systems get moderately complex, completely enumerating their states demands a growing amount of resources, which in some cases makes model checking impractical both in terms of time and memory consumption [2, 4, 8, 16]. This is particularly true for HLPN models, as the use of complex data (with possibly large associated data domains) makes the number of states grow very quickly.

As a result, an intense field of research is targeting to find ways to optimize model checking, either by reducing the state space or by improving the performance of model checkers. A technique called *PN slicing* falls into the first category. It proposes to reduce the state space size by syntactically reducing a

PN model, taking only the portion of the model that impacts the properties to be verified. The resultant model will typically have a smaller state space, thus reducing the cost of model checking.

Slicing was defined for the first time in [17] in the context of program debugging. The proposition was aimed at using program slicing for isolating the program statements that may contain a bug, so that finding this bug becomes simpler for the programmer. The first algorithm about PN slicing presented by Chang et al. [3] slices out all sets of paths in the PN graph, called *concurrency sets*, such that all paths within the same set should be executed concurrently. Some further refined PN slicing algorithms are proposed in [10–13].

One limitation of the cited approaches is that they only apply to low-level PNs. In order to be applied to HLPNs they need to be adapted to take into account data types.

In this work, we propose a slicing algorithm that is adapted to Algebraic Petri nets (APNs, a variant of HLPNs). To the best of our knowledge, there does not exist any algorithm for slicing APNs. The proposed algorithm iteratively builds a subnet from a given APN, according to a *slicing criterion* that is derived from the property to be verified. The resulting subnet preserves LTL_X properties under weak fairness assumptions.

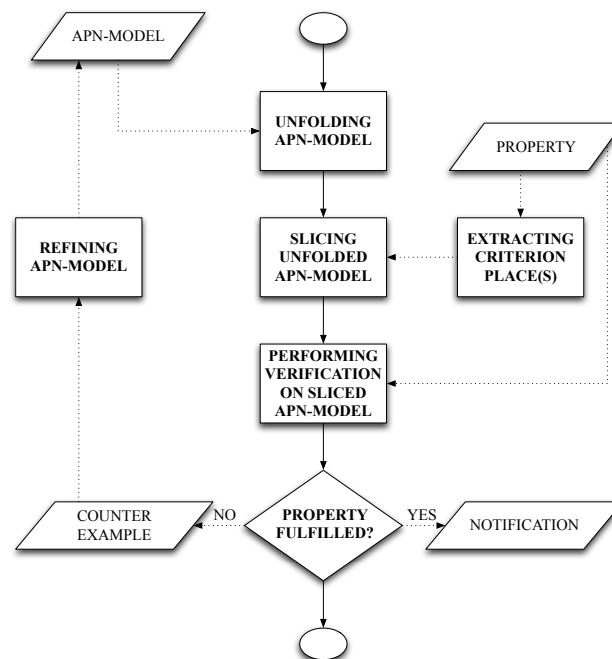


Fig. 1. Process Flowchart of *slicing* based verification of APN models

Fig.1, gives an overview of the proposed approach for slicing based verification of APNs using Process Flowchart. At first, APN-model is unfolded and then by taking property into an account *criterion places* are extracted. After-

wards, slicing is performed for the *criterion places*. Subsequently, verification is performed on the sliced unfolded APNs. The user may use the counterexample to refine the APN-model to correct the property.

The rest of the work is structured as follows: we give basic definitions and concepts of the Algebraic Petri nets (APNs) in section 2. Section 3, illustrates the steps of slicing based verification of APN-models shown in Fig.1. Details about the underlying theory and techniques are given for each activity of the process. In the section 4, we discuss related work and a comparison with the existing approaches. A small case study from the domain of crisis management system (a car crash management system) is taken to exemplify the proposed slicing algorithm in section 5. An experimental evaluation of the proposed algorithm is performed in section 6. In the section 7, we draw conclusions and discuss future work concerning to the proposed work.

2 Basic Definitions

In this section, we give basic formal definitions of algebraic specifications used in this paper. Formal definitions, propositions, lemmas and theorems are taken as is or with slight modifications from [6, 12, 14, 15].

Definition 1. A signature $\Sigma = (S, OP)$ consists of a set S of sorts, $OP = (OP_{w,s})_{w \in S^*, s \in S}$ is a $(S^* \times S)$ -sorted set of operation names of OP . For ϵ being the empty word, we call $OP_{\epsilon,s}$ the set of constant symbols.

Definition 2. A set X of Σ -variables is a family $X = (X_s)_{s \in S}$ of variables set, disjoint to OP .

Definition 3. The set of terms $T_{OP,s}(X)$ of sort s is inductively defined by:

1. $X_s \cup OP_{\epsilon,s} \subseteq T_{OP,s}(X)$;
2. $op(t_1, \dots, t_n) \in T_{OP,s}(X)$ for $op \in OP_{s_1, \dots, s_n, s}$, $n \geq 1$ and $t_i \in T_{OP,s_i}(X)$ (for $i = 1, \dots, n$).

The set $T_{OP,s} \equiv T_{OP,s}(\emptyset)$ contains the ground terms of sort s , $T_{OP}(X) \equiv \bigcup_{s \in S} T_{OP,s}(X)$ is the set of Σ -terms over X and $T_{OP} \equiv T_{OP}(\emptyset)$ is the set of Σ -ground terms.

Definition 4. A Σ -equation of sort s over X is a pair (l, r) of terms $l, r \in T_{OP,s}(X)$.

Definition 5. An algebraic specification $SPEC = (\Sigma, E)$ consists of a signature $\Sigma = (S, OP)$ and a set E of Σ -equations.

Definition 6. A Σ -algebra $A = (S_A, OP_A)$ consist of a family $S_A = (A_s)_{s \in S}$ of domains and a family $OP_A = (N_{op})_{op \in OP}$ of operations $N_{op} : A_{s_1} \times \dots \times A_{s_n} \rightarrow A_s$ for $op \in OP_{s_1 \dots s_n, s}$ if $op \in OP_{\epsilon, s}$, N_{op} congruent to an element of A_s .

Definition 7. An assignment of Σ -variables X to a Σ -algebra A is a mapping $ass : X \rightarrow A$, with $ass(x) \in A_s$ iff $x \in X_s$. ass is canonically extended to $\overline{ass} : T_{OP}(X) \rightarrow A$, inductively defined by

1. $\overline{ass}(x) \equiv ass(x)$ for $x \in X$;
2. $\overline{ass}(c) \equiv N_c$ for $c \in OP_{\epsilon,s}$;
3. $\overline{ass}(op(t_1, \dots, t_n)) \equiv N_{op}(\overline{ass}(t_1), \dots, \overline{ass}(t_n))$ for $op(t_1, \dots, t_n) \in T_{OP}(X)$.

Definition 8. Let SPEC-algebra is $SPEC = (\Sigma, E)$ in which all equations in E are valid. Two terms t_1 and t_2 in $T_{OP}(X)$ are equivalent ($t_1 \equiv_E t_2$) iff for all assignments $ass : X \rightarrow A$, $\overline{ass}(t_1) = \overline{ass}(t_2)$.

Definition 9. Let B be a set. A multiset over B is a mapping $ms_B : B \rightarrow \mathbb{N}$. ϵ_B is the empty multiset with $ms_B(x) = 0$ for all $x \in B$. A multiset is finite iff $\{\forall b \in B \mid ms_B(b) \neq 0\}$ is finite.

Definition 10. Let $MS_B = \{ms_B : B \rightarrow \mathbb{N}\}$ be a set of multisets. The addition function of multisets is denoted by $+$: $MS_B \times MS_B \rightarrow MS_B$. Let $ms1_B, ms2_B$ and $ms3_B \in MS_B$. $(ms1_B + ms2_B) = ms3_B \iff \forall b \in B, ms3_B(b) = ms1_B(b) + ms2_B(b)$.

The subtraction function of multisets is denoted by $-$: $MS_B \times MS_B \rightarrow MS_B$. Let $ms1_B, ms2_B$ and $ms3_B \in MS_B$. $(ms1_B - ms2_B) = ms3_B \iff \forall b \in B, ms1_B(b) \geq ms2_B(b) \Rightarrow \forall b \in B, ms3_B(b) = ms1_B(b) - ms2_B(b)$.

Definition 11. Let $MS_B = \{ms_B : B \rightarrow \mathbb{N}\}$ be a set of multisets. Let $ms1_B, ms2_B \in MS_B$. We say that $ms1_B$ is smaller than or equal to $ms2_B$ (denoted by $ms1_B \leq ms2_B$) iff

- $\forall b \in B, ms1_B(b) \leq ms2_B(b)$. Further, we say that $ms1_B \neq ms2_B$ iff $\exists b \in B, ms1_B(b) \neq ms2_B(b)$. Otherwise, $ms1_B = ms2_B$.

Definition 12. A marked Algebraic Petri Net $APN = \langle SPEC, P, T, F, asg, cond, \lambda, m_0 \rangle$ consist of

- an algebraic specification $SPEC = (\Sigma, E)$,
- P and T are finite and disjoint sets, called places and transitions, resp.,
- $F \subseteq (P \times T) \cup (T \times P)$, the elements of which are called arcs,
- a sort assignment $asg : P \rightarrow S$,
- a function, $cond : T \rightarrow \mathcal{P}_{fin}(\Sigma - \text{equation})$, assigning to each transition a finite set of equational conditions.
- an arc inscription function λ assigning to every (p, t) or (t, p) in F a finite multiset over $T_{OP, asg(p)}$,
- an initial marking m_0 assigning a finite multiset over $T_{OP, asg(p)}$ to every place p .

Definition 13. The preset of $p \in P$ is $\bullet p = \{t \in T \mid (t, p) \in F\}$ and the postset of p is $p \bullet = \{t \in T \mid (p, t) \in F\}$. The pre and post sets of $t \in T$ defined as: $\bullet t = \{p \in P \mid (p, t) \in F\}$ and $t \bullet = \{p \in P \mid (t, p) \in F\}$.

Definition 14. A marking m of an APN assigns to every place $p \in P$ a multiset over $T_{OP, asg(p)}$.

Definition 15. An occurrence mode is a ground substitution of $\text{cond}(t), m(p), \lambda(p, t)$ and $\lambda(t, p)$ where $p \in P, t \in T$. Obviously, ground substitutions are the syntactical representations of assignments.

Definition 16. A transition $t \in T$ of an APN is enabled in an occurrence mode at a marking m iff for all p in P with $(p, t) \in F, \lambda(p, t) \leq m(p)$. If a transition t is enabled in an occurrence mode at a marking m , then t may occur returning the marking m' , where for all $p \in P, m'(p) = m(p) - \lambda(p, t) + \lambda(t, p)$. We write $m[t]m'$ in this case.

Definition 17. A firing sequence σ of a marked APN is maximal iff either σ is of infinite length or $\exists t \in T : m_0([\sigma t]),$ where $|\sigma| \in (\mathbb{N} \cup \{\infty\})$.

Definition 18. Let $\sigma = t_1, t_2 \dots$ be an infinite firing sequence of APN with $m_i[t_{i+1}]m_{i+1}, \forall i, 0 \leq i.$ σ permanently enables $t \in T$ iff $\exists i, 0 \leq i : \forall j, i \leq j : m_j[t]$.

3 Unfolding and Slicing APNs

One characteristic of APNs that makes them complex to model check is the use of variables on arcs. Computing variable bindings at runtime is extremely costly. ALPiNA (a symbolic model checker for Algebraic Petri nets) allows the user to define partial algebraic unfolding and presumed bounds for infinite domains [1], using some aggressive strategies for reducing the size of large data domains. Unfolding generates all possible firing sequences from the initial marking of the APN, though maintaining a partial order of events based on the causal relation induced by the net. Concurrency is preserved.

The basic idea of the slicing algorithm is to start by identifying which places in the unfolded APN model are directly concerned by a property. These places constitute the *slicing criterion*. The algorithm will then take all the transitions that create or consume tokens from the criterion places, plus all the places that are pre-condition for those transitions. This step is iteratively repeated for the latter places, until reaching a fixed point.

We refine the slicing construction by distinguishing between *reading* and *non-reading transitions*. The conception of *reading and non-reading transitions* is some what similar notion introduced in [13]. The principle difference is that we adapt the notion of *reading and non-reading transitions* in the context of APNs. Informally, *reading transitions* are not supposed to change the marking of a place. On the other hand *non-reading transitions* are supposed to change the markings of a place. In our proposed slicing construction, we discard *reading transitions* and include only *non-reading transitions*. Formally, we can define the conception of *reading and non-reading transitions* such as:

Definition 19. Let N be an unfolded APN and $t \in T$ be a transition. We call t a *reading-transition* iff its firing does not change the marking of any place $p \in (\bullet t \cup t \bullet)$, i.e., iff $\forall p \in (\bullet t \cup t \bullet), \lambda(p, t) = \lambda(t, p)$. Conversely, we call t a *non-reading transition* iff $\lambda(p, t) \neq \lambda(t, p)$.

Due to partial unfolding, there could be some domains that are not unfolded. For some cases, we are still able to identify *non-reading transitions* even if domains are not unfolded. If for example, we have a case where the multiplicities or cardinalities of terms in $\lambda(p, t), \lambda(t, p)$ are different then we can immediately state $\lambda(p, t) \neq \lambda(t, p)$. But for some cases, we don't have such a clear indication of the inequality between $\lambda(p, t)$ and $\lambda(t, p)$, for example, in the Fig.2, we see that $\lambda(p, t) = 1 + y$ and $\lambda(t, p) = 2 + x$ (defined over naturals). Both terms has the same multiplicity and cardinality, so we need to know for which values of the variables it would be a *non-reading transition*. In general, the evaluation of terms to check their equality for all the values is undecidable. For this particular case, we would like to have a set of constraints from the user. Informally,

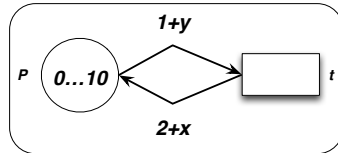


Fig. 2. An example APN model with non-unfolded terms over the arcs

a constraints set denoted by CS , is a set of propositional formulas, predicate formulas or any other logical formulas for certain specific values of variable assignments, describing the conditions under which we can evaluate terms to be equal or not. Consequently, constraints set CS will help to identify under which cases the transitions can be treated as *non-reading*.

A function $eval : T_{OP,s}(X) \times T_{OP,s}(X) \times CS \rightarrow Bool$ is used to evaluate the equivalence of terms based on the constraint set. Let us take the same terms shown over the arcs in Fig.2, $term_1 = 1 + y$, $term_2 = 2 + x$ and a constraint set $CS = \{\exists y, x \in (0, \dots, 2) | y = x + 1\}$. It is important to note that we are not unfolding the domain but evaluating the terms for some specific values provided by user to identify *reading and non-reading transitions*. Of course, the user can provide sparse values too. Let us evaluate the terms $term_1$ and $term_2$ based on the constraints set CS provided. For all those values of x, y for which we get $eval$ function result true are considered to be *reading transitions* and rest of them are *non-reading transitions*. It is also important to note that we include this step during the unfolding. The resulting unfolded APN will contain only *non-reading transitions* for the unfolded domains as shown in Fig.3.

The algorithm proposed in this article assumes that such an unfolding takes place before the slicing. Since this is a step that is involved in the model checking activity anyway, we do not consider this assumption to be adding to the complexity of the algorithm. In this section, we will make an extremely simple example of how the slicing algorithm works, starting from an APN, unfolding it and slicing it.

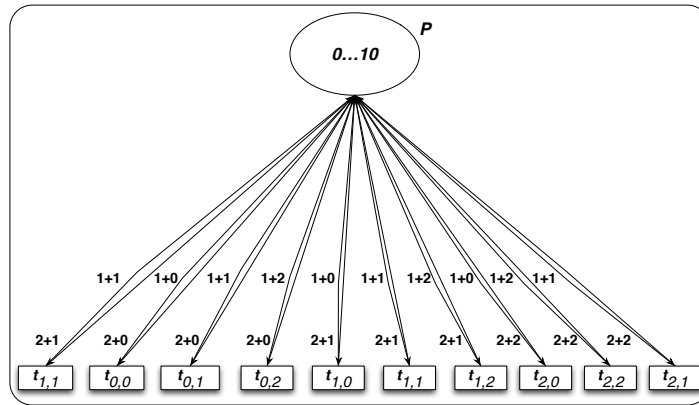


Fig. 3. Resulting unfolded APN after applying the *eval* function

3.1 Example: Unfolding an APN

Fig. 4 shows an APN model. All places and all variables over the arcs are of sort *naturals* (defined in the algebraic specification of the model, and representing the \mathbb{N} set).

Since the \mathbb{N} domain is infinite (or anyway extremely large even in its finite computer implementations), it is clear that it is impractical to unfold this net by considering all possible bindings of the variables to all possible values in \mathbb{N} . However, given the initial marking of the APN and its structure it is easy to see that none of the terms on the arcs (and none of the tokens in the places) will ever assume any natural value above 3. For this reason, following [1], we can set a *presumed bound* of 3 for the *naturals* data type, greatly reducing the size of the data domain. By assuming this bound, the unfolding technique in [1]

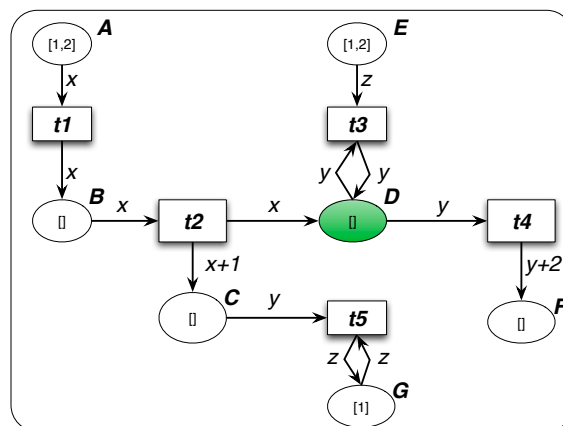


Fig. 4. An example APN model (*APNexample*)

proceeds in three steps. First, the data domains of the variables are unfolded up

to the presumed bound. Second, variable bindings are computed, and only that satisfy the transition guards are kept. Third, the computed bindings are used to instantiate a binding-specific version of the transition. The resulting unfolded APN for this APN model is shown in Fig. 5. The transitions arcs are indexed with the incoming and outgoing values of tokens. A complete explanation of the unfolding algorithm, and in particular the existence of the tokens 4 and 5 between transition t_{23}, t_{42}, t_{43} and place C, F is rather complex and out of the scope of this article. The interested reader can find details about the partial unfolding in [1].

3.2 The slicing algorithm

The slicing algorithm starts with an unfolded APN and a slicing criterion $Q \subseteq P$.

Let $Q \subseteq P$ a non empty set called slicing criterion. We can build a slice for an *unfolded APN* based on Q , using following algorithm:

Algorithm 1: APN slicing algorithm

```

APNSlicing( $\langle SPEC, P, T, F, asg, cond, \lambda, m_0 \rangle, Q$ ) {
   $T' = \{t \in T \mid \exists p \in Q : t \in (\bullet p \cup p \bullet) : \lambda(p, t) \neq \lambda(t, p)\}$ ;
   $P' = Q \cup \{\bullet T'\}$  ;
   $P_{done} = \emptyset$  ;
  while  $((\exists p \in (P' \setminus P_{done}))$  do
    while  $(\exists t \in (\bullet p \cup p \bullet) \setminus T' : \lambda(p, t) \neq \lambda(t, p))$  do
       $P' = P' \cup \{\bullet t\}$ ;
       $T' = T' \cup \{t\}$ ;
    end
     $P_{done} = P_{done} \cup \{p\}$ ;
  end
  return  $\langle SPEC, P', T', F|_{P', T'}, asg|_{P'}, cond|_{T'}, \lambda|_{P', T'}, m_{0|_{P'}} \rangle$ ;
}
```

Initially, T' (representing transitions set of the slice) contains set of all *pre and post* transitions of the given criterion place. Only *non-reading* transitions are added to T' set. And P' (representing places set of the slice) contains all *preset* places of transitions in T' . The algorithm then iteratively adds other *preset* transitions together with their *preset* places in T' and P' . Remark that the *APNSlicing* algorithm has linear time complexity.

Considering the APN-Model shown in fig. 4, let us now take an example property and apply our proposed algorithm on it. Informally, we can define the property:

“The values of tokens inside place D are always smaller than 5”.

Formally, we can specify the property in *LTL* as $\mathbf{G}(\forall tokens \in D | tokens < 5)$. For this property, the slicing criterion $Q = \{D\}$, as D is the only place concerned by the property. Therefore, the application of *APNSlicing(UnfoldedAPN, D)* returns *SlicedUnfoldedAPN* (shown in Fig. 6), which is smaller than the original *UnfoldedAPN* shown in Fig. 5).

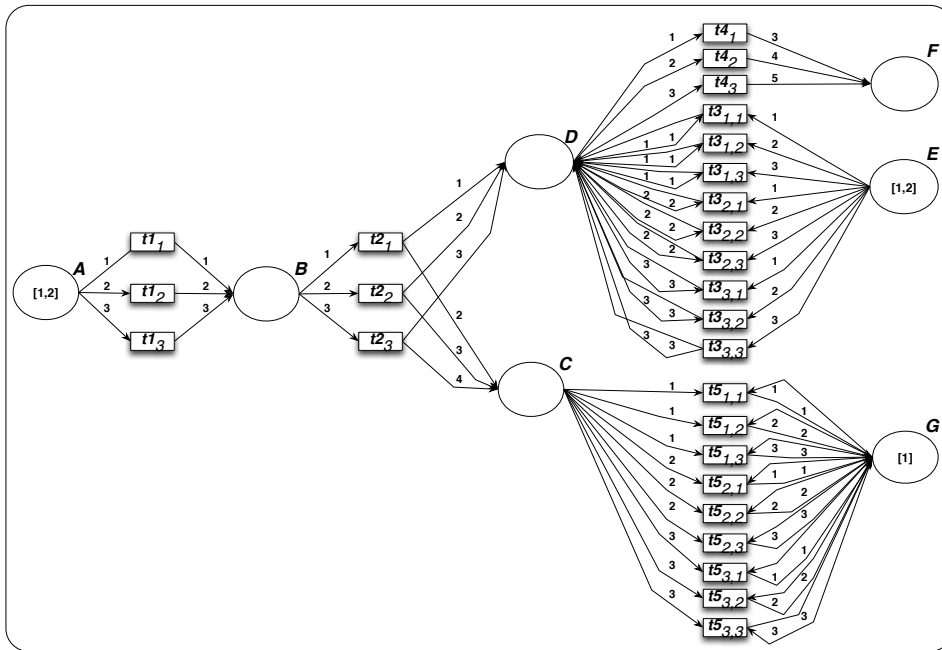


Fig. 5. The unfolded example APN model (*UnfoldedAPN*)

Transitions $t_{3_{1,1}}, t_{3_{1,2}}, t_{3_{1,3}}, t_{3_{1,3}}, t_{3_{2,1}}, t_{3_{2,2}}, t_{3_{2,3}}, t_{3_{3,1}}, t_{3_{3,2}}, t_{3_{3,3}}, t_{5_{1,1}}, t_{5_{1,2}}, t_{5_{1,3}}, t_{5_{2,1}}, t_{5_{2,2}}, t_{5_{2,3}}, t_{5_{3,1}}, t_{5_{3,2}}, t_{5_{3,3}}$, and places C, E, F, G has been sliced away. The proposed algorithm determines a slice for any given criterion $Q \subseteq P$ and always terminates. It is important to note that the reduction of net size depends on the structure of the net and on the size and position of the slicing criterion within the net.

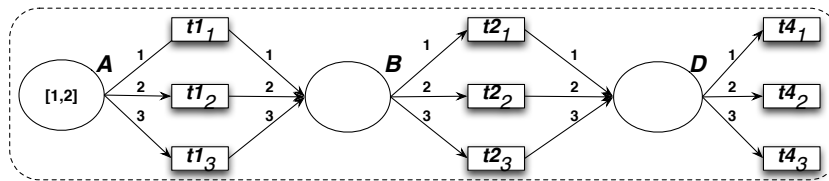


Fig. 6. Sliced and Unfolded example APN model (*SlicedUnfoldedAPN*)

3.3 Proof of the preservation of properties

To allow the verification by slice, we have to make restrictions on the formulas and on admissible firing sequences in terms of fairness assumptions. The original Algebraic Petri net has more behaviors than the sliced APN, as we intentionally do not capture all the behaviors.

Definition 20. Let A be the set of atomic propositions. Let $\varphi, \varphi_1, \varphi_2$ be LTL formulas. The function *scope* associates with an LTL formula φ the set of atomic propositions used in φ i.e. $\text{scope} : \varphi \rightarrow \mathcal{P}A$.

$$\begin{aligned} \text{scope}(a) &= \{a\} \text{ for } a \in A; \\ \text{scope}(\otimes\varphi) &= \text{scope}(\varphi) \text{ with } \otimes \in \{\neg, X\}; \\ \text{scope}(\varphi_1 \otimes \varphi_2) &= \text{scope}(\varphi_1) \cup \text{scope}(\varphi_2) \text{ with } \otimes \in \{\wedge, U\}. \end{aligned}$$

Definition 21. Let N be a marked APN. Let N' be its sliced net for a slicing criterion $Q \subseteq P$. Let $\sigma = t_1 t_2 t_3 \dots$ be a firing sequence of N and m_i the markings with $m_i[t_{i+1}]m_{i+1}, \forall i, 0 \leq i < |\sigma|$. σ is slice-fair w.r.t N' iff either σ is finite and $m_{|\sigma|}$ does not enable any transition $t \in T'$;

or σ is infinite and if it permanently enables some $t \in T'$, it then fires infinitely often some transition of T' (which may or may not be the same as t).

Slice-fairness is a very weak fairness notion. Weak fairness determines that every transition $t \in T$ of a system, if permanently enabled, has to be fired infinitely often, slice-fairness concerns only the transitions of the slice, not of the entire system net and if a transition $t \in T$ of the slice is permanently enabled, some transitions of the slice are required to fire infinitely often but not necessarily t .

Definition 22. Let N be a marked APN and φ an LTL formula. $N \models \varphi$ slice-fairly iff every slice-fair (not necessarily maximal) firing sequence of $\sigma \models \varphi$.

Definition 23. Let N and N' be two marked Algebraic Petri nets with $T' \subseteq T$ and $P' \subseteq P$. We define the function: $\text{slice}_{(N, N')} \in [(T^* \cup T^w) \rightarrow (T'^* \cup T'^w)] \cup [\mathbb{N}^{|P|} \rightarrow \mathbb{N}^{|P'|}]$ such that a finite or infinite sequence of transitions σ is mapped onto the transition sequence σ' with σ' being derived from σ by omitting every transition $t \in T \setminus T'$. A marking m of N is projected onto the marking m' of N' with $m' = m|_{P'}$.

The function *slice* is used to project markings and firing sequences of a net N onto the markings and firing sequences of its slices.

Proposition 1. Let N be a marked APN. Let N' be its sliced net for a slicing criterion $Q \subseteq P$. Let σ be a weakly fair firing sequence of N . σ is slice fair with respect to N' .

Proof. Let us assume, σ is not slice-fair. In case σ is finite this means that $m_{|\sigma|}[t]$ for a transition $t \in T'$. In case σ is infinite, there is permanently enabled transition $t \in T'$ but all transitions of T' are fired finitely often including t . So both cases contradict the assumption that σ is weakly fair.

Lemma 1. Let N be a marked APN and let N' be its sliced net for a slicing criterion $Q \subseteq P$. The coefficients c_{ij} of the incidence matrix equal to zero for all places $p_i \in P'$ and transitions $t_j \in T \setminus T'$.

Proof. Let N' be its sliced net for a slicing criterion $Q \subseteq P$. A transition $t \in T$ is also an element of $T' \subseteq T$, if it is a *non-reading* transition of a place $p \in P'$. Thus a transition $t \in T \setminus T'$ either is not connected to a place $p \in P'$ or it is a *reading* transition.

Lemma 2. *Let N be a marked APN and let N' be its sliced net for a slicing criterion $Q \subseteq P$. Let m be a marking of N and m' be a marking of N' with $m' = m|_{P'}$. $m[t] \Leftrightarrow m'[t], \forall t \in T'$.*

Proof. Let N' be its sliced net for a slicing criterion $Q \subseteq P$. Since a transition $t \in T'$ has the same preset places in N and N' by the slicing algorithm *APNSlicing*, $m' = m|_{P'}$ implies $m[t] \iff m'[t]$.

Every firing sequence σ of N projected onto the transitions of T' is also a firing sequence of slice net N' . The resulting markings m and m' assign the same number of tokens to places $p' \subseteq P$.

Proposition 2. *Let N be a marked APN and let N' be its sliced net for a slicing criterion $Q \subseteq P$. Let σ be a firing sequence of N and let m be a marking of N . $m_0[\sigma]m \Rightarrow m_0|_{P'}[slice(\sigma)]m|_{P'}$.*

Proof. We prove this Proposition by induction over the length l of σ . Let N be a marked APN, σ be a firing sequence of N .

$l = 0$: In this case $slice(\sigma)$ equals ϵ . Thus the initial marking of N and N' is generated by firing ϵ . By definition 23 and the slicing algorithm *APNSlicing*, $m'_0 = m_0|_{P'}$

$l \rightarrow l + 1$: Let σ be a firing sequence of length l and m_l be a marking of N with $m_0[\sigma]m_l$. Let t_{l+1} be a transition in T and m_{l+1} a marking of N such that $m_l[t_{l+1}]m_{l+1}$. By induction hypothesis, $m'_0[slice(\sigma)]m'_k$ with $m_l|_{P'} = m'_k$. If t_{l+1} is an element of T' , it follows by Lemma 2, that m'_k enables t_{l+1} , since m_l enables t_{l+1} . The resulting marking m'_{k+1} is determined by $m'_{k+1}(P'_i) = m'_k(P'_i) + c_{i,l+1}, \forall p_i \in P'$ and m_{l+1} is determined by $m_{l+1}(i) = m_l(i) + c_{i,l+1}, \forall p_i \in P'$.

Since $m_l|_{P'} = m'_k$, it thus follows that $m_{l+1}|_{P'} = m'_{k+1}$. If t_{l+1} is an element of $t \in T \setminus T'$, then it must be a reading transition for all $p \in P$; $slice(\sigma) = slice(\sigma t_{l+1})$ and thus $m'_0[slice(\sigma t_{l+1})]m'_k$ a transition $t \in T \setminus T'$ can not change the marking of on any place $p \in P'$. By Lemma 1 and the resultant markings, $m_{l+1}|_{P'} = m'_l|_{P'}$. \square

A firing sequence σ' of the slice net N' is also a firing sequence of N . The resulting markings of σ' on N and N' , respectively assigns the same markings to places $p \in P'$.

Proposition 3. *Let N be a marked APN and let N' be its sliced net for a slicing criterion $Q \subseteq P$. Let σ' be a firing sequence of N' and let m' be a marking of N' .*

$$m'_0[\sigma']m' \Rightarrow \exists m \in \mathbb{N}^{|P|} : m' = m|_{P'} \wedge m_0[\sigma']m.$$

Proof. We prove this Proposition by induction over the length l of σ' .

$l = 0$: The empty firing sequence generates the marking m_0 on N and the marking m'_0 , which is defined as $m_0|_{P'}$, on N' , by definition 23.

$l \rightarrow l + 1$: Let $\sigma' = t_1 \dots t_{l+1}$ be firing sequence of N' with length $l + 1$. Let m'_l and m'_{l+1} be markings of N' such that $m'_0[t_1 \dots t_l]m'_l[t_{l+1}]m'_{l+1}$. Let m_l be

the marking of N with $m_0[t_1 \dots t_l]m_l$ and $m_l|_{P'} = m'_l$, which exists according to the induction hypothesis. Lemma 2, m_l enables t_{l+1} . The marking m_{l+1} satisfies $m_{l+1}(P_i) = m_l(P_i) + c_{i,l+1}, \forall p_i \in P'$ and m'_{l+1} satisfies $m'_{l+1}(P_i) = m'_l(P_i) + c_{i,l+1}, \forall s_i \in P'$. With $m_l|_{P'} = m'_l$, it follows that $(m_{l+1} |_{P'})$ is equal to m'_{l+1} . \square

Proposition 4. *Let N be a marked APN and let ϕ be an LTL_x formula such that $\text{scope}(\phi) \subseteq P$. Let N' be its sliced net for a slicing criterion $Q \subseteq P$ where $Q = \text{scope}(\phi)$. Let σ be a firing sequence of N . Let us denote the sequence of markings by $\mathcal{M}(\sigma)$. Then, $\mathcal{M}(\sigma) \models \phi \Leftrightarrow \mathcal{M}(\text{slice}(\sigma)) \models \phi$.*

Proof. We prove this Proposition by induction on the structure of ϕ . Let $\sigma = t_1 t_2 \dots$ and $\text{slice}(\sigma)$ be $\sigma' = t'_1 t'_2 \dots$. Let $\mathcal{M}(\sigma) = m_0 m_1 \dots$ and $\mathcal{M}(\sigma') = m'_0 m'_1 \dots$.

$\phi = \text{true}$: In this case nothing needs to be shown. $\phi = \neg\psi, \phi = \psi_1 \wedge \psi_2$: Since the satisfiability of ϕ depends on the initial marking of $\text{scope}(\phi)$ only and $\text{scope}(\phi) \subseteq P' \subseteq P$, both directions hold.

$\phi = \psi_1 U \psi_2$: We assume that $\mathcal{M}(\sigma') \models \psi_1 U \psi_2$. We can divide up σ' such that $\sigma' = \sigma'_1 \sigma'_2$ with $m'_{|\sigma'_1|} m'_{|\sigma'_1|+1} \dots \models \psi_2$ and $\forall i, 0 \leq i < |\sigma'_1| : m'_i m'_{i+1} \dots \models \psi_1$. There are transition sequences σ_1 and σ_2 such that $\sigma = \sigma_1 \sigma_2, \text{slice}(\sigma_1) = \sigma'_1, \text{slice}(\sigma_2) = \sigma'_2$ and σ_1 does not end with a transition $t \in T \setminus T'$.

By proposition 2, it follows that $m'_{|\sigma'_1|} = (m_{|\sigma_1|} |_{P'})$. Since $m'_{|\sigma'_1|} m'_{|\sigma'_1|+1} \dots \models \psi_2, m_{|\sigma_1|} m_{|\sigma_1|+1} \dots \models \psi_2$ by induction hypothesis. Let ϱ be a prefix of σ_1 such that $|\varrho| < |\sigma_1|$. Let ϱ' be $\text{slice}(\varrho)$. The firing sequence ϱ truncates at least one transition $t \in T'$, consequently $|\varrho'| < |\sigma'_1|$. Since $m'_{|\varrho'|} m'_{|\varrho'|+1} \dots \models \psi_1, m_{|\varrho|} m_{|\varrho|+1} \dots \models \psi_1$ by the induction hypothesis. Analogously, it can be shown that $\mathcal{M}(\sigma) \models \psi_1 U \psi_2$ implies $\mathcal{M}(\sigma') \models \psi_1 U \psi_2$. \square

Proposition 5. *Let N be a marked APN and let N' be its sliced net for a slicing criterion $Q \subseteq P$. Let σ' be a maximal firing sequence of N' . σ' is a slice-fair firing sequence of N .*

Proof. Let $\sigma' = t_1 t_2 \dots$. Let m'_i be the marking of N' , such that $m'_i[t_{i+1}]m'_{i+1}, \forall i, 0 \leq i < |\sigma'|$. By Proposition 3 σ' is a firing sequence of N . Let m_i be the marking of N , such that $m_i[t_{i+1}]m_{i+1}, \forall i, 0 \leq i < |\sigma'|$. In case σ' is finite, $m'_{|\sigma'|}$ does not enable any transitions $t' \in T'$.

By Lemma 2, $m_{|\sigma'|}$ does not enable any transition $T' \in T'$, If σ' is infinite it obviously fires infinitely often a transition $t' \in T'$ and thus is slice-fair. \square

Proposition 6. *Let N be a marked APN and let N' be its sliced net for a slicing criterion $Q \subseteq P$. $\text{Slice}(\sigma)$ is maximal firing sequence of N' .*

Proof. Let $\sigma = t_1 t_2 \dots$ with $m_i[t_{i+1}]m_{i+1}, \forall i, 0 \leq i < |\sigma|$. By Proposition 2, $\text{slice}(\sigma)$ is a firing sequence of N' . Let $\text{slice}(\sigma)$ be $\sigma' = t'_1 t'_2 \dots$ with $m'_i[t'_{i+1}]m'_{i+1}, \forall i, 0 \leq i < |\sigma|$. Let us assume σ' is not a maximal firing sequence of N' . Thus σ' is finite and there is a transition $t' \in T'$ with $m'_{|\sigma'|}[t']$. Let σ_1 be the smallest prefix of σ such that $\text{slice}(\sigma_1)$ equals σ' .

By Proposition 2 ($m_{|\sigma_1|} \upharpoonright_{P'} = m'_{|\sigma'_1|}$). By Lemma 2, and the state equation it follows, that $(m_{|\sigma_1|} \upharpoonright_{P'} = m'_{|\sigma'_1|+1} = \dots$. So t' stays enabled for all markings m_j with $|\sigma_1| \leq j \leq |\sigma|$ but is fired finitely many times only. This is a contradiction to the assumption that σ is slice-fair. \square

Theorem 1. *Let N be a marked APN and let ϕ be an LTL formula such that $\text{scope}(\phi) \subseteq P$. Let N' be its sliced net for a slicing criterion $Q \subseteq P$. Let Ψ be an LTL_{-X} formula with $\text{scope}(\Psi) \subseteq P$.*

$N \models \phi$ slice-fairly $\Rightarrow N' \models \phi$, for an LTL formula ϕ .

$N \models \Psi$ slice-fairly $\Leftarrow N' \models \Psi$, for an LTL_{-X} formula Ψ .

Proof. We first show “ $N \models \phi$ slice-fairly $\Rightarrow N' \models \phi$ ”. Let us assume that $N \models \phi$ slice-fairly holds. Let σ' be a maximal firing sequence of N' . Since σ' is a slice-fair firing sequence of N by Proposition 5 $\mathcal{M}(\sigma') \models \phi$. Let us now assume $N' \models \Psi$. Let σ be a slice-fair firing sequence of N . By Proposition 6, $\text{slice}(\sigma)$ is maximal firing sequence of N' and thus satisfies Ψ . By Proposition 4, it follows that σ satisfies Ψ . \square

Verification is possible under interleaving semantics if we assume slice-fairness. A firing sequence σ is fair w.r.t T' , if σ is either maximal and if σ eventually permanently enables a $t' \in T'$, a transition $t \in T'$ will be fired infinitely often, t may not equal t' . Unfolded APN $\models \varphi$ fairly w.r.t. T' holds if all fair firings sequences of N , more precisely, their corresponding traces satisfy φ .

4 Related Work

Slicing is a technique used to reduce a model syntactically. The reduced model contains only those parts that may affect the property the model is analyzed for. Slicing Petri nets is gaining much attention in the recent years [3, 10–13]. Mark Weiser introduced the slicing term in [17], and presented slicing as a formalization of an abstraction technique that experienced programmers (unconsciously) use during debugging to minimize the program. The first algorithm about Petri net slicing was presented by Chang et al [3]. They proposed an algorithm on Petri nets testing that slices out all sets of paths, called concurrency sets, such that all paths within the same set should be executed concurrently. Lee et al. proposed the Petri nets slice approach in order to partition huge place transition net models into manageable modules, so that the partitioned model can be analyzed by compositional reachability analysis technique [9]. Llorens et al. introduced two different techniques for dynamic slicing of Petri nets [10]. A slice is said to be static if the input of the program is unknown (this is the case of Weiser’s approach). On the other hand, it is said to be dynamic if a particular input for the program is provided, i.e., a particular computation is considered. In the first technique of Llorens et al. the Petri net and an initial marking is taken into account, but produces a slice w.r.t. any possibly firing sequence. The second approach further reduces the computed slice by fixing a particular firing sequence.

Astrid Rakow developed two flavors of Petri net slicing, CTL^*_{-X} slicing and *Safety slicing* in [13]. The key idea behind the construction is to distinguish between reading and non-reading transitions. A reading transition $t \in T$ can not change the token count of place $p \in P$ while other transitions are non-reading transitions. For CTL^*_{-X} slicing, a subnet is built iteratively by taking all non-reading transitions of a place P together with their input places, starting with given criterion place. And for the *Safety slicing* a subnet is built by taking only transitions that increase token count on places in P and their input places. CTL^*_{-X} slicing algorithm is fairly conservative. By assuming a very weak fairness assumption on Petri net it approximates the temporal behavior quite accurately by preserving all CTL^*_{-X} properties and for safety slicing focus is on the preservation of stutter-invariant linear safety properties only.

We notice that all the constructions are limited to low-level Petri nets. The main difference between High-level and low-level Petri net is that in high-level Petri nets tokens are no longer black dots, but complex structured data. Whereas in case of low-level Petri nets, all (black) tokens correspond to the same data object. The idea of reading and non-reading transitions introduced in [13] deals only with the token count of places in low-level Petri nets. In Algebraic Petri nets there are properties that may concern to the values of tokens. The main difference between the existing slicing constructions such as, CTL^*_{-X} , *Safety slicing* and our is that in CTL^*_{-X} , *Safety slicing* only transitions are included that change the token count whereas in *APNSlicing*, we include transitions that change the token values together with the transitions that change the token count. A comparison between *APNSlicing*, CTL^*_{-X} and safety slicing algorithms is shown in Fig. 7.

<i>Slice CTL^*_{-X}</i>	Preserving all CTL^*_{-X} properties assuming a weak fairness assumption	Properties are about token count only	Designed for Low-level Petri net
<i>Safety Slicing</i>	Preserving safety properties only	Properties are about token count only	Designed for Low-level Petri net
<i>APNSlicing</i>	Preserving all LTL_{-X} properties assuming a weak fairness assumption	Properties are about token count and token values	Designed for High-level Petri net (APNs)

Fig. 7. A comparison between *APNSlicing*, CTL^*_{-X} and *Safety slicing*

5 Case Study

We took a small case study from the domain of crisis management systems (car crash management system) for the experimental investigation of the proposed

approach. In a car crash management system (CCMS); reports on a car crash are received and validated, and a *Superobserver* (i.e., an emergency response team) is assigned to manage each crash.

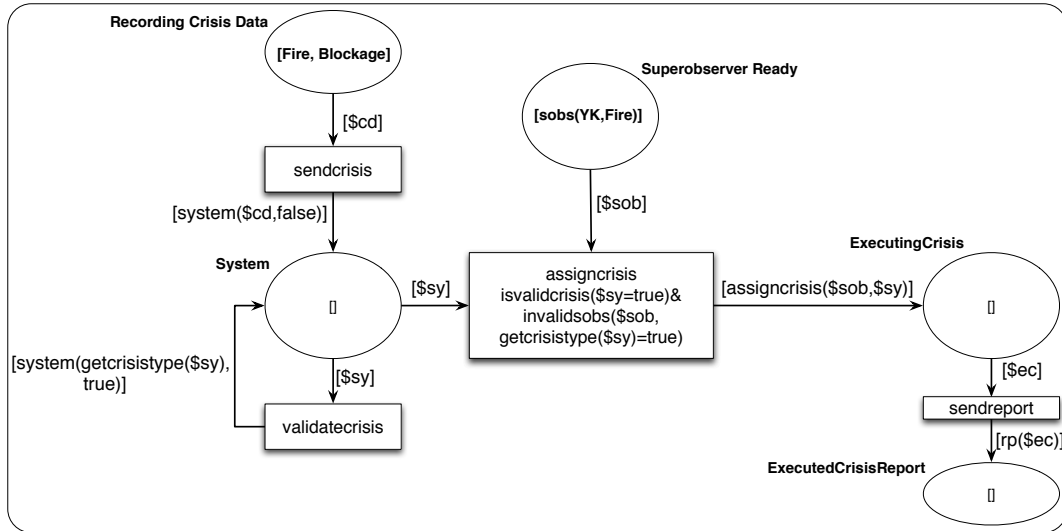


Fig. 8. Car crash APN model

The APN Model can be observed in Fig. 8, it represents the semantics of the operation of a car crash management system. This behavioral model contains labeled places and transitions. There are two tokens of type *Fire* and *Blockage* in place *Recording Crisis Data*. These tokens are used to mention which type of data has been recorded. The input arc of transition *sendcrisis* takes the *cd* variable as an input from the place *Recording Crisis Data* and the output arc contains term $system(cd, false)$ of sort *sys*. Initially, every recoded crisis is set to false. The *sendcrisis* transition passes recorded crisis to system for further operations. The output arc of *validatecrisis* contains $system(getcrisistype(sy), true)$ term which sends validated crisis to system. The transition *assigncrisis* has two guards, first one is $isvalid(sy)=true$ that enables to block invalid crisis reporting to be executed for the mission and the second one is $isvalid(sob, getcrisistype(sy))=true$ which is used to block invalid *Superobserver* (a skilled person for handling crisis situation) to execute the crisis mission. The *Superobserver YK* will be assigned to handle *Fire* situation only. The transition *assigncrisis* contains two input arcs with *sob* and *sy* variables and the output arc contains term $assigncrisis(sob, sy)$ of sort *crisis*. The output arc of transition *sendreport* contains term $rp(ec)$. This enables to send a report about the executed crisis mission. We refer the interested reader to [7] for the algebraic specification of car crash management system.

An important safety threat, which we will take into an account in this case study is that the invalid crisis reporting can be hazardous. The invalid crisis reporting is the situation that results from a wrongly reported crisis. The ex-

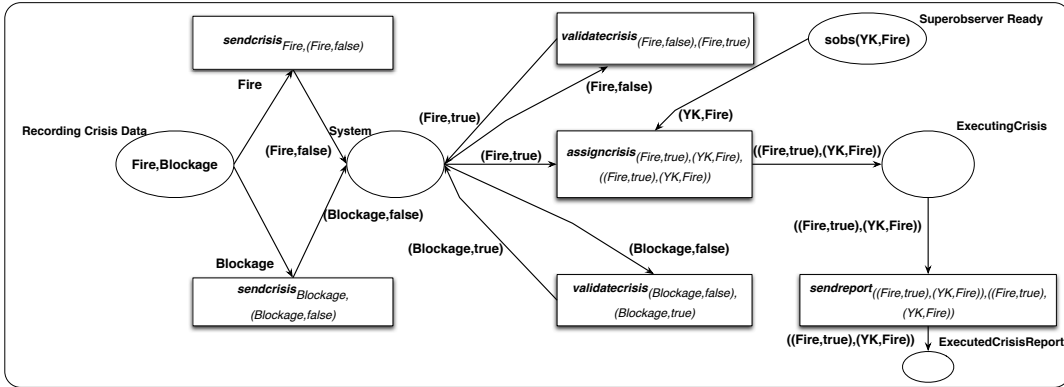


Fig. 9. The unfolded car crash APN model

execution of crisis mission based on the wrong reporting can waste both human and physical resources. In principle, it is essential to validate the crisis that it is reported correctly. Another, important threat could be to see the number of crisis that can be sent to place *System* should not exceed from a certain limit. Informally, we can define the properties:

φ_1 : All the crisis inside place *System* are validated eventually.

φ_2 : Place *System* never contains more than two crisis.

Formally we can specify the properties as, let *Crises* be a set representing recorded crisis in car crash management system. Let $isvalid : Crises \rightarrow BOOL$, is a function used to validate the recorded crisis.

$$\varphi_1 = \mathbf{F}(\forall crisis \in Crises | isvalid(crisis) = true)$$

$$\varphi_2 = \mathbf{G}(|Crises| \leq 2)$$

In contrast to generate the complete state space for the verification of φ_1 and φ_2 , we alleviate the state space by applying our proposed algorithm. For both φ_1, φ_2 LTL formulas, $scope(\varphi_1 \wedge \varphi_2) \subseteq Q$. The criterion place(s) for both properties is *System*.

The unfolded car crash APN model is shown in Fig. 9. The slicing algorithm $APNSlicing(Unfolded\ car\ crash\ APN\ model, System)$ takes the unfolded car crash APN model and *System* (an input criterion place) as an input and iteratively builds a sliced net. The sliced unfolded car crash APN model is shown in Fig. 10, places named *ExecutedCrisis* and *ExecutedCrisisreporting* together with transition named *sendreport* are sliced away. From the initial marking of Car Crash APN Model 36 states are reachable, whereas sliced car crash APN model has 27 reachable states. The resultant sub net is sufficient to verify both properties (see proof in Theorem 1).

6 Evaluation

In this section, we evaluate our slicing algorithm with the existing benchmark case studies. We measure the effect of slicing in terms of savings of the reachable

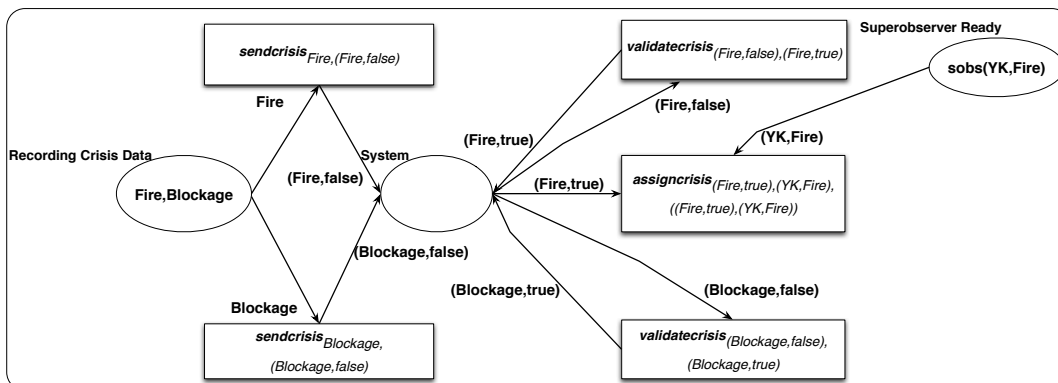


Fig. 10. Sliced and unfolded car crash APN model

state space, as the size of the state space usually has a strong impact on time and space needed for model checking. Instead of presenting case studies where our methods work best, it is equally interesting to see where it gives an average or worst case results, so that we will present a comparative evaluation on the benchmark case studies.

To evaluate our approach, we made the following assumptions:

- Evaluation procedure is independent of the temporal properties. In general, it is not feasible to determine which places correspond to the interesting properties. Therefore, we generated slices for each place in the given APN model (later, we take some specific temporal properties about the APN models under observation) .
- We abandoned the initially marked places (we follow [13] to assume that there are not interesting properties concerning to those places).

Let us study the results summarized in the table.1, the first column shows different APNs models under observation. Based on the initial markings, total number of states is shown in the second column. Best reduction and average reduction (shown in the third and fourth column) refers to the biggest and an average achievable reduction in the state space among all possible properties. In the fifth column total number of places is given, for the properties related to these places, our slicing does not reduce the number of states. Finally, the structure of APN models under observation is given. Results clearly indicate the significance of slicing; the proposed *APNSlicing* algorithm can alleviate the state space even for some strongly connected nets.

To show that the state space could be reduced for the practically relevant properties. Let us take some specific examples of the temporal properties from the APN models shown in table.2 and compare the reduction in terms of states by applying the *APNSlicing* algorithm. For the *Daily Routine of two Employees and Boss APN model*, for an example, we are interested to verify that: “Every time

Table 1. Results on different APN models

<i>System</i>	<i>T.States</i>	<i>Bst.Reduct</i>	<i>Avg.Reduct</i>	<i>Worst Places</i> <i>no reduction</i>	<i>N.Type</i>
<i>Complaint Handling</i>	2200	98.01%	40.54%	2	<i>Weak.Connect</i>
<i>Divide & Conquer</i>	117863	99.09%	14.22%	1	<i>Weak.Connect</i>
<i>Bevarage Vending</i> <i>& Machine</i>	136	80.14%	02.15%	2	<i>Weak.Connect</i>
<i>Daily Routine of 2</i> <i>Employees & Boss</i>	80	93.75%	86.12%	zero	<i>Str.Connect</i>
<i>Simple Protocol</i>	1861	95.91%	39.01%	1	<i>Str.Connect</i>
<i>Producer Consumer</i>	372	0.00%	0.00%	5	<i>Str.Connect</i>

the boss does not schedule a meeting, he will be at home eventually". Formally, we can specify the property:

$\varphi_1 = \mathbf{G}(NM \Rightarrow \mathbf{FB}1)$, where "NM" (resp. B1) means "place NM (resp. B1) is not empty".

For a *Producer Consumer APN model* an interesting property could be to verify that: "Buffer place is never empty". Formally, we can specify the property:

$$\varphi_2 = \mathbf{G}(|Buffer| > 0).$$

And for a *Complaint Handling APN model*, we are interested to verify: "All the registered complaints are collected eventually". Formally, we can specify the property:

$\varphi_3 = \mathbf{G}(RecComp \Rightarrow \mathbf{F}CompReg)$, where "RecComp" (resp. CompReg) means "place RecComp (resp. CompReg) is not empty".

Table 2. Results with different properties concerning to APN models

<i>System</i>	<i>Tot.States</i>	<i>Property</i>	<i>Crit.Place(s)</i>	<i>Percent.Reduction</i>
<i>Daily Routine of 2</i> <i>Employees & Boss</i>	80	φ_1	{NM,B1}	75.00%
<i>Producer Consumer</i>	372	φ_2	{Buffer}	0.00%
<i>Complaint Handling</i>	2200	φ_3	{RecComp,RegComp}	50.54%

Let us study the results summarized in the table shown in table. 2, the first column represents the system under observation whereas in the second column total number of states are given based on the initially marked places. The third column refers the property that we are looking for the verification. In the fourth

column, places are given that are considered as criterion places, and for those places slices are generated. The fifth column represents the number of states that are reduced (in percentage) after applying *APNSlicing* algorithm.

We can draw the following conclusions from the evaluation results such as:

- The choice of the place can have an important influence on the reduction effects (As the basic idea of slicing is to start from the criterion place and iteratively include all the *non-reading transitions* together with their input places. The less *non-reading transitions* attached to the criterion place, the more reduction is possible).
- Reduction can vary with respect to the net structure and markings of the places (The slicing refers to the part of a net that concerns to the property, remaining part may have more places and transitions that increase the overall number of states. If slicing removes parts of the net that expose highly concurrent behavior, the savings may be huge and if the slicing removes dead parts of the net, in which transitions are never enabled then there is no effect on the state space).
- For certain strongly connected nets slicing may produce a reduced number of states (For all the strongly connected nets that contain *reading transitions* slicing can produce noteworthy reductions).
- Slicing produces best results for not strongly connected nets (By definition work-flow nets are not strongly connected and since they model work flows, slicing can effectively reduce such nets).

7 Conclusion and Future Work

In this work, we developed an Algebraic Petri net reduction approach to alleviate the state space explosion problem for model checking. The proposed work is based on slicing. The presented slicing algorithm (*APNSlicing*) for Algebraic Petri net guarantees that by construction the state space of sliced net is at most as big as the original net. We showed that the slice allow verification and falsification if Algebraic Petri net is slice fair. Our results show that slicing can help to alleviate the state space explosion problem of Algebraic Petri net model checking.

The future work has twofold objectives; first to implement the proposed slicing construction in ALPiNA (Algebraic Petri net analyzer) a symbolic model checker [5]. As discussed in the section 3.1, we are using the same unfolding approach for APNs as ALPiNA. Obviously, this will reduce the effort in terms of implementation. Secondly, we aim to utilize the sliced net when verifying the evolutions of the net. Slicing can serve as a base step to identify those evolutions that do not require re-verification.

References

1. D. Buchs, S. Hostettler, A. Marechal, A. Linard, and M. Risoldi. Alpina: A symbolic model checker. *Springer Berlin Heidelberg*, pages 287–296, 2010.
2. J. R. Burch, E. Clarke, K. L. McMillan, D. Dill, and L. J. Hwang. Symbolic model checking: 1020 states and beyond. In *Logic in Computer Science, 1990. LICS '90, Proceedings., Fifth Annual IEEE Symposium on*, pages 428–439, 1990.
3. J. Chang and D. J. Richardson. Static and dynamic specification slicing. In *In Proceedings of the Fourth Irvine Software Symposium*, 1994.
4. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8:244–263, 1986.
5. S. Hostettler, A. Marechal, A. Linard, M. Risoldi, and D. Buchs. High-level petri net model checking with alpina. *Fundamenta Informaticae*, 113(3-4):229–264, Aug. 2011.
6. K. Jensen. Coloured petri nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties*, volume 254 of *Lecture Notes in Computer Science*, pages 248–299. Springer Berlin Heidelberg, 1987.
7. Y. I. Khan. A formal approach for engineering resilient car crash management system. Technical Report TR-LASSY-12-05, University of Luxembourg, 2012.
8. L. Lamport. What good is temporal logic. *Information processing*, 83:657–668, 1983.
9. W. J. Lee, H. N. Kim, S. D. Cha, and Y. R. Kwon. A slicing-based approach to enhance petri net reachability analysis. *Journal of Research Practices and Information Technology*, 32:131–143, 2000.
10. M. Llorens, J. Oliver, J. Silva, S. Tamarit, and G. Vidal. Dynamic slicing techniques for petri nets. *Electron. Notes Theor. Comput. Sci.*, 223:153–165, Dec. 2008.
11. A. Rakow. Slicing petri nets with an application to workflow verification. In *Proceedings of the 34th conference on Current trends in theory and practice of computer science, SOFSEM'08*, pages 436–447, Berlin, Heidelberg, 2008. Springer-Verlag.
12. A. Rakow. *Slicing and Reduction Techniques for Model Checking Petri Nets*. PhD thesis, University of Oldenburg, 2011.
13. A. Rakow. Safety slicing petri nets. In S. Haddad and L. Pomello, editors, *Application and Theory of Petri Nets*, volume 7347 of *Lecture Notes in Computer Science*, pages 268–287. Springer Berlin Heidelberg, 2012.
14. W. Reisig. Petri nets and algebraic specifications. *Theor. Comput. Sci.*, 80(1):1–34, 1991.
15. K. Schmidt. T-invariants of algebraic petri nets. *Informatik-Bericht*, 1994.
16. A. Valmari. The state explosion problem. In *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets*, pages 429–528, London, UK, UK, 1998. Springer-Verlag.
17. M. Weiser. Program slicing. In *Proceedings of the 5th international conference on Software engineering, ICSE '81*, pages 439–449, Piscataway, NJ, USA, 1981. IEEE Press.

This work has been supported by the National Research Fund, Luxembourg, Project MOVERE, ref.C09/IS/02.

A Proposal for the Modeling of Organizational Structures and Agent Knowledge in MAS

Lawrence Cabac, David Mosteller, Matthias Wester-Ebbinghaus

University of Hamburg
Faculty of Mathematics, Informatics and Natural Sciences
Department of Informatics
{cabac,2mostell,wester}@informatik.uni-hamburg.de
<http://www.informatik.uni-hamburg.de/TGI>

Abstract. One of the most important tasks when developing multi-agent systems (MAS) is to determine the overall organizational structure of the system. In this paper we present a service-oriented perspective on the organizational structure of MAS and we present modeling techniques and tools for supporting this perspective. We pursue a model-driven approach and a tight integration between various models on the one hand and between the models and the generated code on the other hand. In particular, we combine ontology modeling and organization structure modeling in a way that we can easily generate the initial content of agent knowledge bases in the form of FIPA semantic language (SL) fragments (depending on what positions the agents occupy in the context of the organizational structure). In addition, this allows the agents to reason about and to communicate about their organizational embedding using the same ontology.

Keywords: RENEW, MULAN, PAOSE, Petri nets, multi-agent systems, model-driven development, organizational structure

1 Introduction

The modeling of the fundamental organizational structure is one of the central tasks during the development of a multi-agent system (MAS) [9]. While agents are considered autonomous in their actions, they are also supposed to fulfill certain functions in relation to the purpose of the overall multi-agent application (MAA). A wide spectrum of approaches for organizing multi-agent systems exists [15] and some of them are quite sophisticated in drawing inspiration from organizing principles of social systems (including multiple organizational modeling dimensions like social structures, tasks, social interactions, norms etc., cf. [1,8]). We argue that at the core of most of these approaches lies the determination of an organizational structure in terms of agent functions and agent dependencies based on functional dependencies. This concerns the questions, *which agents* are required / allowed to do *what* (responsibilities / abilities) and *to whom* they can

refer for help in certain cases (support / delegation). Basically, this is a service-oriented perspective on agent relationships. Agents offer functional services to other agents and in turn require the services of other agents in order to fulfill some of their own functionality.

We apply this functional and service-oriented perspective for the design of the basic organizational structure of a MAS in our PAOSE approach (**P**etri net-based **A**gent- and **O**rganization-oriented **S**oftware **E**ngineering, <http://www.paose.net>). It provides a general basis for MAS organization that can be extended if necessary.¹ We have presented our PAOSE approach on previous occasions and we have particularly elaborated on the model-driven nature of PAOSE in [6]. Our multi-agent platform MULAN/CAPA [16,22] tightly combines model and code as it is based on a fusion of high-level Petri nets and Java. This allows us to model / implement all processes as directly executable Petri nets. In addition, we use UML-style modeling techniques for development where we need a more declarative perspective than is offered by Petri nets.

In this paper, we specifically refer to the part of PAOSE that is concerned with modeling organizational structures in terms of agent roles and service dependencies between roles. This part relies on ontology modeling as we explicate the concepts used for organizational structures as an ontology. This has the additional benefit that we can easily translate an organizational structure model into multiple initial knowledge bases for multiple agents (depending on what positions the agents occupy in the organizational structure). The content of the knowledge bases is generated in FIPA semantic language (<http://www.fipa.org>), which provides the technical basis for agents to reason about and to communicate about their organizational embedding. Compared with our previous work presented in [6,7], we present a considerable rework including new tools. Our revision basically takes care of a better and tighter integration between the tools used as well as between the models and the generated code.

In Section 2 we provide an overview of role and service (dependency) modeling in the MAS field and motivate our own approach. In Section 3, we present our concrete models and the supporting tools. We place our contribution in the context of our development process PAOSE and introduce the agent framework MULAN/CAPA. We also describe how the tools fit into the model-driven nature of our PAOSE approach. Section 4 gives an example of our tools in use, demonstrated in a concrete application scenario. We close with a short summary and some aspects of future research and development that builds upon the results presented in this paper in Section 5.

¹ For example, we have developed the SONAR model [17,18] for multi-agent teamwork support, where we use a more elaborate model of functional service dependencies between agents based on task delegation structures and a behavior-based notion of service refinement.

2 Organizational Structures of Multi-Agent Systems

In this section we elaborate on our conceptual approach to modeling organizational structures in terms of agent roles and service dependencies. We motivate the use of the two core concepts of *roles* and *services* in the context of related work.

2.1 Modeling Agent Roles and Service Dependencies

The interest in establishing organizational structures in a MAS has always been an important part of agent research. One can argue that it is an important part of software design in general (although the architecture metaphor is more established than the organization metaphor). However, in the case of MAS this topic becomes even more imperative. Artificial social agents are regarded as very sophisticated software components with complex knowledge and reasoning mechanisms that often only offer a limited visibility. Consequently, high-level system perspectives are necessary, in which one can abstract from agent-internal details and still comprehend the system on a more abstract level.

The concept of a role has been used extensively in this context and has been established as one of the core concepts of agent-oriented software design [19]. Rights and responsibilities are associated with roles independently from the specific agents that will occupy the roles. Consequently, this leads to a certain degree of predictability and controllability of global MAS behavior without knowing anything about the agents' internals. Examples of bringing the concept of roles to use (cf. [1]) is to enable as well as constrain agent behavior in terms of (1) which roles belong together to a common group context (allowing acquaintance and communication between group members), (2) defining which roles are expected to be associated with which goals, tasks and necessary capabilities and (3) which roles are supposed to take part in which conversations in which way.

Basically, all these efforts boil down to the abstract question what an agent occupying a specific role is supposed to do just because of it taking on that role. We are mainly interested in an explication of a functional perspective on roles and role relationships. Of special interest is the specification of functionality of roles occupants in the context of the wider multi-agent application and the dependencies that exist between different role occupants. Thus, we apply a service-oriented perspective on agent roles: Which roles are associated with the provision of which services and on which other services are they dependent? We are aiming at a rather minimalistic model of agent roles and their relationships in terms of service dependencies that can be enriched with more sophisticated concepts if needed (e.g. goal/task hierarchies, conversation guidelines).

2.2 Related Work

Not only in agent-oriented approaches to software development the modeling of component dependencies is one of the major challenges. One main problem (also applying to some of the approaches for role-based specifications mentioned

above) is that dependencies are often hidden underneath quite complex specifications. Ensel and Keller summarize Gopal [13] in the following way: “However, the main problem today lies in the fact that dependencies between services and applications are not made explicit, thus making root cause and impact analysis particularly difficult” [10, p. 148]. Therefore our motivation is to gain the ability to explicitly model dependencies for MAS and our choice is to model component dependencies (agent dependencies) in terms of roles and service dependencies. The actual dependencies between running agents then result from the roles they occupy.

In the context of the different approaches to software development there exist various ways of handling component dependencies. Some of them are restricted to managing service dependencies by utilizing declarative service descriptions, i.e. using XML [10, p. 148], [24]. From our point of view the more promising approach consists in making use of diagram-based methods.

The most obvious benefit lies in the incomparably better visualization of diagram-supported models over declarative service descriptions. This was identified as a central issue, taking up the above mentioned citation by Ensel and Keller again. On the one hand, the diagram is the means to make the dependencies explicit [3] instead of an implicit declaration located in the configuration files of the (distributed) components as it is for example the case in OSGI service descriptions [24]. An explicit representation of the dependencies is of special value during the design phase for a developer / administrator. On the other hand, the capabilities of model transformation are given in both possibilities to describe dependencies as model-based and as declarative descriptions. A similar approach was taken in [26] for Web Services and BDI-Agents. Service dependencies are specified in the model domain and tool support is realized as a Rational Software Modeler extension. “Dependencies between the various components are modeled at the PIM-level and two-way model transformations help us to ensure interoperability at the technical level and consistency at the PIM-level” [26, p. 114]. There are other efforts, which mainly address specification of dependencies between agents and Web Services (e.g. [14]) whereas our work is focused on agent relations.

Most software developing methodologies contain a technique for modeling some kind of dependencies between their components. The Tropos methodology distinguishes four kinds of dependencies between agents, from hard dependencies (resource) to soft ones (soft-goal). Silva and Castro [23] display how Tropos dependency relations can be expressed in UML for real time systems. Ferber et al. [11] show how the organizational structure of an agent-based system can be modeled using the AGR technique. One of the proposed diagrams, the organizational structure diagram, shows roles, interactions and the relations between roles and interactions. This diagram is comparable to the Roles/Dependencies diagram.

In Gaia Zambonelli et al. [25] focus strongly on the organizational modeling. One of the important models is the service model. Our Roles/Dependencies diagram can be regarded as an implementation of the Gaia service model. However,

Gaia does not recognize hierarchical roles. Padgham and Winikoff [21] explicitly model acquaintances in Prometheus. But from these models they do not derive any agent (role) dependencies. Roles are not modeled in Prometheus, instead the focus lies on agents. The system model in Prometheus gives a good overview of the system comparable with the overview of the Roles/Dependencies diagram. It is much more detailed but does not explicitly show any dependencies except the interaction protocols or messages that connect agents. The structure of the system model reflects the one of the acquaintances model.

In the following, we introduce our approach for a minimalistic (but extensible) comprehension of organizational structures of MAS in terms of role descriptions and role dependencies based on service relationships. Our previous work covered details on the conceptual side of modeling the basic organizational structure of MAS, introducing modeling techniques [7,5,4] and tools [6]. In our current work we improve the methods and tools by putting an even stronger focus on the model-driven nature of our approach. We pursue a tighter integration of different tools and to minimize the gap between the models and the code generated from the models. One specific benefit of our approach lies in the fact that the meta-model for organizational structures is expressed in the agents' language – i.e. as an agent ontology. Thus, the agents are able to communicate and reason about their own organizational structures.

3 Role/Dependency Tool Support for Model-Driven Development in PAOSE

In the following we point out how the integration of the conceptual basis we introduced in the previous section is established in our MULAN framework and the PAOSE development process. We introduce two types of diagrams, namely for ontology modeling and for roles/dependencies modeling. They support the discussed features in a clear and intuitive way, making use of well-known constructs from UML. We also present our tool solution to support our model-driven development approach. All our PAOSE tools are realized as plugins for the high-level Petri net tool RENEW (<http://www.renew.de>).² They extend RENEW with modeling techniques that are not based on Petri nets.

3.1 The PAOSE Development Process

This section puts the subsequent work into the context of the PAOSE approach, which aims at the development of MULAN applications. The approach focuses on aspects of distribution, concurrency and model-driven development. The framework MULAN offers the basic artifacts and structuring for the application. Its four layered architecture features as basic artifacts the communication infrastructure, the agent platforms, the agents and the agent internals (protocols, decision components and knowledge bases). With the exception of the communication

² RENEW also provides the virtual machine that executes MULAN applications.

infrastructure, all artifacts are implemented as Java Reference nets. CAPA extends the MULAN architecture with FIPA-compliant communication features, providing inter-platform (IP-based) agent communication. Also the MULAN applications (MAA) are – similar to the MULAN/CAPA framework – implemented in Java Reference nets and Java. They are executed, together with the MULAN/CAPA framework, in the RENEW virtual machine. While the implementation in Java Reference nets introduces concurrency for MULAN applications, the CAPA extension enables the agents to run in distributed environments.

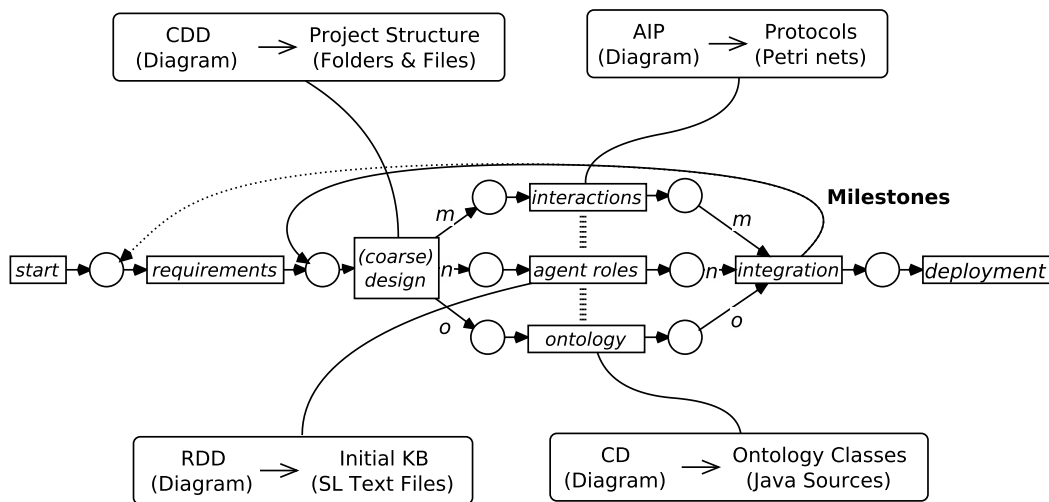


Figure 1. The PAOSE development process and techniques. Modified from [3, p. 133]

The organization of MAS can be explicitly modeled using model-driven techniques [6], as described in the following sections. However, in addition to the organizational structure of the MAA, we apply the agent-oriented view onto the organizational structure of the development team through the metaphor of the *multi-agent system of developers* [2]. The metaphor provides the perspective that human participants of the development team form an organization, similar to agents in an MAA, and their collaborative efforts constitute the development process, similar to the MAA process. During development the responsibilities for the diverse tasks are distributed among the developers, which allows for concurrent and distributed collaboration as well as explicit identification of dependencies between the team participants. In the previous sections we motivated a service-oriented composition of MAS based on roles and service dependencies. Here we argue that developers dependencies result from the organizational structure and the application's dependencies. These dependencies are also reflected in the PAOSE development process, which consists in iterative repetitions of specific fundamental steps of design and implementation, as shown in Figure 1. The figure depicts a simplified Petri-net process of the PAOSE design cycle.

A project starts with the requirements analysis resulting in a coarse design of the overall structure of the MAA. The coarse design identifies essential roles and interactions of the organization. It is used to generate the initial structure (development artifacts) of a project. The main step of an iteration consists of three tasks of modeling and implementation. These are the modeling of interactions, agent roles and ontologies, as well as generating sources from the models and refining the implementation. The integration of the resulting artifacts completes an iteration. In the diagram annotations refer to modeling techniques, which are utilized to carry out a corresponding task and the artifacts, which are generated from the design models. Taking up the aforementioned view on the organization of a development team, the completion of an iteration requires the synchronized, collaborative effort of the participants.

In the context of this work we introduce a technique and a tool for the modeling of agent roles. To this end, we utilize the ontology model used in PAOSE as a meta-model. In this sense the following section describes how our integration approach essentially applies ontology concepts for the design of a new modeling technique and a corresponding tool – in this case the modeling of organizational structures of MAA.

3.2 Integration Approach

Within our development process we apply a few presumptions. The approach taken relies on two fundamental ideas. These are the support for the development process by making use of methods from model-driven development (MDD) and the tightening of the integration of models (diagrams), generated code and serialized representations. This leads to a threefold integrative approach that is illustrated in Figure 2 and that we discuss in the following.

Integration encompasses three parts: (1) an ontology including multiple concepts, (2) the code generated from the ontology and (3) the serialized representations of concept instances in FIPA Semantic Language (SL) format [12].

Ontologies are modeled using a light-weight technique called *Concept Diagram*³. The concepts defined in the ontology are transformed into Java classes, one class for each concept. Instances of these classes (ontology objects) can be extracted into SL-formatted text. Through an SL parser the serialized SL text representations can be used to instantiate Java ontology objects in the reverse direction. Consequently, we utilize three tools for these three tasks: (1) the *ConceptDiagramModeler*, (2) the *OntologyGenerator* and (3) the *SL parser*.

This basic integration approach described so far has several benefits. The development process takes on a model-driven approach, which allows for the specification of ontological concepts in a graphical notation. Concept Diagrams are very similar to the widely-used UML Class Diagrams. They are quite intuitively comprehensible and easy to manage. Manipulation of attributes can be carried out directly in the diagram. Additionally, by making use of code

³ An example of a Concept Diagram will be discussed in the context of defining a knowledge base format in the following section (3.3).

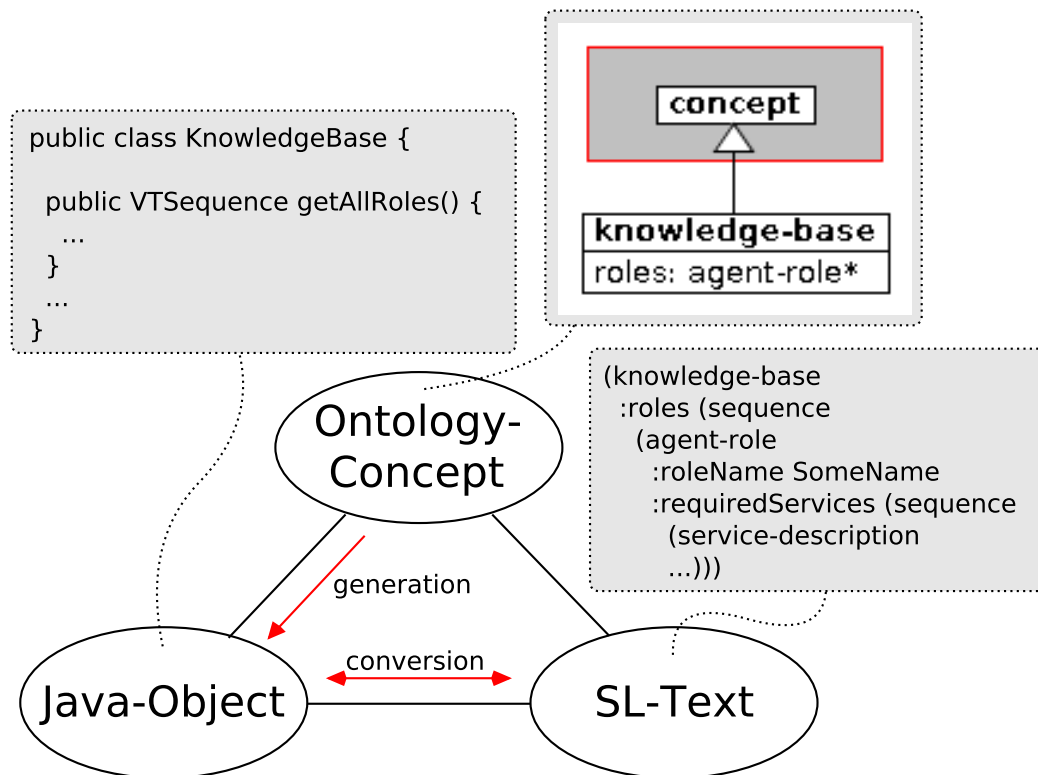


Figure 2. The three-part basic model of agent knowledge

generation and the bi-directional conversion between Java objects and SL text representations for concept instances, the integration of the different representations is very tight, i.e. transformation is transparent to a user. By using SL for the text representations of ontology objects we employ an agent-comprehensible format, as MULAN agents use SL text representations for message encoding. In addition, our experience has indicated that SL text is also better human-readable in comparison to an equivalent XML representation and shows a lower overhead.

In the following, we show how we apply this method in the case of modeling service dependencies and agent knowledge.

3.3 Concept Diagrams for Role and Knowledge Base Concepts

The previous section provides a general overview of our model-driven approach based on the integration of multiple representations. Now we describe how the three basic parts (ontology, Java code, SL text representation) are applied in the case of modeling as well as establishing organizational structures in multi-agent applications (MAA). The model-driven approach starts with the specification of an ontology encompassing organizational concepts (roles, services, protocols) and concepts necessary for the generation of agent knowledge from organizational structure models (knowledge base as an aggregation of role definitions resulting

from a mapping between agents and roles, in which multiple roles can be assigned to each agent). The ontology we use is shown in Figure 3 as a Concept Diagram. It is created with the ConceptDiagramModeler.

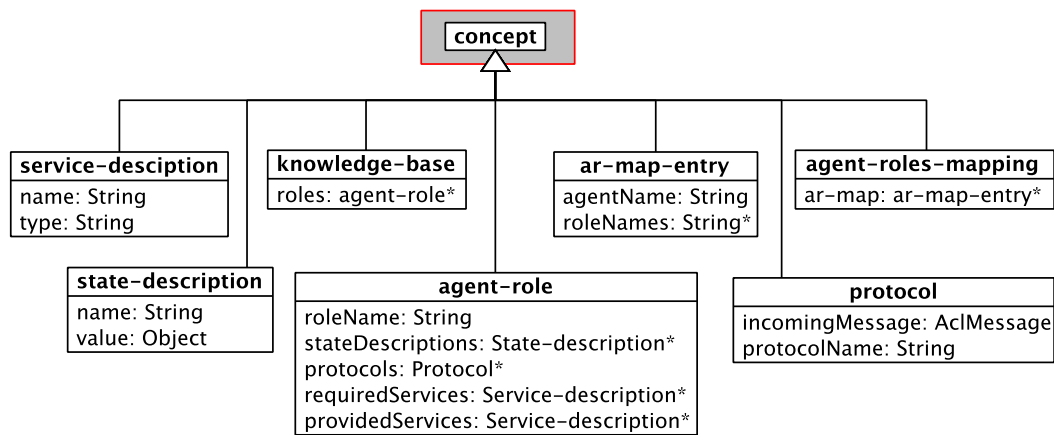


Figure 3. A Concept Diagram for agent knowledge base concepts

The Concept Diagram serves in a twofold way. First, it defines all the content types of the agent communication in an application. Second, it serves as a meta-model for the tools that handle the modeled contents.

From here on, we rely on further tool support for code generation and conversion between the different representations of concepts and concept instances. We use the OntologyGenerator (based on Velocity, <http://velocity.apache.org>) and the SL parser provided by the MULAN framework. Ontology modeling in terms of Concept Diagrams and code generation from these models is already a part of the PAOSE development process (cf. [3, p. 173]). Thus, the approach described here for handling organizational structures and agent knowledge fits neatly into the context of our wider work.

Basically, Figure 3 can be regarded as capturing the ontology for knowledge bases of MULAN agents (the schema of a knowledge base). It illustrates the modeling technique of Concept Diagrams in terms of inheritance and the use of concepts for the definition of other concepts (this could also be modeled via associations between different concepts).

Besides capturing the ontology for knowledge bases, the ontology is also used by the AgentRoleModeler tool presented in the next subsection. Java ontology classes that are generated by the OntologyGenerator tool are specializations of generic ValueTuple (VT) and KeyValueTuple (KVT) classes. VT and KVT structures are the root interfaces of an implementation of the FIPA SL.

As mentioned above, by using an SL parser, ontology objects can be instantiated from their SL string representations. This is a feature that lies at the heart of creating *agent instances* from knowledge base patterns (because knowledge-base is a concept in the diagram from Figure 3 and thus each knowledge base

as a whole has an SL representation). Ontology classes provide *getters*, *setters* and convenience methods for operations on the data structures. The Ontology-Generator tool is integrated into the build environment of the MULAN framework and the SL parser can be used on the fly in a running MULAN MAA. All in all, this supports our ambition of realizing a tight integration of different models, tools and code.

Using the knowledge base ontology shown in Figure 3 the following section explains how we model roles and dependencies in multi-agent applications.

3.4 Roles/Dependencies Diagrams

Roles and role dependencies are modeled with the AgentRoleModeler tool. The corresponding Roles/Dependencies Diagrams combine notations from Class Diagrams and Communication Diagrams. The tool is embedded in our model-driven development approach. The content of Roles/Dependencies Diagrams (Figure 5) is based on the concepts that were already defined in the ontology from Figure 3. Because of this, the AgentRoleModeler tool allows for the generation of knowledge base descriptions in FIPA SL from Roles/Dependencies Diagrams using the knowledge base ontology from Figure 3 as a meta-model. Thus the concepts from the Concept Diagram reappear as stereotypes in the Roles/Dependencies Diagram. The knowledge base descriptions resulting from a Roles/Dependencies Diagram are used as patterns for the initialization of agent instances in the MULAN multi-agent framework.

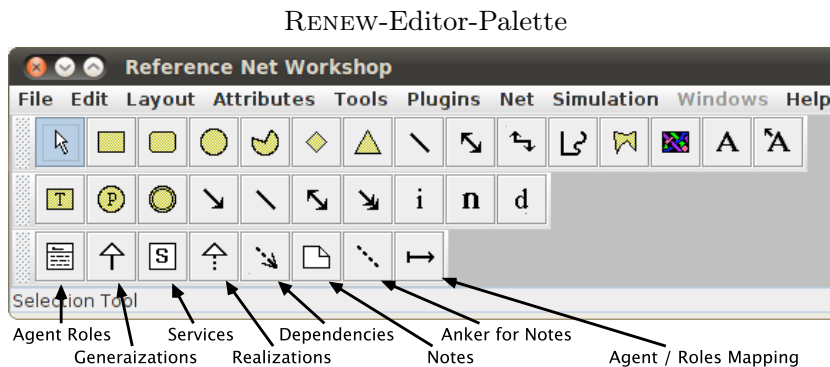


Figure 4. The RENEW-Editor-Palette

The AgentRoleModeler is a drawing plugin for RENEW and adds a custom palette for drawing elements of Roles/Dependencies Diagrams as shown in Figure 4 (the AgentRoleModeler palette is shown at the bottom, under RENEW's standard palettes). The graphical representation of Roles/Dependencies Diagram elements is displayed in Figure 5. The nodes of Roles/Dependencies Diagrams (roles and services) contain the text in FIPA SL format, specifying the corresponding attributes of the element. For a compact representation all drawing elements can be collapsed to a smaller view. This provides a very compact

and high-level view of an organizational structure in terms of roles and role dependencies based on service dependencies. Expanding the drawing elements allows manipulation of their attributes.

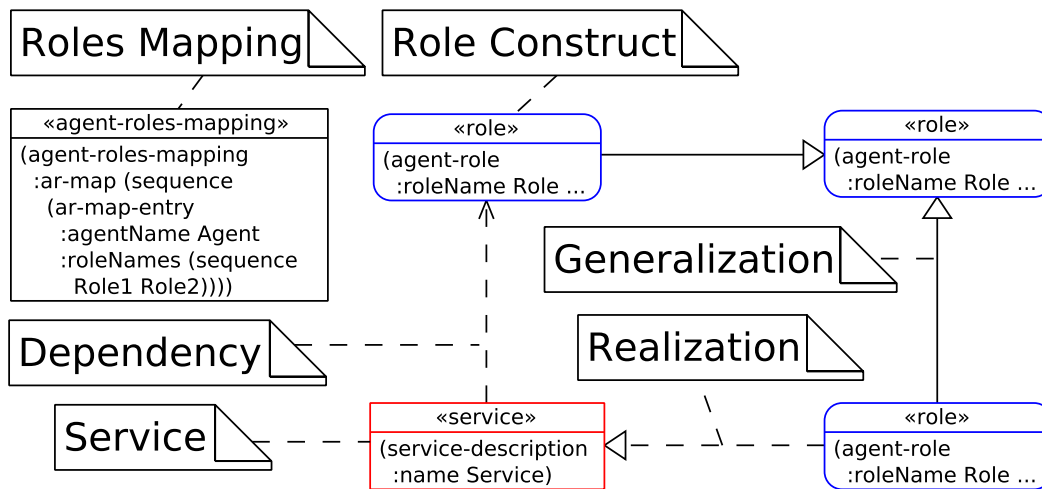


Figure 5. Constructs of a Roles/Dependencies Diagram.

Following Figure 3, the knowledge base of a MULAN agent contains an arbitrary number of agent role descriptions, depending on which roles the agent occupies. The attributes of agent roles (besides having a role name) are basically of three different types: (1) service dependencies, (2) protocol triggers and (3) state descriptions. Such (initial) knowledge base content can be generated from Roles/Dependencies Diagrams. Required and provided services of a role (i.e. hard dependencies) are shown explicitly as independent service nodes and offer / use associations connected to role nodes⁴. Protocol triggers are key-value tuples that define, which conversation protocol (value) an agent should initiate in reaction to incoming messages of a certain message pattern (key). They are not represented in a Roles/Dependencies Diagram as explicit nodes but are inserted directly into the corresponding role description. Further, state descriptions for a role may contain any kind of key-value tuples that shall serve as initial knowledge for role occupants. In addition to this flat specification of role dependencies, it is also possible to define inheritance relationships between roles. This introduces hierarchical relationships.

A Roles/Dependencies Diagram contains exactly one node that defines an *agent-role mapping*. Basically, this node serves to define *agent types* in terms of what roles a specific agent type should encompass. For each such agent type, a pattern in FIPA SL can be generated that serves as the basis for the initial knowledge of instantiated agents of that type.

⁴ Refer to [7] for a discussion of our view on hard and soft dependencies.

4 Application

Aforementioned, we motivate the modeling of organizational structures in MAS. Our approach to modeling organizations grounds on a ontological content definition, namely the three-part basic model of agent knowledge. We introduced the three-part basic model in Section 3.2. We showed how the concrete realization of a tool for modeling organizations utilizes the ontological content definition. In this spirit the previous section introduced the notation of the Roles/Dependencies diagram and the AgentRoleModeler tool. We will now use the technique of a Roles/Dependencies diagram to model a sample application. The organizational structure is brought into the MAS by the means of generic knowledge base patterns. In the following example we demonstrate our method of extracting initial agent knowledge and structural information from the graphical model and show how they are brought into the running system. We illustrate the modeling of organizational structures and agent knowledge in sample applications.

The AgentRoleModeler tool was developed in the context of a bachelors thesis [20] at the University of Hamburg. After its completion it was used in several student projects for agent-oriented software development. In one of these projects about 20 team members worked on implementing applications for an agent-based collaboration platform. The following example from Figure 6 is taken from this project. It displays the scenario of a chat application, in which agents occur in the roles of chat senders and receivers.

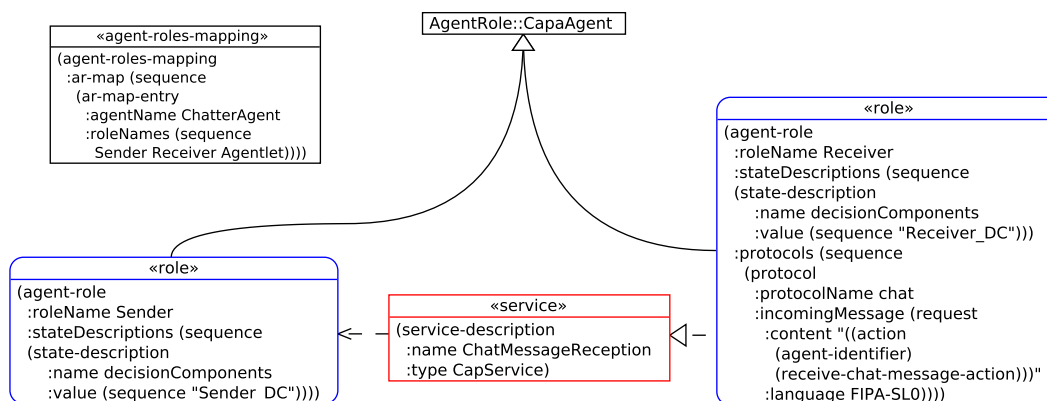


Figure 6. ARM of WebChat.

Figure 6 shows an instance of a Roles/Dependencies diagram. The model consists in the concepts that were already used in the previous section, introduced in Section 3.2 and illustrated in Figure 5. The Sender/Receiver-Scenario is an example we regularly use to demonstrate the MULAN-Framework in student projects. It consists in two roles, a Sender and a Receiver. Both roles inherit attributes from the generic CapaAgent role, thus they are specializations of this role. We will go into more detail about this later. The Sender is in possession of

a decision component `SenderDC`, which enables him to sporadically participate in conversation. He is dependent on a service (`ChatMessageReception`) allowing him to find and address chat partners. The Receiver is a role, which provides such a service, as can be seen by the realization relation between the Receiver role and the `ChatMessageReception` service. Upon receiving a chat message, the Receiver role reacts by initiating a chat protocol. The role specification formalizes reactive behavior as a protocol trigger, which can be seen on the lower right part of the above figure. A protocol trigger maps a type of message, identified by a message pattern, to a protocol. Every time a message of the defined type is received, the protocol will be triggered. The chat protocol passes chat messages to a decision component of the Receiver (`ReceiverDC`) allowing him to process the message. He can carry out internal reasoning about the conversation and decide on his further actions, such as creating a response or initiating a new conversation. The diagram constructs described up to this point make up the Roles/Dependencies model. There is a part we have not yet discussed. The agent-roles mapping construct shown on the upper left formalizes an instance specification. It determines, which agents occupy a previously defined set of roles. The agent-roles mapping in this case maps both roles to one type of agent, a `ChatterAgent`. The reason is that a `ChatterAgent` should naturally have the ability to do both, send and receive chat messages.

The example displays our notion of functional rights and responsibilities in terms of services dependencies. This specification of roles and services in form of the Roles/Dependencies diagram can be used to generate initial knowledge base contents for the agent instances dedicated to fulfill the corresponding roles. The following example focuses on the succeeding step of extracting the information required to initialize agent instances from the model.

Figure 7 shows a fragment of the Roles/Dependencies diagram that basically refers to one of the roles from the example above. The blue-bordered role figure (round corners) displays the attributes for the role name, protocol triggers and state descriptions. On the right hand side one can see a snippet of the FIPA SL code generated from the Roles/Dependencies diagram. Here, the service provided by the Receiver role (`ChatMessageReception`) is also included directly in the FIPA SL text. It can also be seen that the FIPA SL fragment contains more than one state descriptions. The additional state descriptions are inherited from the `CapaAgent` super role.

The roles and their mutual service dependencies are compiled into knowledge base patterns. The knowledge base patterns are in FIPA SL text, so they can directly be used to initialize agent instances, as they are specified in the language (ontology) of `MULAN`-agents. The example shows how this information is extracted from the model. It also shows how the model can express hierarchies of roles in terms of role specializations, enabling inheritance of attributes. Besides using the specialization relation between role constructs inside one single diagram, the mechanism implemented in the `AgentRoleModeler` tool also allows using inheritance across diagrams. This is also shown in the above figure. The `CapaAgent` role is accessed from the Roles/Dependencies diagram named `Agent-`

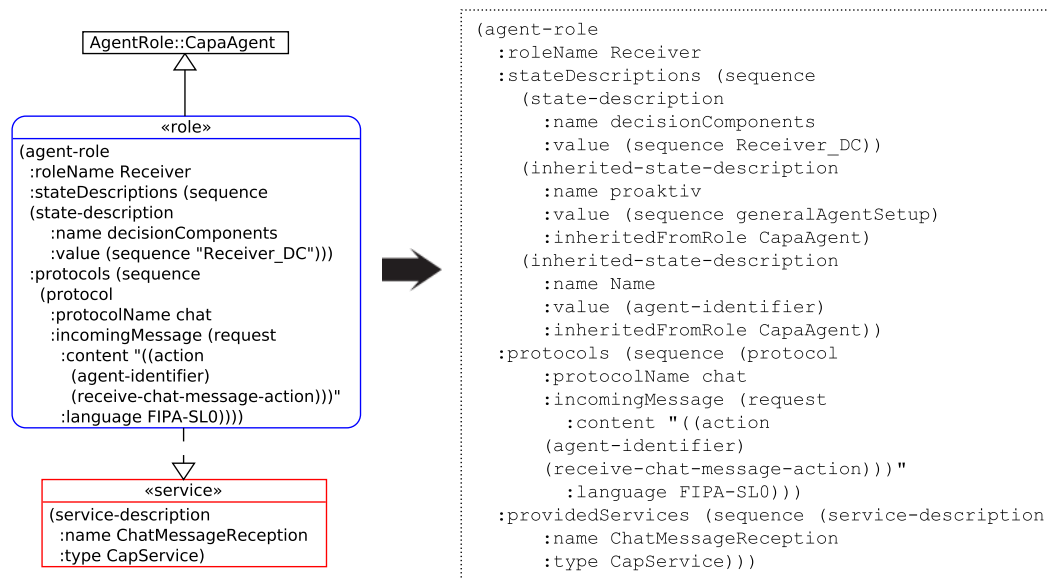


Figure 7. Role attributes and FIPA SL code generation.

Role. This is denoted by the displayed notation containing double colons. With the support for expressing specializations with the AgentRoleModeler tool it is possible to build graphical models containing hierarchies and compile them into knowledge base patterns, which allows us to project the overall organizational structure onto the MAA.

The approach for modeling basic organizational structures of MAS in terms of roles and role relationships fits neatly into the general model-driven nature of our PAOSE approach. In particular, in this case it helps to generate initial agent knowledge.

5 Conclusion

In agent-oriented software engineering and especially in the context of developing MULAN applications two essential design aspects are the modeling of the organizational structures and the initial knowledge of the agents. For the purpose of modeling these fundamental features it requires a conceptual basis as well as corresponding techniques, methods and tools.

5.1 Summary

In the context of the PAOSE approach we utilize the technique of Concept Diagrams and the OntologyGenerator tool to specify ontology concepts in order to design multi-agent applications. In the context of this paper this technique and tool is introduced together with the method of modeling agent roles and service dependencies. Furthermore we present a technique and a supporting tool

– also implemented as plugin for our IDE Renew – for a light-weight modeling of service dependencies and agent roles.

In Section 2 we elaborate on organizational concepts in MAS research and motivate our approach to modeling organizational structures. The main part of our contribution is preceded by an introduction to the PAOSE development process and the MULAN framework (Section 3.1), which constitute the context of our work. Our approach to modeling roles and dependencies is introduced in three steps. First, we introduce the three-part basic model of our approach (Section 3.2). Second, we illustrate the modeling of ontology concepts utilizing Concept Diagrams and the OntologyGenerator (Section 3.3). Third, we present the modeling of agent roles and dependencies with Roles/Dependencies Diagrams and the AgentRoleModeler (Section 3.4). The presented technique is an occurrence of the Roles/Dependencies Diagram – a Class Diagram that includes notations from Communication Diagrams for the modeling of agent role dependencies – that makes use of the Semantic Language (SL) as a description language for the agents’ initial knowledge base contents. Finally, the techniques and tools presented in the course of this contribution are demonstrated in a application scenario in Section 4.

5.2 Future Work

In the context of our current research we elaborate on generalizing the approach that was presented in this paper to a further step. The idea is not only to apply the model-driven approach for code generation, conversion and transformation of models, but to generate special purpose tools from ontology diagrams as well. A step in this direction is generalizing the AgentRoleModeler tool to a generic SLEditor tool. The UI of a current prototype is shown in Figure 8. It displays the previously introduced role description of a Receiver from the Sender/Receiver application in a nested graphical figure. The outer frame is that of the agent-role. The highlighted constructs (in gray) indicate *ValueTuples*. This representation allows for displaying any nested structure of *KeyValueTuples* and *ValueTuples*. It can be seen as an alternative view to the plain text representation in Semantic Language. Further efforts are being made to utilize an ontology – modeled with the technique of a Concept Diagram – as a meta-model to generate specialized structures and at the same time generate the modeling tools by using the generic SLEditor. We are occasionally confronted with criticism against grounding our work on an *outdated* infrastructure, because we still rely on the FIPA Semantic Language for the specification of MAA. We address this subject with our development plan on extending the SLEditor. With the generic SLEditor tool we can support the modeling of MAA using other languages for content specification, such as XML. This can be achieved by extending the ontology to support XML to express knowledge contents.

Taking up the Figure 2 from Section 3.2 the three-part basic model of agent knowledge is extended to display an XML ontology of an agent role. The Figure reveals what is required to enable this feature: an XML schema definition specifying the format of our ontologies and the methods for conversion of representa-

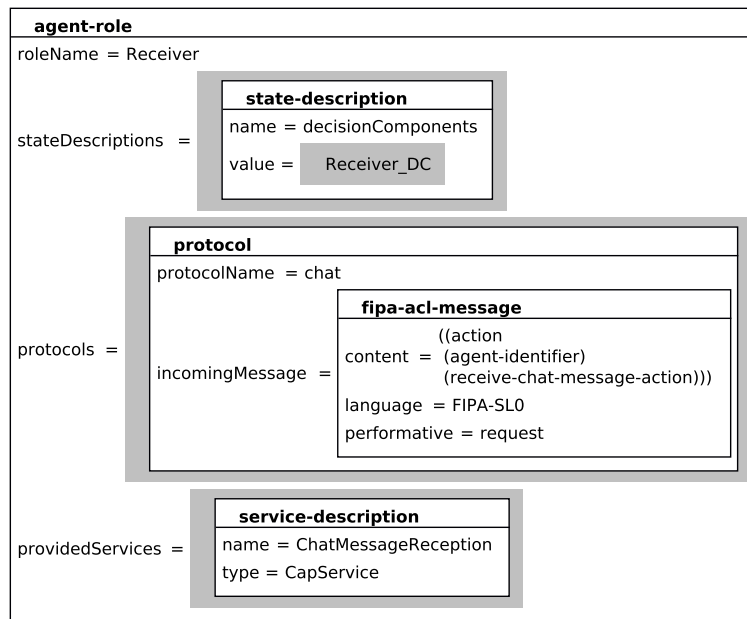


Figure 8. Alternative representation of SL content.

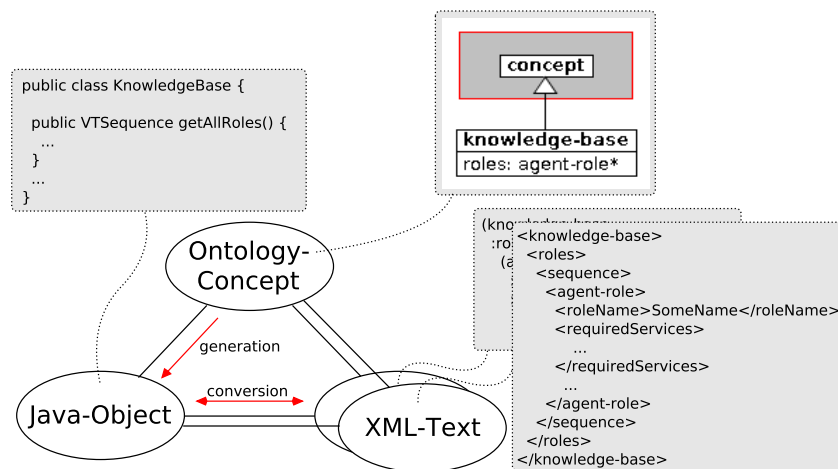


Figure 9. The three-part basic model of agent knowledge, extended to support XML for content specification.

tions between XML-Text and Java-Object. They are at this time not integrated in MULAN, but there exist standard tools that support XML conversion.

References

1. Olivier Boissier, Jomi Hübner, and Jaime Simão Sichman. Organization oriented programming: From closed to open organizations. In G. O’Hare, A. Ricci, M. O’Grady, and O. Dikenelli, editors, *Engineering Societies in the Agents World*

- VII, volume 4457 of *Lecture Notes in Computer Science*, pages 86–105. Springer, 2007.
2. Lawrence Cabac. Multi-agent system: A guiding metaphor for the organization of software development projects. In Paolo Petta, editor, *Proceedings of the Fifth German Conference on Multiagent System Technologies*, volume 4687 of *Lecture Notes in Computer Science*, pages 1–12, Leipzig, Germany, 2007. Springer-Verlag.
 3. Lawrence Cabac. *Modeling Petri Net-Based Multi-Agent Applications*, volume 5 of *Agent Technology – Theory and Applications*. Logos Verlag, Berlin, 2010.
 4. Lawrence Cabac, Ragna Dirkner, and Daniel Moldt. Modeling with service dependency diagrams. In Daniel Moldt, Ulrich Ultes-Nitsche, and Juan Carlos Augusto, editors, *Proceedings of the 6th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems, MSVVEIS-2008, In conjunction with ICEIS 2008, Barcelona, Spain, June 2008*, pages 109–118, Portugal, 2008. INSTICC PRESS.
 5. Lawrence Cabac, Ragna Dirkner, and Heiko Rölke. Modelling service dependencies for the analysis and design of multi-agent applications. In Daniel Moldt, editor, *Proceedings of the Fourth International Workshop on Modelling of Objects, Components, and Agents. MOCA'06*, number FBI-HH-B-272/06 in Report of the Department of Informatics, pages 291–298, Vogt-Kölln Str. 30, D-22527 Hamburg, Germany, June 2006. University of Hamburg, Department of Informatics.
 6. Lawrence Cabac, Till Döriges, Michael Duvigneau, Daniel Moldt, Christine Reese, and Matthias Wester-Ebbinghaus. Agent models for concurrent software systems. In Ralph Bergmann and Gabriela Lindemann, editors, *Proceedings of the Sixth German Conference on Multiagent System Technologies, MATES'08*, volume 5244 of *Lecture Notes in Artificial Intelligence*, pages 37–48, Berlin Heidelberg New York, 2008. Springer-Verlag.
 7. Lawrence Cabac and Daniel Moldt. Support for modeling roles and dependencies in multi-agent systems. In Michael Köhler-Bußmeier, Daniel Moldt, and Olivier Boissier, editors, *Organizational Modelling, International Workshop, OrgMod'09. Proceedings*, Technical Reports Université Paris 13, pages 15–33, 99, avenue Jean-Baptiste Clément, 93 430 Villetaneuse, June 2009. Université Paris 13. Preproceedings available online at <http://www.informatik.uni-hamburg.de/TGI/events/orgmod09/#proceedings>.
 8. Luciano Coutinho, Jaime Sichmann, and Olivier Boissier. Modelling dimensions for agent organizations. In Virginia Dignum, editor, *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, pages 18–50. Information Science Reference, 2009.
 9. Virginia Dignum. The role of organization in agent systems. In Virginia Dignum, editor, *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, pages 1–16. Information Science Reference, 2009.
 10. Christian Ensel and Alexander Keller. An approach for managing service dependencies with xml and the resource description framework. *Journal of Network and Systems Management*, 10:147–170, 2002.
 11. Jacques Ferber, Olivier Gutknecht, and Fabien Michel. From agents to organizations: An organizational view of multi-agent systems. In Paolo Giorgini, Jörg P. Müller, and James Odell, editors, *Agent-Oriented Software Engineering IV, 4th International Workshop, AOSE 2003, Melbourne, Australia, July 15, 2003, Revised Papers*, volume 2935 of *Lecture Notes in Computer Science*, pages 214–230. Springer-Verlag, 2003.
 12. Foundation for Intelligent Physical Agents (FIPA). FIPA SL Content Language Specification. <http://www.fipa.org/specs/fipa00008/index.html>, 2002.

13. R. Gopal. Layered model for supporting fault isolation and recovery. In *Network Operations and Management Symposium, 2000. NOMS 2000. 2000 IEEE/IFIP*, pages 729–742. IEEE, 2000.
14. C. Hahn, S. Jacobi, and D. Raber. Enhancing the interoperability between multi-agent systems and service-oriented architectures through a model-driven approach. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, volume 2, pages 415–422. IEEE, 2010.
15. Bryan Horling and Victor Lesser. A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, 19(4):281–316, 2005.
16. Michael Köhler, Daniel Moldt, and Heiko Rölke. Modelling the structure and behaviour of Petri net agents. In J.M. Colom and M. Koutny, editors, *Proceedings of the 22nd Conference on Application and Theory of Petri Nets 2001*, volume 2075 of *Lecture Notes in Computer Science*, pages 224–241. Springer-Verlag, 2001.
17. Michael Köhler-Bußmeier, Daniel Moldt, and Matthias Wester-Ebbinghaus. A formal model for organisational structures behind process-aware information systems. volume 5460 of *Lecture Notes in Computer Science*, pages 98–115. Springer-Verlag, 2009.
18. Michael Köhler-Bußmeier, Matthias Wester-Ebbinghaus, and Daniel Moldt. Generating executable multi-agent system prototypes from sonar specifications. In Nicoletta Fornara and George Vouros, editors, *11th Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems, COIN@Mallow 2010*, pages 82–97, 2010.
19. Xinjun Mao and Eric Yu. Organizational and social concepts in agent oriented software engineering. In James Odell, Paolo Giorgini, and Jörg Müller, editors, *Agent-Oriented Software Engineering V*, volume 3382 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin / Heidelberg, 2005.
20. David Mosteller. Entwicklung eines Werkzeugs zur Modellierung der initialen Wissensbasen und Rollen-Abhängigkeiten in Multiagentenanwendungen im Kontext von PAOSE / MULAN. Bachelor's thesis, University of Hamburg, Department of Informatics, December 2010.
21. Lin Padgham and Michael Winikoff. *Developing Intelligent Agent Systems : A Practical Guide*. Wiley Series in Agent Technology. Chichester [u.a.] : Wiley, 2004. isbn:0-470-86120-7, Pages 225.
22. Heiko Rölke. *Modellierung von Agenten und Multiagentensystemen – Grundlagen und Anwendungen*, volume 2 of *Agent Technology – Theory and Applications*. Logos Verlag, Berlin, 2004.
23. Carla T. L. L. Silva and Jaelson Castro. Modeling organizational architectural styles in UML: The tropos case. In Oscar Pastor and Juan Sánchez Díaz, editors, *Anais do WER02 - Workshop em Engenharia de Requisitos*, pages 162–176, 11 2002.
24. Andre L.C. Tavares and Marco Tulio Valente. A gentle introduction to OSGi. *SIGSOFT Softw. Eng. Notes*, 33:8:1–8:5, August 2008.
25. Franco Zambonelli, Nicholas Jennings, and Michael Wooldridge. Developing multi-agent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3):317–370, 2003.
26. Ingo Zinnikus, Gorka Benguria, Brian Elvesæter, Klaus Fischer, and Julien Vayssière. A model driven approach to agent-based service-oriented architectures. In Klaus Fischer, Ingo Timm, Elisabeth André, and Ning Zhong, editors, *Multi-agent System Technologies*, volume 4196 of *Lecture Notes in Computer Science*, pages 110–122. Springer Berlin / Heidelberg, 2006.

Mining Declarative Models Using Time Intervals

Jan Martijn van der Werf ^{*}, Ronny S. Mans ^{**}, and Wil M.P. van der Aalst

Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{ j.m.e.m.v.d.werf, r.s.mans, w.m.p.v.d.aalst }@tue.nl

Abstract. A common problem in process mining is the interpretation of the time stamp of events, e.g., whether it represents the moment of recording, or its occurrence. Often, this interpretation is left implicit. In this paper, we make this interpretation explicit using time intervals: an event occurs somewhere during a time window. The time window may be fine, e.g., a single point in time, or coarse, like a day. As each event is related to an activity within some process, we obtain for each activity a set of intervals in which the activity occurred. Based on these sets of intervals, we define ordering and simultaneousness relations. These relations form the basis of the discovery of a declarative process model describing the behavior in the event log.

Keywords: Process mining, time intervals, concurrency theory, declarative process models

1 Introduction

Information systems of today collect large amounts of data. For example, banks are saving information about the granting of mortgages and loans, insurance companies are saving information concerning the handling of claims, and hospitals are saving the actions taken to treat patients. Many of the recorded data concern events which have been performed in the context of a certain business process. For each event, different aspects are stored, for example, the activity and case for which the event is raised, its type, and when it has been raised. Process mining [1] aims to extract process knowledge from these recorded events to discover, monitor and improve the actual processes supported by these systems.

The information about when the event occurred, for example using the order in which events are recorded, or its recorded timestamp is used to discover, monitor and check the control flow of processes. The implicit assumption many of the process mining algorithms make is that if two events are recorded consecutively, e.g. one is recorded

^{*} supported by the PoSecCo project (project no. 257129), partially co-funded by the European Union under the Information and Communication Technologies (ICT) theme of the 7th Framework Programme for R&D (FP7).

^{**} supported by the Dutch Technology Foundation STW, applied science division of NWO and the Technology Program of the Ministry of Economic Affairs.

before the other, or the timestamp of the first is before the latter, they occurred consecutively. However, in many cases, this assumption may not hold, as systems implement log recording differently. So, although time information is recorded, it can be interpreted in many different ways.

One interpretation of the timestamp is that it is the time on which the event actually occurred. More likely, many systems implement this as the time on which the event is recorded. Other systems implement logging using a queue system, i.e., the event is placed in the queue, and then written. Thus, if two events occur at the same time, their timestamp may differ as they are written consecutively.

A second problem of timestamps is their scale. On the one hand, a too fine time scale introduces causality that in reality does not exist. For example, consider an information system consisting of many different components each with their own logging mechanism. To construct the process of the information system, the recordings of each component need to be combined in a single event log. As a result, one needs to ensure that all components have the same time. On the other hand, a too coarse time scale may falsely introduce concurrency. For example, if the time scale is in days, the order of activities executed on the same day cannot be discovered.

A third problem lies in the reliability of the time information. For example, the order based on timestamps of events is more reliable if the same timestamp generator is used. Thus, timestamps of events in the same component are more reliable than when the events are recorded by different components. Another source of unreliability is whether time information depends on user input, such as calendars, or if it is generated by the system.

As a result, one should always first check the order in which events occur. One way to resemble this is to use intervals for event occurrences instead of single timestamps. This allows to change the time scale from a very fine scale, such as single points, to very coarse time scales, such as days. For example, an event that occurred on timestamp '2013/04/12 12:24:36.3', can be seen as an interval of a single point, or, if the required time scale under consideration is in days, it can be seen as an event that occurred on April 12, 2013, i.e., in the interval '2013/04/12 0:00' - '2013/04/12 23:59:59'.

Process mining focuses on the extraction of process knowledge. Whereas process knowledge mainly focuses on the level of activities, systems recordings are on an event level, which is not necessarily the same level, as several events may be raised for the same activity, for example when the activity started or completed. Thus, to be able to reason on the level of activities, events should be combined into activities. Aggregation of events in activities can be done in many ways, such as the life-cycle of activities [1]. Depending on the aggregation, each activity may occur several times, each in its own time interval, resulting in a set of time intervals for each activity.

In this paper, we want to make the time intervals in which an activity occurs explicit. Based on a set of intervals for each activity, we reason about which relations can be inferred. For example, do two activities occur simultaneously or do they occur sequentially. As we use intervals instead of single time points, many activities may occur concurrently. Procedural languages, like Petri nets [3], model explicitly the order in which activities occur. For example, places in a Petri net are used to control choices and to reduce the degree of concurrency in a model. As a consequence, concurrency needs

to be modelled explicitly, rather than being a language primitive. In a declarative approach, all events may be executed concurrently, unless it is prohibited by constraints. Therefore, we choose a declarative modelling language instead, called *Declare* [2], and show how a declarative model can be derived from the intervals induced by the timestamps of the events.

This paper is structured as follows. In Sec. 2 we introduce the basic notions used throughout the paper. Sec. 3 discusses the role of intervals within an event log. These events and their intervals can be mapped onto activities in many different ways, as shown in Sec. 4. Next, in Sec. 5, we define simultaneousness and causality relations on sets of intervals. Sec. 6 presents a method to build a declarative model based on these interval relations. Last, Sec. 7 concludes the paper.

2 Basic Notions

Let S be a set. The powerset of S is denoted by $\mathcal{P}(S) = \{S' \mid S' \subseteq S\}$. We use $|S|$ for the number of elements in S . Two sets U and V are *disjoint* if $U \cap V = \emptyset$. We denote the cartesian product of two sets S and T by $S \times T$. On a cartesian product we define two projection functions $\pi_1 : S \times T \rightarrow S$ and $\pi_2 : S \times T \rightarrow T$ such that $\pi_1((s, t)) = s$ and $\pi_2((s, t)) = t$ for all $(s, t) \in S \times T$. We lift the projection function to sets in the standard way.

A binary relation R from S to T is defined by $R \subseteq (S \times T)$. For $(x, y) \in R$, we also write xRy . For a relation $R \subseteq (S \times T)$, the inverse relation R^{-1} is defined as $R^{-1} = \{(y, x) \in (T \times S) \mid xRy\}$. A relation R is called a function if xRy and xRz implies $y = z$ for all $x \in S$ and $y, z \in T$. It is called a binary relation over S if $R \subseteq (S \times S)$. A binary relation R is reflexive if xRx for all $x \in S$. It is transitive if xRy and yRz implies xRz for all $x, y, z \in S$. It is reflexive if $(x, x) \in R$ for all $x \in S$, and irreflexive if $(x, x) \notin R$ for all $x \in S$. Relation R is symmetric if xRy implies yRx for all $x, y \in S$ and asymmetric if xRy implies $\neg yRx$ for all $x, y \in S$. The relation is antisymmetric if xRy and yRx imply $x = y$ for all $x, y \in S$. The transitive closure of a binary relation R is defined as the smallest relation R^+ such that xR^+y if either $x = y$, or xR^+z and zRy for some $z \in S$.

A binary relation R over some set S is an *equivalence relation* if it is reflexive, symmetric and transitive. A transitive, irreflexive binary relation is called a *strict order*. It is a *preorder*, denoted by (S, R) , if R is reflexive and transitive. A preorder is a *partial order* if (S, R) is also antisymmetric. A partial order is called a *total order*, if in addition also xRy or yRx for all $x, y \in S$.

A *sequence* over S of length $n \in \mathbb{N}$ is a function $\sigma : \{1, \dots, n\} \rightarrow S$. If $n > 0$ and $\sigma(i) = a_i$ for $i \in \{1, \dots, n\}$, we write $\sigma = \langle a_1, \dots, a_n \rangle$. The length of a sequence is denoted by $|\sigma|$. The sequence of length 0 is called the *empty sequence*, and is denoted by ϵ . The set of all finite sequences over S is denoted by S^* . Let $\nu, \gamma \in S^*$ be two sequences. *Concatenation*, denoted by $\sigma = \nu; \gamma$ is defined as $\sigma : \{1, \dots, |\nu| + |\gamma|\} \rightarrow S$, such that for $1 \leq i \leq |\nu|$: $\sigma(i) = \nu(i)$, and for $|\nu| + 1 \leq i \leq |\nu| + |\gamma|$: $\sigma(i) = \gamma(i - |\nu|)$.

Given a set S and a, possibly infinite set $T \subseteq \mathbb{R}$, a function $f : S \rightarrow T \times T$ is called an *interval function* if $\pi_1(f(a)) \leq \pi_2(f(a))$ for all $a \in S$.

2.1 Event Logs

For each user action on the system, an *event* is raised. An event records its type, for which activity it has been raised, for which *case* or business process instance, when it was raised, by whom, and the data inserted by the user. Such a recording is called an *event log* [1]. The set of all possible events, i.e., the event universe is denoted by \mathcal{E} . Similarly, we denote the case, attribute and value universes by \mathcal{C} , \mathcal{A} and \mathcal{V} , respectively, such that $\mathcal{E}, \mathcal{C}, \mathcal{A} \subseteq \mathcal{V}$ and \mathcal{E}, \mathcal{C} and \mathcal{A} are pairwise disjoint. We assume $A \subseteq \mathcal{V}$ to be the (possibly infinite) set of activities.

Definition 1 (Event log). An event log is a 3-tuple $L = (C, E, \#)$ where

- $C \subseteq \mathcal{C}$ is a set of case identifiers in the event log;
- $E \subseteq \mathcal{E}$ is a set of event identifiers in the log;
- $\# : \mathcal{A} \times (C \cup E) \rightarrow \mathcal{P}(\mathcal{V})$ is an attribute mapping.

For an attribute $n \in \mathcal{A}$ we write $\#_n(\cdot)$ as a shorthand for $\#(n, \cdot)$. The following attributes are always defined:

- Each event belongs to exactly one case and each case has at least one event, denoted by the mandatory attribute $\text{case} \in \mathcal{A}$, i.e., for all events $e \in E$, a case $c \in C$ exists with $\#_{\text{case}}(e) = \{c\}$, and for all $c \in C$ an event $e \in E$ exists with $\#_{\text{case}}(e) = \{c\}$;
- Each event belongs to some activity, denoted by the mandatory attribute $\text{act} \in \mathcal{A}$, i.e., for all events $e \in E$ an activity $a \in A$ exists such that $\#_{\text{act}}(e) = \{a\}$;
- An event may record the time it was recorded using the timestamp attribute $\text{time} \in \mathcal{A}$, i.e., for all events $e \in E$ we have $\#_{\text{time}}(e) = \{t\}$ for some timestamp $t \in T$, where T resembles the set of timestamps.

3 Intervals in Event Logs

There are many techniques for discovering a process model out of an event log. An extensive overview of available process discovery techniques can be found in [24]. Some examples are the alpha miner [4], the ILP miner [27] and the declarative miner [17]. In many discovery methods, events are considered to be instantaneous: they occur at a single point in time. However, in many information systems, such as electronic patient records, or financial statements, only a date is recorded. Consequently, even if events are considered to occur instantaneously, if they are observed within the same interval, the only conclusion to be drawn is that these occurred simultaneously.

The more coarse the chosen time scale (e.g., days, weeks or months), the more events will occur concurrently. Another consequence of a more coarse time scale is that events occur in some time window, rather than occurring at a single moment in time. It is important to note that there are some techniques which do not consider events to be instantaneous. That is, the authors of [15], exploit the fact that activities take time, i.e. each activity has a start and complete event. As a result, parallelism can be detected explicitly. Two activities are considered to occur in parallel if there is at least one case in which the activities overlap in time. In [20], the authors consider the execution of an activity as a time interval based on a starting and ending event. Parallelism is detected

Table 1. Example event log, time scale in days

date	events	date	events
7-1-2013	(1, A)	14-1-2013	(3, G) (4, A)
8-1-2013	(1, B), (1, E)	15-1-2013	(4, F), (5, A), (6, A)
9-1-2013	(2, A), (1, G)	16-1-2013	(4, G), (5, D)
10-1-2013	(2, E), (2, C), (3, A)	17-1-2013	(5, G), (6, F)
11-1-2013	(2, G), (3, D)	18-1-2013	(6, G)

by identifying two executions in which one activity occurs before the other one, and the other way around. The work described in [21] is comparable to [20], which presents a different control-flow discovery algorithm based on the notion of time intervals. All the aforementioned techniques only use one notion for determining intervals for activities and whether they overlap. In this paper, we study the case where activities occur in multiple intervals within the same execution.

Consider the events presented in Tbl. 1 showing for each day the events that occurred. For each event, its case and activity are recorded. The time stamps of these events are in days, e.g., event (1, *B*) occurred on January 1, 2013, as well as event (1, *E*). Based on this information, we cannot infer any order between *B* and *E*, the only fact that can be inferred is that these events occurred simultaneously.

As the time scale is relatively coarse, a first analysis of this event log would be the degree of concurrency. We can build a graph that depicts the intervals on a time scale, as shown in Fig. 1(a). Based on this graph, we derive a *concurrency relation* $I \subseteq \mathcal{E} \times \mathcal{E}$, such that $a I b$ if and only if a and b occur within the same time interval. This results in a graph as depicted in Fig. 1(b), where the dashed and solid edges together represent the relation I . For readability, the self loops have been omitted. Note that (*A*, *G*) is an edge in the graph, while no case exists in which activities *A* and *G* occur simultaneously. Therefore, we can partition the relation I into two relations I_S and I_G such that $a I_S b$ if and only if $\#_{\text{case}}(a) = \#_{\text{case}}(b)$, and I_G analogously. In Fig. 1(b), the edges of relation I_S are solid, the edges of relation I_G are dashed. Similarly, the concurrency relation is not transitive with respect to the event log: even though (*B*, *E*) and (*E*, *C*) are edges in the graph, *B*, *C* and *E* never occur simultaneously in any case.

Whereas in Fig. 1(b) an absolute time window is taken, one could also choose to map each event to a relative interval, e.g. the respective day from the start of the day, as shown in Fig. 1(c). To allow such abstractions, we introduce the notion of an *event interval mapping function* that maps each event onto a time interval.

Definition 2 (Event interval mapping function). Let $L = (C, E, \#)$ be an event log. A function $m_L : E \rightarrow T \times T$ is an event interval mapping function for L if it is an interval function. The default interval mapping function $D_L : E \rightarrow T \times T$ of L is defined by $D_L(e) = (\#_{\text{time}}(e), \#_{\text{time}}(e))$ for all $e \in E$.

Based on the event interval mapping function, two notions of concurrency can be observed: one based on the whole event log, called the *concurrency relation*, and one based on the individual executions: the *simultaneousness relation*. Thus, the simultaneousness relation I for an event log L can be defined as the events that occur in the same interval defined by some interval mapping function.

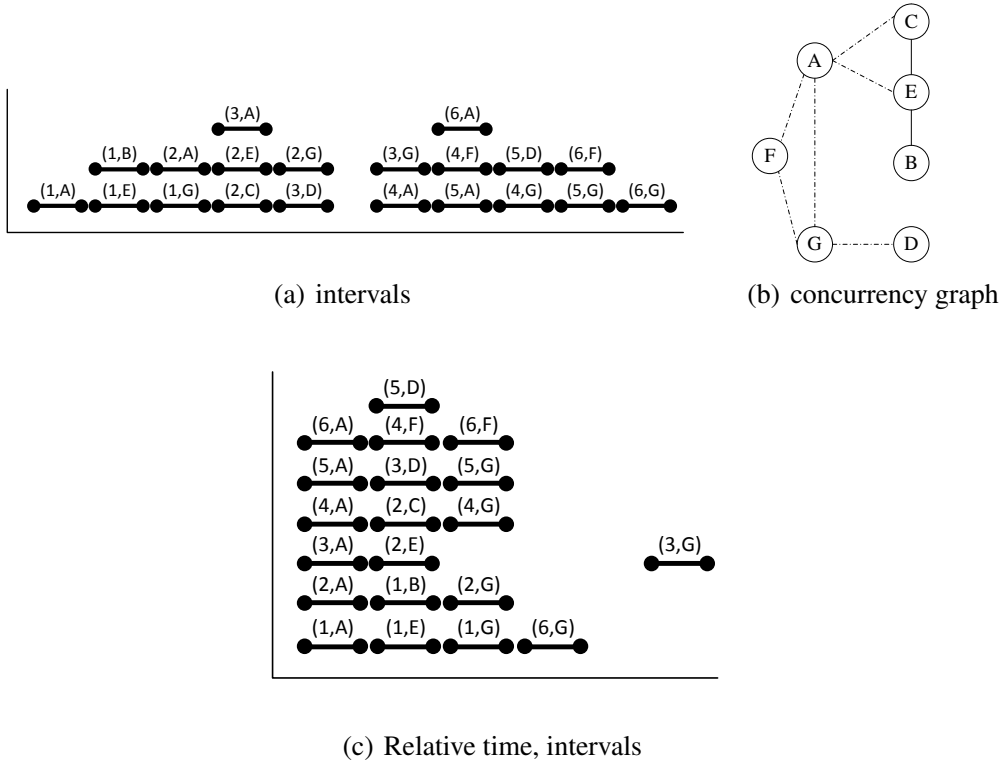


Fig. 1. Intervals of Tbl. 1

Definition 3 (Concurrency, simultaneousness relation). Let L be an event log, and m a corresponding event interval mapping function. Its concurrency relation $\bar{I}_m \subseteq E \times E$ is defined by $a \bar{I}_m b$ iff $\pi_1(m(a)) \leq \pi_2(m(b))$ and $\pi_1(m(b)) \leq \pi_2(m(a))$ for $a, b \in E$. Its simultaneousness relation $I_m \subseteq E \times E$ is defined by $a I_m b$ iff both $a \bar{I}_m b$ and $\#_{case}(a) = \#_{case}(b)$ for $a, b \in E$.

In the literature, the graph imposed by the concurrency relation is called the *interval graph* [11, 16]. Following [11], we can define an ordering relation $>$ that is defined by $a > b$ iff $\pi_2(m(a)) < \pi_1(m(b))$, stating that b “wholly occurs after” a . Relation $>$ is called an *interval order* [28, 29], as proven in [11].

Definition 4 (Interval order). A binary relation R over some set S is an interval order if $a R b$ and $c R d$ imply $a R d$ or $c R b$ for all $a, b, c, d \in S$.

Using intervals in concurrency is not new. For example, Janicki and Koutny [14] show that the notion of interval orders naturally follows from a basic assumption on concurrency: “the observer can state that one event preceded another event, or that two events occurred simultaneously”. The authors show that for finite event logs, events can be interpreted as intervals on a discrete time scale. The authors introduce a model as a set of relations defining (weak) causalities, commutativity and synchronisation. An observation is called a *history* of a model if the relations induced by the observation coincide with the relations of the model.

In [6], Allen defines a set of assertions and properties based on time intervals: “before”, “equal”, “meets”, “overlaps”, “during”, “starts” and “finishes”. Based on these predicates, the authors introduce the assertion “occurs” with two variables: an event and an interval. This approach is often used in the area of artificial intelligence to reason over time using logic programming [7, 22].

4 Activities as Sets of Intervals

The interval mapping function on event logs introduced in the previous section induces an interval order on the events in the event log. In this way, approaches like in [9, 14] are directly applicable on this interval mapping function. These approaches mainly focus on a single run of a system: each event occurs exactly once. However, process mining mainly focuses on the analysis of the process implied by the activities for which the events in the event log occurred.

Different events for the same activity may indicate that the activity has been executed several times. Or, if an event represents the different stadia of some life cycle of activities, like a start and complete type, multiple events occur for the same activity. In [19] an approach is given for identifying pairs of events which denote the start and end of an activity. Thus, a single execution involves multiple occurrences of activities with some duration. Therefore, we search for new relations such that we can describe the relations on activity level, rather than on the level of events.

One way to lift the interval functions from events to activities is by defining a relation based on the interval order. Similar to the concurrency and simultaneousness relation, one would obtain two relations \bar{R} and R such that

$$\begin{aligned} a\bar{R}b &\Leftrightarrow \exists e_1, e_2 \in E : \#_{\text{act}}(e_1) = a \wedge \#_{\text{act}}(e_2) = b \wedge e_1 > e_2 \\ aRb &\Leftrightarrow \exists e_1, e_2 \in E : \#_{\text{case}}(e_1) = \#_{\text{case}}(e_2) \wedge \#_{\text{act}}(e_1) = a \wedge \#_{\text{act}}(e_2) = b \wedge e_1 > e_2 \end{aligned}$$

In fact, using the default event interval mapping of an event log, relation R coincides with the weak order relation of [25], which allows us to construct a relation set [26] based on intervals. In this paper, we will focus on behavioural relations based on the interval in which an activity is executed.

Although the above relation \bar{R} is transitive, it abstracts away from the observed sequences in the event log. As activities may have multiple occurrence intervals, it is not an interval function. Therefore, we need to generalize the interval function to sets of intervals.

Definition 5 (Generalized interval function). *Given a set S and a, possibly infinite set $T \subseteq \mathbb{R}$, a function $f : S \rightarrow \mathcal{P}(T \times T)$ is called a generalized interval function if $x \leq y$ for all $(x, y) \in f(a)$ and $a \in S$.*

A generalized interval function can define a large set of small intervals, or a small set of large intervals. We call this the *granularity* of the interval function. Given any generalized interval function, we can define its most fine granular interval function, i.e., each point is its own interval, and the most coarse granular interval function, i.e., the conjunction of all intervals.

Table 2. Event log of a single case

Act.	Type	Time	Act.	Type	Time
A	start	1	B	start	9
B	start	2	D	complete	10
B	complete	3	B	complete	11
C	start	4	E	complete	12
A	complete	5	D	start	13
C	complete	6	F	start	14
D	start	7	D	complete	15
E	start	8	F	complete	16

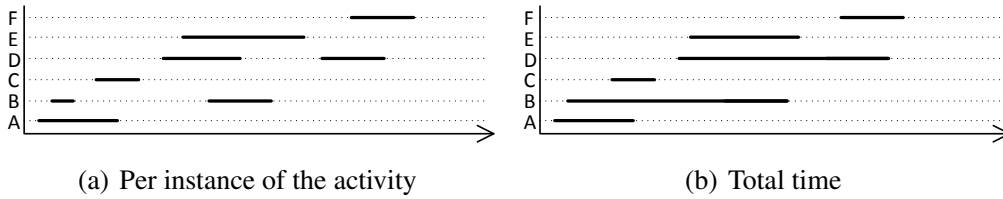


Fig. 2. Possible occurrence intervals of Tbl. 2

Definition 6 (Finest and coarsest interval functions). Let $f : S \rightarrow \mathcal{P}(T \times T)$ be a generalized interval function. Its finest interval function, denoted by $f \downarrow : S \rightarrow \mathcal{P}(T \times T)$, is defined by

$$f \downarrow (s) = \{(t, t) \mid \exists (x, y) \in f(s) : x \leq t \leq y\}$$

The coarsest interval function of f , denoted by $f \uparrow : S \rightarrow \mathcal{P}(T \times T)$, is defined by:

$$f \uparrow (s) = \{ (\min\{\pi_1(f(s))\}, \max\{\pi_2(f(s))\}) \}$$

Consider as an example the event log shown in Tbl. 2 representing the events of a single case. In this example, the time scale is defined as hours since the start of the execution. Many different ways exists to map these events to a generalized interval function on activities.

Two example mappings are given in Fig. 2. In the first example, the start and complete events of each activity are used to define the different intervals, whereas in the second example the very first start event of the activity defines the begin of the interval, and the very last complete event of the activity the end of the interval. Observe that in Fig. 2(a) activities B and C have no overlap, whereas in Fig. 2(b) these activities do have overlap.

In general, an event log records many different executions. Therefore, we map each execution to its own activity interval function. This results in an activity interval mapping for an event log.

As each event belongs to a single activity, we require that an activity interval mapping defines a unique interval for each event in the event log. On the other hand, as an activity may be represented by multiple occurrences, multiple events may be related to the same activity interval.

Definition 7 (Activity interval mapping). Let $L = (C, E, \#)$ be an event log with corresponding event interval mapping m , let A be a set of activities of L , and let $T \subseteq \mathbb{R}$ be the time scale. The function $G : C \times A \rightarrow \mathcal{P}(T \times T)$ is called an activity interval mapping iff

- each event has a unique corresponding interval, i.e.,

$$\begin{aligned} \forall e \in E : & (\exists I \in G(\#_{case}(e), \#_{act}(e)) : m(e) \subseteq I) \\ & \wedge (\forall I, J \in G(\#_{case}(e), \#_{act}(e)) : (m(e) \subseteq I \wedge m(e) \subseteq J) \implies I = J) \end{aligned}$$

- each interval has at least one event occurrence, i.e.,

$$\forall a \in A, c \in C, I \in G(c, x) : \exists e \in E : \#_{case}(e) = c \wedge \#_{act}(e) = a \wedge m(e) \subseteq I$$

The default activity interval mapping of an event log L , denoted by $\bar{L} : C \times A \rightarrow T \times T$, is defined by:

$$\bar{L}(c, a) = \{m(e) \mid \exists e \in E : \#_{case}(e) = c \wedge \#_{act}(e) = a\}$$

Many different interval functions can be defined for an event log. As the next corollary shows, such activity interval mappings are related, as intervals may be combined into larger intervals, or split into several smaller intervals. It is simple to see that given some activity interval mapping, its coarsest interval function is also an activity interval mapping. Further, the finest interval function of the minimal activity interval mapping is contained in the finest interval function of the activity interval mapping.

Corollary 8. Given an event log L with corresponding event interval mapping m and activity interval function G . Let A be the set of activities in L . Then (1) $G \uparrow$ is an activity interval mapping, (2) $\bar{L} \downarrow \subseteq G \downarrow$ and (3) $\pi_1(G \uparrow(a)) \leq \pi_1(\hat{L} \uparrow(a))$ and $\pi_2(G \uparrow(a)) \geq \pi_2(\bar{L} \uparrow(a))$ for all activities $a \in A$.

5 Relations on Interval Sets

In general, a generalized interval function does not define any interval order. Consequently, approaches like in [6, 14] cannot be used to determine causality and similarity relations. In this section, we derive such notions based on the generalized interval function.

5.1 Notions of Simultaneousness

In an interval order two intervals are unrelated if one does not wholly occur after the other, and vice versa. With sets of intervals, different degrees of simultaneousness can be defined.

The weakest form of simultaneousness is when two elements have some overlapping intervals. For example, in the intervals shown in Fig. 3(a), activities A and B have some intervals that overlap. Note that the relation is not transitive, as shown in the same figure. We say an element s is *dependent simultaneous* with some other element t if for every interval of s , an overlapping interval of t exists. Thus, everytime s is started, t will be started as well, whereas if t occurs, s does not have to occur. If s always overlaps with t , we say they are *strongly dependent*.

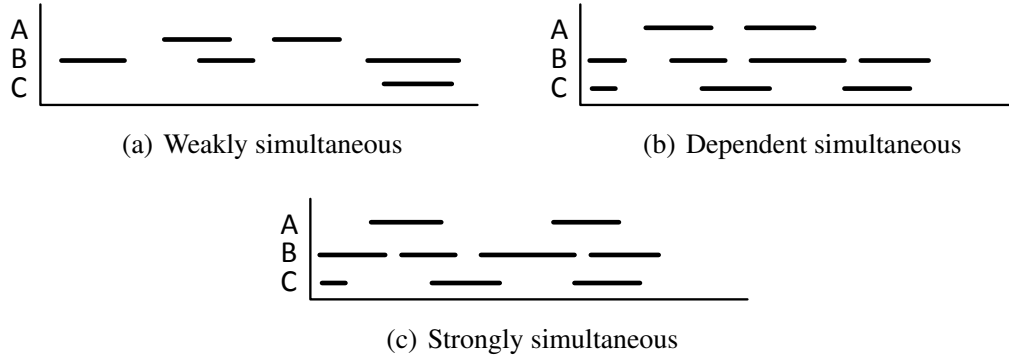


Fig. 3. Simultaneousness relations

Definition 9 (Simultaneousness). Let f be a generalized interval function over some set S . Let $s, t \in S$. Then:

- s and t are weakly simultaneous, denoted by $s \leftrightarrow t$, if s and t share some interval, i.e., $\exists I \in f(s), J \in f(t) : I \cap J \neq \emptyset$;
- s is dependently simultaneous with t , denoted by $s \rightrightarrows t$, if always if s occurs, then t occurs in the same interval, i.e., $\forall I \in f(s) : \exists J \in f(t) : I \cap J \neq \emptyset$;
- s and t are strongly simultaneous, denoted by $s \rightleftarrows t$, if s and t always overlap, i.e., $s \rightrightarrows t$ if and only if $s \Rightarrow t$ and $t \Rightarrow s$.

Consider again Fig. 3. In Fig. 3(a) we have $A \leftrightarrow B$ and $B \leftrightarrow C$ but not $A \leftrightarrow C$, in Fig. 3(b) we have $A \rightrightarrows B$ as every interval of A overlaps with some interval of B , $B \rightrightarrows C$ as each interval of B overlaps with some interval of C and $A \leftrightarrow C$ but not $A \rightrightarrows C$ as not every interval of A overlaps with an interval of C . Last, in Fig. 3(c) we have $A \rightleftarrows B$, $B \rightleftarrows C$ and $A \rightrightarrows C$ but not $A \rightleftarrows C$, as every interval of A overlaps some interval of C but not vice versa.

Based on their definitions, it is trivial to see that strong simultaneousness implies dependent simultaneousness which in turn implies weak simultaneousness.

Corollary 10. Let f be a generalized interval function over some set S , and let $s, t \in S$. Then (1) $s \rightrightarrows t \wedge f(s) \neq \emptyset \implies s \leftrightarrow t$, and (2) \rightleftarrows and \leftrightarrow are symmetric and reflexive.

As shown in Fig. 3, none of these relations is transitive. Consequently, we cannot obtain equivalence classes based on the intervals. As the relation \rightleftarrows is symmetric and reflexive, it can be used as a *dependence relation* over the set of activities, which allows us to use Mazurkiewicz trace theory [10] for e.g. synthesis and to check completeness of event logs.

5.2 Notions of Causality

Fishburn showed in [11], that given an interval function f , any order $>$ with $x > y$ iff $\pi_2(f(x)) < \pi_1(f(y))$, i.e., that the interval of x is wholly after the interval of y , is an interval order. Similarly, the $>$ relation in relation sets [4, 26] states that if $a > b$ but not

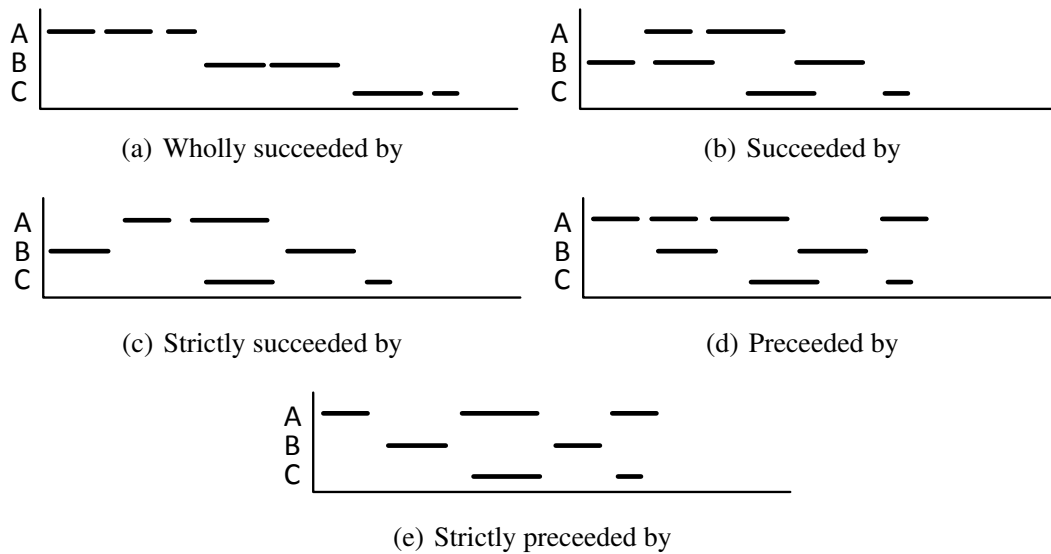


Fig. 4. Different causal relations based on the intervals

$b > a$, then a and b are causally ordered, i.e., a is followed by b , but b never followed by a . In terms of intervals, similar relations can be defined. Again, as an activity possibly has multiple intervals, we need to adapt the notion of causality to sets of intervals.

The first causality relation we introduce is if all intervals of some activity t occur after the intervals of s occurred, i.e., s is wholly succeeded by t . An example is depicted in Fig. 4(a), in which A is wholly succeeded by B and B is wholly succeeded by C . If for each interval of s some interval of t can be found that wholly succeeds the interval of s , we say that s is succeeded by t . In Fig. 4(b), A is always succeeded by B , and B is always succeeded by C . Note that this allows intervals of t to occur simultaneously with intervals of s , or even occurring before s , as shown in Fig. 4(b) where B occurs before A . If s is succeeded by t and they have no overlapping intervals, we say that s is *strictly* succeeded by t . An example is shown in Fig. 4(c), where A is strictly succeeded by B , and B strictly succeeded by C . Note that whereas the succeeded relation is transitive, the strictly succeeded is not, as A and C have overlap.

Symmetrically, if for each interval of t an interval of s can be found that wholly precedes the interval of t , we say that t is preceded by s . This allows intervals of s to occur after intervals of t , or even simultaneously, as shown in Fig. 4(d) where B is preceded by A , and C by B . The relation is called *strict*, if s and t are not simultaneously. Again, as shown in Fig. 4(e), the strictly preceded relation is not transitive, as B is strictly preceded by A , and C by B , but A and C have overlap. This leads to the following notions of causality.

Definition 11 (Causality). Let f be a generalized interval function over some set S . Let $s, t \in S$. Then:

- s is wholly succeeded by t , denoted by $s \blacktriangleright t$, if all intervals of t are after the intervals of s , i.e., $\pi_2(f \uparrow(s)) < \pi_1(f \uparrow(t))$;

- s is succeeded by t , denoted by $s \supseteq t$, if each interval of s is followed by an interval of t , i.e., $\forall(a, b) \in f(s) : \exists(c, d) \in f(t) : b < c$;
- s is strictly succeeded by t , denoted by $s \triangleright t$, if $s \supseteq t$ and not $s \leftrightarrow t$;
- t is preceded by s , denoted by $s \sqsupseteq t$, if each interval of t is preceded by an interval of s , i.e. $\forall(c, d) \in f(t) : \exists(a, b) \in f(s) : b < c$;
- t is strictly preceded by s , denoted by $s \sqtriangleright t$, if $s \sqsupseteq t$ and not $s \leftrightarrow t$.

It is easy to see that the wholly succeeded relation is a strict order. Similarly, the followed by and preceded by relations are transitive. However, these relations are not irreflexive in general. Only if the set of intervals for some activity is finite, the relations are irreflexive as well, and thus a strict order. If an activity has an infinite set of intervals, then it is succeeded by itself.

If some activity is wholly succeeded by some other activity, then it is easy to show that the former activity is strictly succeeded by the latter, and the latter is strictly preceded by the former.

Corollary 12. *Let f be a generalized interval function over some set S . Then (1) \triangleright is a strict order, (2) \supseteq , and \sqsupseteq are transitive, and (3) $x \triangleright y \implies x \triangleright y \wedge x \sqtriangleright y$ for all $x, y \in S$.*

Further, the strictly succeeded by and strictly preceded by relations are subsets of the succeeded by and preceded by relations, respectively.

Corollary 13. *Let f be a generalized interval function over some set S . Then (1) $\triangleright \subseteq \supseteq$, and (2) $\sqtriangleright \subseteq \sqsupseteq$.*

As for the interval order $>$ defined on events, the wholly succeeded by relation on activities is an interval order, which follows directly from the definitions.

Lemma 14 (Wholly succeeded is an interval order). *Let f be a generalized interval function over some set S . Then \triangleright is an interval order.*

Proof. Let $a, b, c, d \in S$ such that $a \triangleright b$, and $c \triangleright d$. We need to show that either $a \triangleright d$ or $c \triangleright b$ holds.

Suppose $a \triangleright d$ does not hold, i.e., $\pi_2(f \uparrow(a)) \geq \pi_1(f \uparrow(d))$. Then $\pi_2(f \uparrow(c)) < \pi_1(f \uparrow(d)) \leq \pi_2(f \uparrow(a)) < \pi_1(f \uparrow(b))$. Hence, $c \triangleright b$.

Similarly, suppose $c \triangleright b$ does not hold, i.e., $\pi_2(f \uparrow(c)) \geq \pi_1(f \uparrow(b))$. Then $\pi_2(f \uparrow(a)) < \pi_1(f \uparrow(b)) \leq \pi_2(f \uparrow(c)) < \pi_1(f \uparrow(d))$. Hence, $a \triangleright d$. \square

5.3 Other Control-Flow Relations

The simultaneousness and causality relations form the basic building blocks of any process modelling language. Many other control-flow relations can be defined, depending on the needs within the process modelling notation. For example, one can define a *next-to* relation on activities, defining whether two activities are directly after one another, without any activity in between. As for simultaneousness, this can be a weak relation, i.e., for two activities there are intervals next to each other, or a strong relation, i.e., for all intervals.

Definition 15 (Next-to relation). Let f be a generalized interval function over some set S . Let $s, t \in S$. We say s is next to t , denoted by $s \circ t$, if some interval of s is directly followed by an interval of t , without any occurrence of other activities in between, i.e., $\exists(k, l) \in f(s), (o, p) \in f(t) : (l < o \wedge \neg(\exists u \in S : (m, n) \in f(u) : l < n \wedge m < o))$

Similarly, s is followed by t , denoted by $s \bullet t$, if all intervals of s are directly followed by an interval of t , without any occurrence of other activities in between, i.e., $\forall(k, l) \in f(s) : (\exists(o, p) \in f(t) : l < o \wedge \neg(\exists u \in S : (m, n) \in f(u) : l < n \wedge m < o))$

Naturally, if s is followed by t , then s is also succeeded by t .

Corollary 16 (Follows implies succeeded). Let f be a generalized interval function over some set S , and let $s, t \in S$. Then if $s \bullet t$ then also $s \succeq t$.

As activities are represented by sets of intervals, an activity can be enclosed by some other activity, i.e., some activity B always occurs between two intervals of A . We call this relation *betweenness*. Again, this can be a strong notion, requiring this for all intervals of B , or a weak notion, only requiring the existence of such an interval of B .

Definition 17 (Betweenness). Let f be a generalized interval function over some set S . Let $s, t \in S$. We say t is weakly in between s , denoted by $s \leftrightarrow t$, if some interval of t is in between two intervals of s , i.e., $\exists(m, n) \in f(t), (k, l), (o, p) \in f(s) : l < m \wedge n < o$.

Similarly, we say t is in between s , denoted by $s \cup t$, if all intervals of t are between two intervals of s , i.e., $\forall(m, n) \in f(t) : (\exists(k, l), (o, p) \in f(s) : l < m \wedge n < o)$.

Although betweenness seems a natural choice, it can be expressed in terms of the basic causality notions defined in Def. 11.

Corollary 18 (Betweenness implies basic causality). Let f be a generalized interval function over some set S , and let $s, t \in S$. If $s \cup t$ then $s \supseteq t$ and $t \supseteq s$.

6 Discovering Declarative Models

The density of the time scale has a great impact on the level of concurrency in an event log, and hence in the model that describes the allowed behaviour of the executions in the event log. Procedural languages prescribe the order in which activities are supposed to occur. Consequently, concurrency needs to be modelled explicitly in such languages. Instead, we use a declarative approach that has concurrency as a language primitive: activities may occur simultaneous, unless constraints prohibit the execution of the activity.

6.1 Declare Language

In this paper, we use the declarative language Declare [2]. The language provides a graphical layout to visualize the activities and constraints in the model. It does not come with a predefined set of language constructs. Instead it offers a set of language constructs called *constraint templates*, which the user may adapt to its own needs. These constraint templates are based on Linear Time Logic (LTL) [8]. Declare comes with a

Table 3. Basic language constructs in Declare



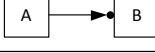
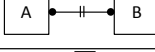

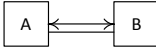
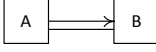
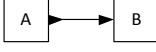
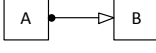
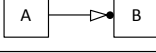
Constraint	Template	Graphically
init	$\sigma(1) = A$	
response	$\Box(A \implies \Diamond B)$	
precedence	$((\neg B) U A) \vee \Box \neg B$	
non coexistence	$\neg((\Diamond A) \wedge (\Diamond B))$	
$(n..m)$ occurrences	$ \{i \mid \sigma(i) = A\} \in [n..m] \subseteq \mathbb{N}$	

Table 4. Newly introduced constraints in Declare

Constraint	Template	Graphically
strongly simultaneous	$A \rightleftarrows B$	
Dependently simultaneous	$A \rightrightarrows B$	
wholly succeeded	$A \blacktriangleright B$	
strict response	$A \triangleright B$	
strict precedence	$A \sqsupset B$	

basic set of language constructs. Tbl. 3 depicts the language constructs from Declare used in this paper.

The first constraint template, *init*, states that the first activity of any sequence, represented by σ , should start with A , where A is a placeholder for the actual activity. Similarly, the *response* template states that every A should eventually be followed by some activity B . The *precedence* constraint template expresses that some activity B has to be preceded by some activity A . With the *non coexistence* template, it is possible to express that two activities should not occur together in any sequence. Last, we allow to limit the number of times an activity can be executed using the *n..m occurrences* template, where $n \leq m$ specifies the minimal and maximal number of times some activity A is executed.

6.2 Interval-Based Constraints

The constraints in Declare do not take activity duration into account. Consequently, we need to relate the constructs used in Declare with the simultaneousness and causality relations defined in the previous section.

First, consider the *response* constraint template. This constraint expresses that activity A is always eventually followed by B . This can be interpreted in many different

ways, e.g., “once activity *A* is *started*, activity *B* will eventually start”, or “once activity *A* is *finished*, activity *B* will eventually start”. We choose the latter interpretation, i.e., after activity *A* finished, eventually activity *B* will start. A second consideration is whether the response and precedence templates should allow the activities in the constraint to occur simultaneously. As the response template is transitive in the Declare language, we allow the activities to overlap. Thus, we interpret the response template with the succeeded by relation introduced in the previous section. Similarly, we interpret the precedes template as “before activity *B* starts, activity *A* should be finished”, which coincides with the precedes by relation introduced in the previous section. Therefore, the strictly succeeded by and strictly preceded by relations are added to the Declare language, as shown in Tbl. 4.

Although in Declare concurrency is a language primitive, each activity in the model is considered to be instantaneous. The language does not provide any constraint that limits concurrency without destroying it. Thus, the weak simultaneousness relation as presented in the previous section is directly supported in the language. The two stronger simultaneousness relations impose an order on the activities: although the activities may overlap, the other activity must be executed simultaneously. This is expressed by the strongly simultaneous template and dependent simultaneous template as depicted in Tbl. 4.

6.3 Discovery

In the previous section, we introduced several notions of simultaneousness and causality. Up to now, these relations only consider a single execution of the system. An event log contains a set of executions that are executed by, most likely, the same process. Hence, to come to a model that describes each of the executions in the event logs, we need to aggregate the relations over the different executions in the event log.

In what follows, we sketch a declarative discovery algorithm based on time intervals. Here, it is important to mention that the choice of the generalized interval functions for the activities breaks or makes the approach presented in this paper.

Events to interval First step in the approach is to map each event to an interval. In many cases the default event interval mapping, i.e., that maps each event to a single-point interval, can be used. In some cases, for example if event logs of multiple systems are combined, a reliability interval can be attached to each interval.

Activities to sets of intervals Next step is the construction or discovery of an accurate activity interval mapping. For example, one can fix the granularity of the time scale, make it relative or absolute, and then map each event to the corresponding time interval. Or, one can use the event types to determine the life cycle of an activity, and base the activity interval mapping on this information. Although at first sight this seems to be a trivial step, there are many pitfalls [12]. For example, if two instances of the same activity run simultaneously, which interval should be used to map the activity on?

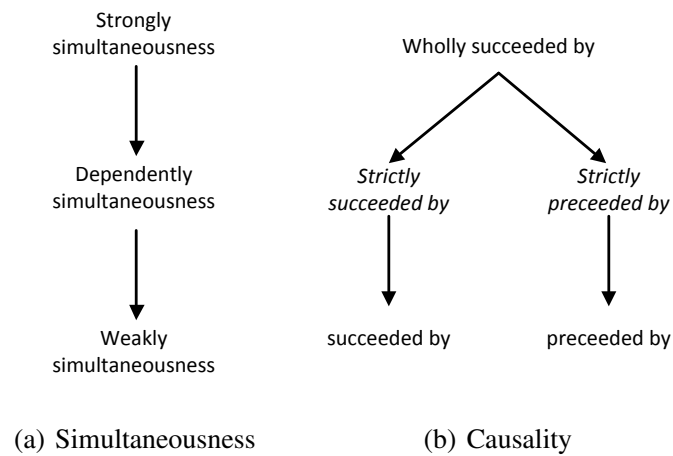


Fig. 5. Hierarchy of simultaneousness and causality relations

Derive relations Once the activity interval mapping has been established, we can start to derive the different relations. For example, the $(n..m)$ occurrences template can be easily constructed by analyzing the number of occurrences in each of the sequences in the event log. Similarly, the non-coexistence relation can be calculated by a single walk through the sequences of the event log.

Next step would be to derive the different simultaneousness relations and causality relations. For this, we use the relation hierarchy as depicted in Fig. 5, which follows directly from Cor. 10 and Cor. 12. An arrow from one relation to another relation means that the former is included in the latter. For example, the strongly simultaneousness relation is included in the dependently simultaneousness relation. The algorithm starts with assuming the strongest relation between each of the activities. By going through the different intervals, relations are weakened, until all intervals of all sequences in the event log have been inspected.

The algorithms to derive the simultaneousness and causality relations do not take transitivity into account, which results in models with many constraints, expressing the complete transitive closure of the respective relations. Therefore, these relations need to be reduced, such that the transitive closure of this reduced relations remain the same. For this, standard algorithms as described in [5] can be used. Although at first sight this seems a straightforward task, it is not, as one wants to take the hierarchy of relations into account during the reduction, which is closely related to the “minimum equivalent graph” problem [18], which is NP-hard.

Last, we sugar the models using nesting, as has been done in e.g. Dynamic Condition Response Graphs [13]. In this approach, set of nodes having the same constraints are nested in a so called “super nodes”.

For the example event log of Tbl. 1, the discovery algorithm that has been sketched above results in the model depicted in Fig. 6. For the activities B , C , and E , we briefly illustrate the steps of the algorithm. In the first step, we take a relative time scale for the activity interval mapping. Moreover, a “start” event denotes the start time of an activity and a “complete” event denotes the end time of an activity. Secondly, all three

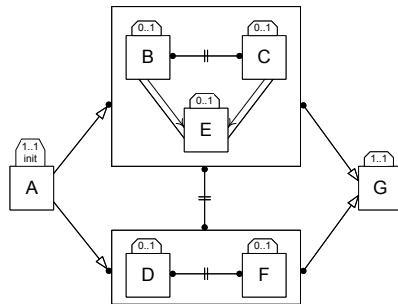


Fig. 6. Model discovered from the event log in Tbl. 1

activities occur at most once in each of the sequences of the event log resulting into a 0..1 occurrence relation for each of them. Also, activity *A* and *B* never occur together in each sequence. In the third step, it is discovered that activities *B* and *C* are strongly simultaneous. Also, the tree activities are all preceded by activity *A* and succeeded by activity *G*. Finally, in the last step, activities *B*, *E* and *C* are nested, as these activities are all preceded by *A*, do not coexist with *D* and *F*, and are all succeeded by *G*.

7 Conclusions

Timestamps in an event log play an essential role in process mining to determine the order in which events occur. A typical problem in process mining is the impreciseness of these timestamps. In this paper, we overcome this problem by assuming that each event occurs in some time window, i.e., in some interval. As the intervals in an event log are on the level of events, rather than on the level of activities, we have presented an approach based on sets of intervals to represent the occurrences of the activities in the model. On these sets of intervals new notions of simultaneousness and causalities are derived. These notions form the basis to discover declarative models.

The simultaneousness relation forms a natural candidate for the dependency relation in Mazurkiewicz traces. In this way, simultaneousness can be used to test the completeness of event logs, by exploring the Mazurkiewicz equivalent traces.

Although intervals are a natural choice to overcome the impreciseness of timestamps, choosing the right time window is a hard problem. The events and activities can be mapped to intervals in many different ways. The granularity of the time scale, like milliseconds, hours or days, can be used to define the intervals, the time scale can be relative or absolute. Or, if the event log contains a transition life cycle, like a start and complete event, then the first and last event of each execution can be used to determine the intervals in the activity interval mapping. Empirical research is needed to test, validate and compare the different alternatives.

As the proof of the pudding is in the eating, we will implement the presented approach in ProM [23] to perform more case studies to test and fine tune the resulting declarative models.

Acknowledgements The authors would like to thank Jetty Klein for the fruitful discussions about paradigms of concurrency and interval orders.

References

1. W.M.P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer-Verlag, Berlin, 2011.
2. W.M.P. van der Aalst, M. Pesic, and M.H. Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - Research and Development*, 23:99–113, 2009.
3. W.M.P. van der Aalst and C. Stahl. *Modeling Business Processes - ũ A Petri Net-Oriented Approach*. The MIT Press, 2011.
4. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *Knowledge & Data Engineering*, 16(9):1128–1142, 2004.
5. A.V. Aho, M. R. Garey, and J. D. Ullman. The Transitive Reduction of a Directed Graph. *SIAM Journal on Computing*, 1(2):131–137, June 1972.
6. J.F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154, July 1984.
7. J.F. Allen. Actions and Events in Interval Temporal Logic 1 Introduction. *Journal of Logic and Computation*, 4:531–579, 1994.
8. E. Clarke and E. Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Logics of Programs*, volume 131 of *LNCS*, pages 52–71. Springer-Verlag, Berlin, 1982.
9. P. Degano and U. Montanari. Concurrent Histories: A Basis for Observing Distributed Systems. *Journal of Computer and System Sciences*, 34(2-3):422–461, 1987.
10. V. Diekert and G. Rozenberg, editors. *The Book of Traces*. World Scientific, Singapore, 1995.
11. P.C Fishburn. Interval graphs and interval orders. *Discrete Mathematics*, 55(2):135–149, July 1985.
12. T. Gschwandtner, J. Gärtner, W. Aigner, and S. Miksch. A Taxonomy of Dirty Time-Oriented Data. In G. Quirchmayr, J. Basl, I. You, and E. Weippl, editors, *CD-ARES 2012*, volume 7465 of *LNCS*, pages 58–72. Springer, 2012.
13. T. Hildebrandt, R. Mukkamala, and T. Slaats. Nested dynamic condition response graphs. *Fundamentals of Software Engineering*, pages 343–350, 2012.
14. R. Janicki and M. Koutny. Structure of concurrency. *Theoretical Computer Science*, 112(1):5–52, April 1993.
15. Wen. L., J. Wang, W.M.P. van der Aalst, B. Huang, and J. Sun. A Novel Approach for Process Mining based on Event Types. *Journal of Intelligent Information Systems*, 32:163–190, 2009.
16. R.D. Luce. Semiorders and a Theory of Utility Discrimination. *Econometrica*, 24(2):178–191, 1956.
17. F.M. Maggi, R.P.J.C. Bose, and W.M.P. van der Aalst. Efficient discovery of understandable declarative process models from event logs. In *CAiSE*, volume 7328 of *LNCS*, pages 270–285. Springer-Verlag, Berlin, 2012.
18. D.M. Moyses and G.L. Thompson. An algorithm for finding the minimum equivalent graph of a digraph. *Journal of the ACM*, pages 455 – 460, 1969.
19. J. Nakatumba and W.M.P. van der Aalst. Analyzing Resource Behavior Using Process Mining. In S. et al. Rinderle-Ma, editor, *BPM 2009 Workshops*, volume 43 of *LNBIP*, pages 69–80. Springer, 2009.

20. S.S Pinter and M. Golani. Discovering Workflow Models from Activities' Lifespans. *Computers in Industry*, 53:283–296, 2004.
21. Y.-L. Qu and T.-S. Zhao. Building Process Models Based on Interval Logs. In M. Ma, editor, *Communication Systems and Information Technology*, volume 100 of *LNEE*, pages 71–78. Springer, 2011.
22. G. Rosu and S. Bensalem. Allen Linear (Interval) Temporal Logic – Translation to LTL and Monitor. In *Computer Aided Verification*, pages 263–277. Springer, 2006.
23. H.M.W. Verbeek, J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. XES, XE-Same, and ProM 6. In *Information System Evolution*, volume 72, pages 60–75. Springer, 2011.
24. J. de Weerd, M. de Backer, J. Vanthienen, and B. Baesens. A Multi-dimensional Quality Assessment of State-of-the-Art Process Discovery Algorithms using Real-Life Event Logs. *Information Systems*, 37:654–676, 2012.
25. M. Weidlich, J. Mendling, and M. Weske. Efficient consistency measurement based on behavioral profiles of process models. *IEEE Trans. Software Eng.*, 37(3):410 – 429, 2011.
26. M. Weidlich and J.M.E.M. van der Werf. On Profiles and Footprints – Relational Semantics for Petri Nets. In *Application and Theory of Petri Nets*, LNCS, pages 148–167. Springer, 2012.
27. J.M.E.M. van der Werf, B.F. van Dongen, C.A.J. Hurkens, and A. Serebrenik. Process Discovery Using Integer Linear Programming. *Fundamenta Informatica*, 94(3 – 4):387 – 412, 2009.
28. N. Wiener. A contribution to the theory of relative position. *Proceedings of the Cambridge Philosophical Society*, 17:441–449, 1914.
29. N. Wiener. A new theory of measurement: A study in the logic of mathematics. *Proceedings of the London Mathematical Soc.*, s2-19(1):181–205, 1921.

ModBE'13: Short Presentation

Improving Emergency Department Processes Using Coloured Petri Nets

Khodakaram Salimifard¹, Seyed Yaghoub Hosseini², Mohammad Sadegh Moradi¹

¹ Industrial Management Department, Persian Gulf University, Bushehr, Iran

salimifard@pgu.ac.ir, msadeghmoradi@gmail.com

² Business Management Department, Persian Gulf University, Bushehr, Iran

hosseini@pgu.ac.ir

Abstract. With increasing demand for medical services, emergency departments (ED) are facing problems such as overcrowding and dissatisfaction. Improving the key performance indicators of EDs has been the focal point of healthcare management. This paper addresses performance analysis of ED of a general hospital. To this aim, a discrete event dynamic modeling approach is used to model the ED processes. The model employs a hierarchical timed Coloured Petri net framework in a concise and detailed way to capture patient flow and care processes within the ED. The simulation model was validated against historical data and then different types of scenarios were used to assess, compare and improve ED key performance indicators, such as patients waiting time, length of stay (LOS), and resource utilization rate. The proposed model helped the hospital policy makers to configure the ED in a way to improve its efficiency and staff satisfaction.

Keywords: Healthcare System, Emergency Department, Coloured Petri Net, Performance Analysis

1 Introduction

Emergency departments (ED) are facing different problems which affect their performance. Of these, overcrowding is a common issue around the world which provides EDs patient with long length of stay (LOS), waiting times for receiving services and then dissatisfaction [1]. Although most emergency departments are under growing demand, they often face with insufficient staffing and budget constraints. One solution to this problem is to increase capacity of ED, providing adequate facilities and manpower, but this is not the best approach for solving the problem, and perhaps not achievable [2]. Recently, the need for improvement in ED processes due to cost, overcrowding and safety of patients admitted to a large extent [3]. To improve the efficiency and quality of ED processes, different methods were used which include process mapping, demand management, critical path identification, queuing systems, statistical forecasting, balanced scorecard and computer simulation [4].

In the last years, the use of computer simulation to help effective decision making in health care and to improve the medical operations has been rising [5]. One of the main rea-

sons that simulation has become a common practice in solving medical problems is its ability to dynamically analyze situations and present to the stakeholders a more realistic view of the system [6]. The main purpose of the use of simulation studies in health care is to reduce waiting times and length of stay for patients, better use of resources and reducing operating cost [7]. Among the various methods for simulation in health care, discrete event simulation is the most used method especially in EDs, and it seems to be a better alternative with less time and cost compared to more traditional statistical methods [8].

This study is intended to present a general simulation model for studying hospital emergency department. For this purpose, we used Coloured Petri Nets modeling and simulation formalism for making a general model of emergency department of a general hospital. In addition to internal processes, external relations between ED and other hospital wards, such as Radiology and Laboratory, is also considered. The main objective of this paper is, hence, to improve ED processes. The problem we are dealing with is ED overcrowding which provide patient with long length of stay and waiting times.

The remainder of the paper is organized as follows. Section 2 covers a brief literature review of the application of simulation in emergency departments. Research methodology, simulation model, input data and variables are presented in Section 3. Section 4 focuses on improvement scenarios and results of simulation runs. Finally, the paper is concluded in Section 5.

2 Literature review

In the literature, the main focal point of discrete-event simulation models that are used for analysis of hospital emergency department is improving the flow of patients to reduce waiting times. Examples of this include studying patient flow and forecasting ED overcrowding using simulation by Hoot et al. [9], defining buffer concept to reduce waiting times and increase throughput, and comparing amount of improvement gain by buffers by Kolb et al. [10], and Khandekar et al. [11] paper on rearranging sequence of activities of care process in order to reduce waiting times. Another area of ED simulations study focus is on capacity estimation which determines the optimum number of personnel and physical resources such as bed [12], [13] and also ED layout [14]. Improving quality of services and ED processes is another area of study [5], [15].

Although the use of Petri nets in the health sector is less than other fields such as computer networks and production system, but it can be a useful method in this area. Here some related works in this area are presented. Xiong et al. [16], apply petri nets for modeling and analysis of health care process. They used a Petri net model to examine the effect of changes in arrival pattern and resources on performance metrics such as waiting times and resource utilization. Chockalingam et al [17] used Petri nets to model patient and resource flow in a hospital system. Using the Petri net model they obtained a stochastic representation of a metric termed distance to divert which measure the proximity of a hospital to a divert state. Dotoli et al. [18] focused on pulmonology department workflow and drug distribution system and used simulation as a decision support system. They employed a timed Petri net (TPN) framework to describe the workflow in the department. Another example is the work done by Ronny Man et al [19] of using process mining and Petri nets for pre hospital stroke care. In the paper, process mining is used to extract process related information e.g. timing information. Jorgensen et al [20] have used CPN for implementing a new Electronic Patient Record Workflow System at two stages. The first CPN model is

used as an execution engine for a graphical animation called EUC and the second CPN model is a Coloured Workflow Net (CWN). Together, the EUC and the CWN are used to close the gap between the given requirements specification and the realization of these requirements with the help of an IT system.

In this paper, care processes of ED are modeled using hierarchical timed CPN. The main focus of the model is on patient flows. The model also concentrates on the inter-department care processes. It is aimed to find a suitable operating scenario to improve some performance metrics of the department. Similar to [17], this paper has considered the relationship between different departments (Labs, Radiology) of the hospital. Performance metrics including waiting time, patient length of stay, and resource utilization are calculated under different operating scenarios. Compared to existing literature, this paper puts more emphasis on using features of CPN (color, time, and hierarchy) to capture the complex nature of the system.

3 Research methodology

In this paper, CPN Tools is utilized to create the Coloured Petri net model of the system and to simulate the model to produce desired outputs. CPN Tools [21] is a powerful software tool for modeling and simulation of discrete event systems modeled in CPN. Our choice of using CPNs to model ED patient flow stems from the fact that PNs capture structural properties of the underlying system which we can study and use. Petri nets provide the foundation of the graphical notation and the basic primitives for modeling concurrency and synchronization, conditions which are common in our model. After reviewing a wide range of related literature, an initial model was prepared. Based on the initial model, the generic conceptual model was developed. The generic model aimed to capture the characteristics of an emergency department of a general hospital in Iran. Information required for the modeling and simulation of processes were collected using hospital information system, sampling in ward, and also open interview with employees. In order to simulate the model under different configurations, different types of improvement scenarios were defined and compared against performance criteria. Please note that here the term “Generic” as Gunal and Pidd [22] mentioned means that the model has a defined structure with probability distributions that can be parameterized by the user.

The hospital under study is a general hospital in the city of Yazd of Iran. The emergency department of the hospital consists of one triage room, one primary visit room, admission and discharge unit, CPR room, and two inpatient areas with 24 inpatient bed. It works 3 shifts a day with 1 triage nurse, 1 general practitioner (GP), 1 emergency medicine specialist (SP), 1 admission staff and 6 nurses.

3.1 Process flow chart

Fig.1. depicts an overall patient flow of Emergency Department of the hospital. This flow chart is depicted based on researcher observation of the process and also domain expert opinion. The process diagram was drawn in such a way that in addition to our hospital it could also be used in other Iranian hospitals. Patients are triaged on arrival at the emergency department. The triage nurse makes an initial evaluation of patient symptoms. Then, according to Emergency Severity Index (ESI), she classifies the patient in one of the 5 levels of emergency. A patient in level 1 is with more acuity while a patient in level 5 is in fact

an outpatient with less acuity. After this stage, patient is referred to the GP. The GP determines whether or not the patient requires other care services. Usually, patients with acuity level 5 will leave the ED as soon as the payment cleared. Other patients who need more medical services, such as diagnostic tests, need to be registered and will be directed through the other processes. The final decision about the patient including discharge, inpatient at ED, or being referred to other wards is taken by SP.

3.2 Performance variables

For each process improvement project, establishing quantitative measures to implement changes and develop monitoring system for continuous improvement is crucial. In this paper, we investigate three key performance metrics including patients waiting times, length of stay, and ED resource utilization.

- Waiting times [min]. It is the mean duration a patient need to spend in the ED waiting room.
- Length of stay (LOS) [min]. The total time of staying at ED, from arrival to the time of final decision made by SP.
- Resource utilization [%]. Represents the total busy time of resources compared with total working time.

3.3 Data collection

Model inputs are distribution functions of ED activities. Random sampling was used to estimate required data for patient's arrival times and service time for all resources. All distributions determined from the data and used in the model were validated by using Kolmogorov Smirnov goodness of fit test with a 5% significance level. Using statistical goodness of fit method, the distribution of processing time of different activities have been defined.

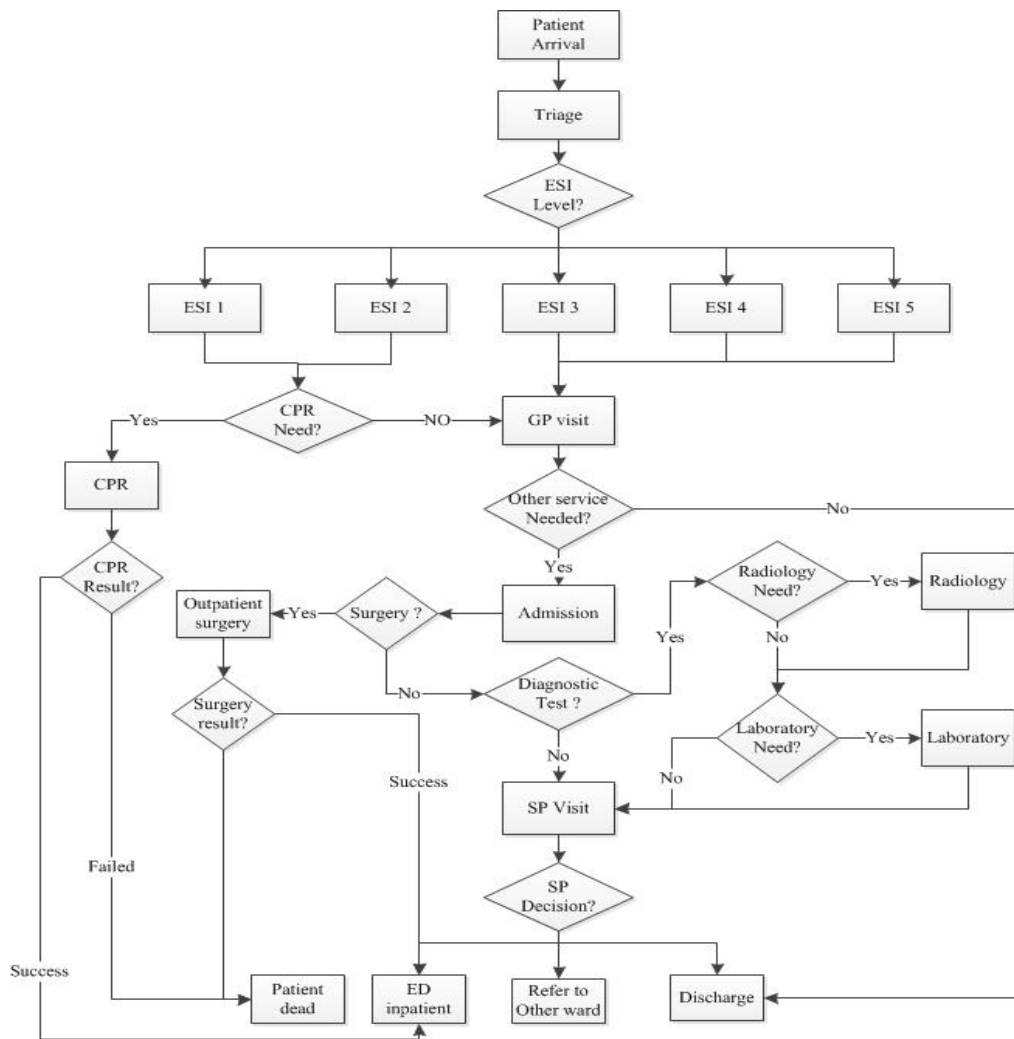


Fig. 1. Process flow of emergency department

Table 1. Simulation input distribution functions

Input parameter	Distribution
Patients inter arrival pattern	Exponential , expel(9)
Triage time	Lognormal , 0.21 + LOGN(0.875, 0.613)
GP visit time	Gamma , 1 + GAMM(0.732, 2.2)
admission	Lognormal , 0.16 + LOGN(1.11,0.729)
SP visit time	Triangular (1,2,3)
CPR time	Triangular (5,15,30)
ED outpatient surgery(OR)	Triangular (10,20,30)

In cases where there was no possibility of sampling, based on information available in the hospital information system and also hospital staff experience, minimum, average and maximum duration of each activity were chosen as the statistical distribution.

3.4 The Hierarchical Timed Petri Net Model of the ED

We used Color Petri-nets (CPNs) to model patient flow in an emergency department of a hospital. A PN consists of a set of places, and a set of transitions and arcs that connect place(s) to a transition and vice-versa. Non-negative integers assigned to every place in the net are known as tokens.

Overview of Colored Petri Nets. Coloured Petri nets are a discrete-event modeling language combining the capabilities of Petri nets with the capabilities of a high-level programming language. Petri nets are directed, bipartite graphs that can be used to model discrete distributed systems. A CPN as defined by [23] is a nine-tuple $CPN = (P, T, A, \Sigma, V, C, G, E, I)$, where P is a finite set of place T , is a finite set of transitions T such that $P \cap T = \emptyset$, $A \subseteq P \times T \cup T \times P$ is a set of directed arcs, Σ is a finite set of non-empty color sets, V is a finite set of typed variables such that type $[v] \in \Sigma$ for all variable $v \in V$, $C: P \rightarrow \Sigma$ is a color set function that assigns a color set to each place, $G: T \rightarrow \text{EXPR}_v$ is a guard function that assigns a guard to each transition t such that type $[G(t)] = \text{Bool}$, $E: A \rightarrow \text{EXPR}_v$ is an arc expression function that assigns an arc expression to each arc α such that type, $[E(\alpha)] = c(p)_{MS}$ where p is the place connected to the arc α , $I: P \rightarrow \text{EXPR}_0$ is an initialization function that assigns an initialization expression to each place p such that type. $[I(p)] = c(p)_{MS}$. A CP-net has a distinguished initial marking, denoted by M_0 , and obtained by evaluating the initialization expressions. The marking can be viewed as a ‘snapshot’ of how tokens are distributed in the PN [24].

The simulation model of ED. Fig.2. shows the key structures of the model. The top layer of the ED model is illustrated in this figure. This layer is the core part of the model. In the model each place (circle) represents the state where patients may to be exposed there (table 2). Entry of each patient to the ED is modeled by a token on the place `New Patient` (Fig.2). This place has the color set `PAT`, whose elements are 5-tuples (`ESI`, `at`, `qtr.`, `wt`, `pt`) consisting of patient Emergency Severity Index (`ESI=1, . . . , 5`), patient arrival time to the ED (`at`), an intermediate variable for Calculating wait time (`qtr.`), patient wait time for receiving services (`wt`) and activities process time (`pt`). In the initial marking, the `New Patient` has a random integer ESI number Between 1 to 5, an arrival time based on an exponential distribution with mean 9, `qtr.` is equal to `at` and `wt` and `pt` are equal to 0. In the ED layer (Fig.2) there are 8 transitions (the rectangles) with tag beside them which called substitution transitions. Each of this transition has a subnet page belong to it that corresponds to one of the considered tasks in the process. To know about the model mechanism in each subnet consider GP visit subnet page as an example (Fig.3). The occurrence of the transition `start visit` models the situation where a general Practitioner (resource) changes from being ready to being busy until the transition `End visit` occurs. Patients wait to seize GP and after stochastic delay (GP visit time) and receiving GP orders, they release that resource and come back to top layer to carry on rest of the process. The other page is as the illustrated mechanism.

Table 2. Place Description of the ED core layer

Place	description
P1	State for non urgent patient
P2	State for urgent patient (CPR needed)
P3	Patients visited and need extra services
P4	Patients visited and will be discharge
P5	Patients admitted
P6	Patients admitted and need surgery
P7	Number of patients waiting for radiology
P8	Number of patients waiting for laboratory
P9	Patients have done radiology test
P10	Patients have done radiology and also need lab test
P11	Surgery result with success
P12& P13	Patients with their Lab test result
P14	Patients have done radiology and do not need Lab

The model contains resources (shown in Table 3) like nurses; physicians etc. in the form of tokens, and patients use these resources according to model logic to receive care. These resource tokens are held by the patients until they move to the next stage. The delay in availability of a resource is represented as non-availability of tokens to advance the patient tokens through the net. This delay increases the number of patient tokens in the system waiting for a resource. Also Sets of variables used on the transitions in Fig. 2 are showed in Table 4.

Table 3. Initial resource-token distribution

resources places	tokens
Triage nurse	1
GP Doc	1
admission	1
SP Doctor	1
Radiology staff	6
Laboratory staff	12
CPR equipment	1

Table 4. Model variables and description

variable	description
p	Represent each patient and carry five attribute: <i>ESI</i> , arrival time, <i>qt</i> , wait time, process time which are assigned to them
l	determine that patient need laboratory test or not
r	determine that patient need radiology test or not
LR	Laboratory result
RR	Radiology result
gp	General Practitioner (resource)
sp	Emergency medicine specialist (resource)

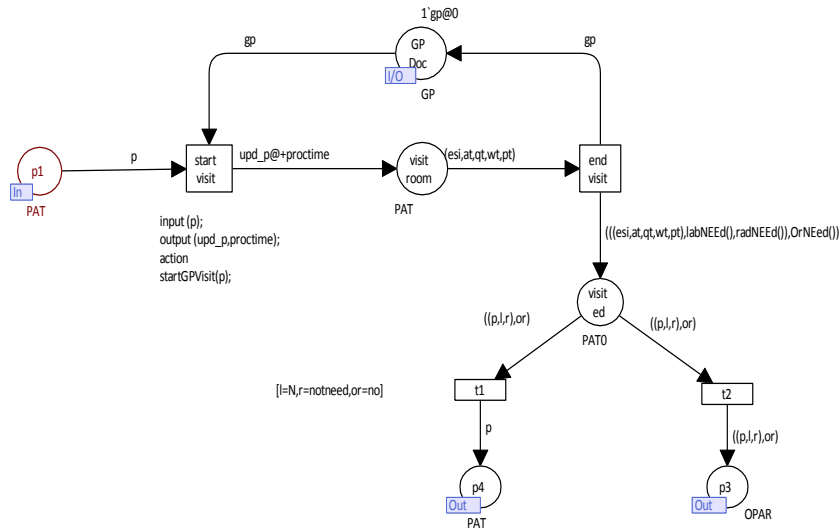


Fig. 3. GP visit page of substitution transition GP visit in ED page (subnet layer)

Finally we also used CPN Tools state space graph to investigate whether the model works truly or not. The state space tools are used to calculate state spaces and to generate state space reports. Because our graph is very large, it is not possible to show it here.

Table 5. Comparison between simulated and real data.

Performance criteria	Simulation output	real data
Wait for GP visit	3.2	2.9
Wait for Admission	5.12	5.5
Wait for SP visit	1.15	1
ESI 1&2 LOS	38.2	37.5
ESI 3 LOS	185	187
ESI 4 LOS	140	136
ESI 5 LOS	11.5	12

The performance metrics are investigated using 5 different replications with 95% confidence interval. In each case the system is simulated by a long simulation run of 3 years.

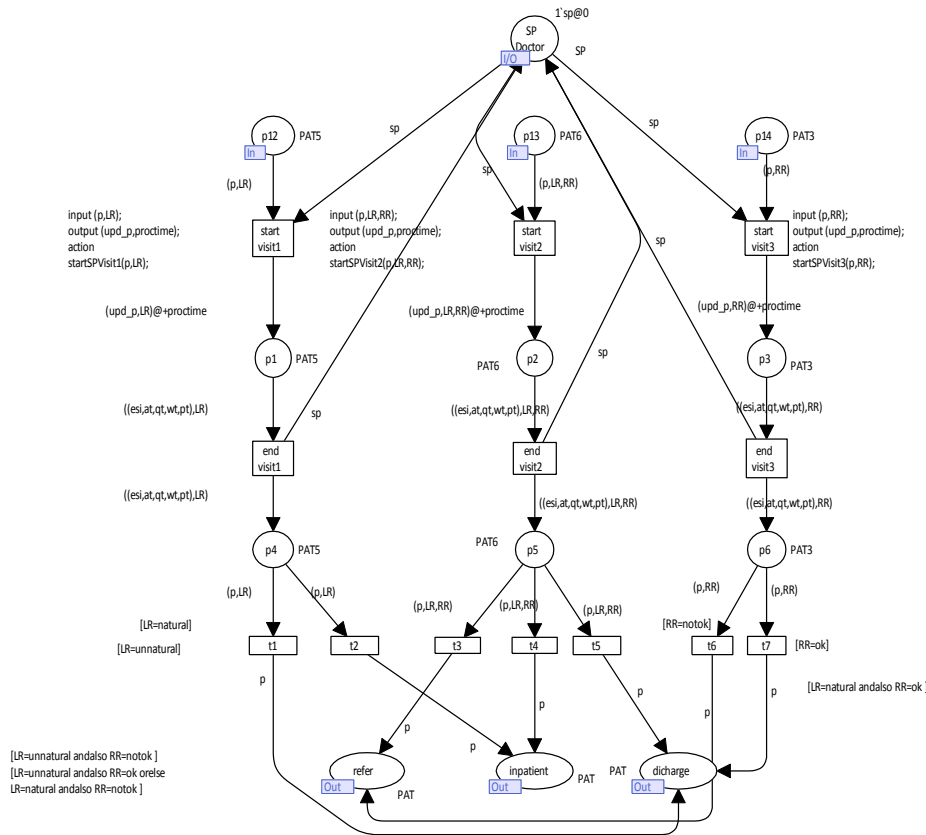


Fig. 4. SP visit page of substitution transition SP visit in ED page (subnet layer)

4 Improvement scenarios

In order to improve processes in terms of system performance metrics, four types of scenarios were defined. These alternate scenarios are validated with domain experts and then were implemented in simulation model. They are as follow:

- Current scenario (benchmark):

A – Current state of the ED as a basis for comparisons.

- **Increase or decrease scenarios.** It is, in fact, the most common type of scenario associated with simulation studies. In this scenarios (increase or decrease) number of resources, the number of emergency room doctors, nurses, beds and other physical resources will be changed. In this view, one scenario is defined here:

B – Increase an emergency medicine specialist (SP)

- **Displacement Scenarios.** In these scenarios, if possible, an alternative resource will be replaced with available resource.

C – Putting in place an emergency medicine specialist instead of GP

- **Structural scenarios.** The purpose of these scenarios is change of the process activities and even delete or add new activities as part of the process.

D – Remove triage unit and refer patients for triage and visit to GP

- **Hybrid scenarios.** These scenarios are defined as a combination of two or more than two of the above scenarios. For example, displacement scenarios, and a structural scenario combined to make a hybrid one.

E – Replace GP visit and triage activities with a substitute emergency medicine specialist who does these two.

In scenario B, we have added 1 specialist to SP Doctor place and then run the simulation model. To implement scenarios D and E, we have to change our basic model. In these scenarios triage and visit is done simultaneously by a substitute doctor (GP or SP). The difference is on the time of visit done by each of them. It's less for SP than GP.

The results of running simulation model with scenarios A to E alongside with their improvement are shown in Table 6. We ran each of the improvement options individually as separate scenario for this purpose.

Table 6. Improvement rate of each scenario considering its resources

Scenarios	Resources			Improvement rate (%)									
	GP	SP	Admission Staff	Resource utilization rate (%)			Wait for GP	Wait for SP	Wait for Admission	ESI 1&2 LOS	ESI 3 LOS	ESI 4 LOS	ESI 5 LOS
				Admission Staff	SP	GP							
A	1	1	1	62	63.7	58	3.2	5.12	1.1	38	185	140	11.5
B	1	2	1	+9.67	-10.5	0	0	+23	0	+5.2	+2.7	+2.65	0
C	-	2	1	+7.25	-1.8	-	-	-21	-7.2	+5.2	+0.5	+1	+8.7
D	1	1	1	+8	-0.3	+14.5	-65	+0.3	-81	+8.6	+1	+1.42	+19.1
E	-	2	1	+11.3	-0.3	-	-	-2	-81	+9.2	+1.35	+2.15	+21.7

Benchmark scenario, A, represent current situation in terms of three performance measures. Waiting time is one of the effective measures of patient satisfaction. Here are three main areas of patients waiting for service. Current scenario has the lowest waiting for GP. Scenario B and C reduce SP waiting by about 45% and 0.4%. Scenario B reduces admission waiting by 10%. Patients' length of stay is a measure of ED efficiency and very important in hospital performance evaluation. We compare LOS for patients with different ESI level. ESI 1 and 2 include those patients who need CPR and then go to inpatient. In this level, E has 9.2% improvement. D reduces LOS by about 8.6% and B and C by about 5.2%. Patients with ESI 3 are patients who need two diagnostic tests here include Laboratory and

Radiology. In this level B, C, D and E reduced LOS by about 2.7%, 0.5%, 1% and 1.35%. Patients with ESI 4 just need one diagnostic test, laboratory or radiology. In this level B, C, D and E reduced LOS by about 2.65%, 1%, 1.42% and 2.15%. Finally, patients with ESI 5 are outpatient and leave ED after GP visit. In this level B, C, D and E reduced LOS by about 0%, 8.7%, 19.1% and 21.7%. Resource utilization represents total busy time of resources to available working time under the simulated conditions. It is a good measure for ED manager in the allocation of resources. Scenario A improves GP utilization by 18%. Scenario B, C, D and E improve admission staff utilization by about 9.67%, 7.25%, 8% and 11.3%. Scenario A has the most SP utilization and other scenarios reduced it. Substituted SP utilization for scenario E is more than C by about 2.1 minute. Also ED staff reaction to our work was positive and they helped us through the work but due to the reluctance of ED managers, we failed to implement the proposed changes in reality.

5 Conclusions and future work

In this research a CPN model was developed to analyze the performance of an emergency department. To evaluate the system under different conditions and improve processes, improvement scenarios were defined. These scenarios may not greatly improve the performance of the model parameters, but could be considered as an existing and potential alternative. We compare 5 scenarios by three variables.

In table 7, improvement rate of each scenario, considering its resources, presented. To have a better analysis in choosing scenarios, it is necessary to see cost and benefit of each scenario simultaneously. Another option which should be considered is ED's mission, saving patients with high acuity (ESI 1&2), and scenarios that aim to facilitate this purpose even if they cost more than other, are selected. Among defined scenarios, E and D have more improvement especially about patient with ESI 1&2. Although the cost of scenario E to scenario D is some more, but given the purpose of the improvements resulting from the scenario E, this scenario is selected as major one. In the next stage scenario D due to lower cost and also more overall improvement than the other two scenarios have been chosen as the second better scenario.

Based on the model proposed in this paper, it is possible to translate the flow diagram into a Generalized Stochastic Petri net. It would be interesting to compare the results of the two modeling approaches. That is, the exact values of different performance indices can be compared with the simulated values. Because of the hierarchical nature of the model and that every activity has a separate page belong to it; acceptance and use of this model in various conditions may seem easy and by just few change it could be localized. Using Coloured Petri net, we were able to assign different attributes to patients entered into the ED and therefore, the model traces them to calculate performance metrics. The tools and features that are available for simulating CPN models in CPN Tools e.g. hierarchy, functions, guards, state space analysis etc. made it a useful option in simulating complex systems specially healthcare. Future development to this work would be to add other attributes to tokens color such as cost of each activity in the process and engage other wards. Also, it would be of value to consider other resources including beds, facilities and equipment.

Reference

1. Buckley, B., Castillo, E., Killeen, J., Guss, D., Chan, T.: Impact of an express admit unit on emergency department length of stay. *Journal of emergency medicine* 39(5), 669-673 (Nov 2010)
2. Soremekun, O., Takayesu, J., Bohan, S.: Framework for analyzing wait times and other factors that impact patient satisfaction in the emergency department. *The Journal of Emergency Medicine* 41(6), 686-692 (2011)
3. Holden, R.: Lean Thinking in Emergency Departments: A Critical Review. *Annals of Emergency Medicine* 57(3), 265-278 (2010)
4. Eitel, , Rudkin, S., Malvey, A., Killeen, J., Pines, J.: Improving service quality by understanding emergency department flow: a white paper and position statement prepared for the american academy of emergency medicine. *The Journal of Emergency Medicine* 38(1), 70-79 (2010)
5. Zeng, Z., Ma, X., Hu, , Li, J.: A simulation study to improve quality of care in the emergency department of a community hospital. *Journal of Emergency Nursing* 38(4), 322-328 (2011)
6. Baldwin, L., Eldabi, T., Paul, R.: Simulation in healthcare management:a soft approach (MAPIU). *Simulation Modelling Practice and Theory* 12(7 - 8), 541 - 557 (2004)
7. Alvarez , A., Centeno , : Enhancing simulation models foremergency rooms using VBA. In : *Proceedings of the 1999 Winter Simulation Conference*, New York, NY, USA, vol. 2, pp.1685-1693 (1999)
8. Villamizar, J., Coelli, F., Pereira, W., Almeida, R.: Discrete-event computer simulation methods in the optimisation of a physiotherapy clinic. *Physiotherapy* 97(1), 71 - 77 (2011)
9. Hoot, N., LeBlanc, , Jones, I., Levin, S., Zhou, , Gadd, C., Aronsky, D.: Forecasting Emergency Department Crowding: A Discrete Event Simulation. *Annals of Emergency Medicine* 52(2), 116-125 (2008)
10. Kolb , E., Peck , J., Schoening , S., Lee , T.: Reducing emergency department overcrowding – five patient buffer in comparison. In : *Proceedings of the 2008 Winter Simulation Conference*, Austin, TX, pp.1516-1525 (2008)
11. Khandekar, S., Mari, J., Wang, S., Gandhi, T.: Implementation of Structural Changes to the Care Process in the Emergency Department using Discrete Event Simulation. In : *Proceedings of the 2007 Industrial Engineering Research Conference* (2007)
12. Nagula, P.: *Redesigning the patient care delivery processes at an emergency department.*, New York (2006)
13. Park, , Park, J., Ntuen, , Kim, D., Johnson, K.: Forecast Driven Simulation Model for Service Quality Improvement of the Emergency Department in the Moses H. Cone Memorial Hospital. *The Asian Journal on Quality* 9(3) (2008)
14. Khadem, M., Bashir, H., Al-Lawati, Y., Al-Azri, F.: Evaluating the Layout of the Emergency Department of a Public Hospital Using Computer Simulation Modeling: A Case Study. In : *Industrial Engineering and Engineering and Management*, Singapore, pp.1709-1713 (2008)
15. Gonzilez, , Gonzilez, , Rios, N.: Improving the quality of service in an emergency room using simulation-animation and total quality management. In : *Proceedings of the 21st international conference on Computers and industrial engineering*, Tarrytown, NY, USA , vol. 33, pp.97-100 (1997)
16. Xiong, , Zhou, M., Manikopoulos, : *Modeling and Performance Analysis of Medical*

- Services Systems Using Petri Nets. In : Systems, Man, and Cybernetics, San Antonio, TX, pp.2339-2342 (1994)
17. Chockalingam, A., Jayakumar, , Lawley, : A stochastic control approach to avoiding emergency department overcrowding. In : Proceedings of the 2010 Winter Simulation Conference, Baltimore, MD, pp.2399-2411 (2010)
 18. Dotoli, M.: Modeling and Management of a Hospital Department via Petri Nets. In : 2010 IEEE Workshop on Health Care Management (WHCM), Venice, pp.1-6 (2010)
 19. Mans, R., Schonenberg, H., Leonardi, G., Panzarasa, S., Cavallini, A., Quaglini, S., van der AALST, W.: Process Mining Techniques: an Application to Stroke Care 136. *Stud Health Technol Inform* (2008)
 20. Jørgensen, J., Lassen, K., van der Aalst, W. M.: From task descriptions via Coloured Petri nets towards an implementation of a new electronic patient record workflow system. *International Journal on Software Tools for Technology Transfer* 10(1), 15-28 (2008)
 21. CPN Tools. In: AIS group, Eindhoven University of Technology, The Netherlands. (Accessed 2012) Available at: <http://www.cpn-tools.org>
 22. Günal , M., Pidd , : Moving from specific to generic: Generic modelling in health care. In : Proceedings of the 2007 INFORMS Simulation Society Research Workshop (2007)
 23. Jensen, K., Kristensen, L.: Coloured Petri Nets , Modelling and Validation of Concurrent Systems. Springer, Denmark (2009)
 24. Jensen, K.: An introduction to the practical Use of coloured Petri nets. Lecture Notes in Computer Science, Springer-Verlag, 1-14 (1996)

ModBE'13: Poster Abstracts

Advantages of a Full Integration between Agents and Workflows

Thomas Wagner and Lawrence Cabac

University of Hamburg, MIN Faculty, Department of Informatics
<http://www.informatik.uni-hamburg.de/TGI/>

Abstract. This poster describes the notion of a *full* integration of agents and workflows. It differentiates the term from the more common *partial* integrations already well documented and researched. Finally, the advantages of a full integration are discussed.

Multi-agent systems feature a very structure-centric perspective on a software system. Agents are the main modelling abstraction, and other aspects are always seen in relation to them. Workflow systems on the other hand feature a very behaviour-centric perspective. The main abstraction here are the workflows/processes, which incorporate the data/information about other aspects. An integration of the two concepts agent and workflow can offer many advantages. These advantages represent the first outcomes in our current research on modelling systems and are the main result presented in this poster.

It is possible to differentiate between two kinds of integrations: partial and full. In a partial integration one of the concepts is used to enhance the other one. This includes agent-based workflow management systems (WFMS) and workflow-based agent management systems. Partial integrations feature only one of the two concepts main abstraction. This main abstraction may be enhanced and enriched in a number of ways, but still remains, at its core, either an agent or a workflow. This limits the potential benefits in a partial integration.

A full integration between agents and workflows aims to address that limit. In contrast to partial integrations it features both agents *and* workflows incorporated into one main modelling unit. This unit can serve as agent, workflow, or a hybrid between the two and can dynamically change its role during runtime. We call these hybrids that provide all the functionality agents and workflows would usually provide, including communication and user interaction facilities, entities. Using entities enables a system modeller to dynamically switch and mix structural and behavioural aspects of a system. This allows for a new integrated perspective on the system during development.

There are numerous examples of partial integrations. Agent-based WFMS are, for example, presented in [1,2]. A workflow-based agent management system is discussed in [3]. All of these make use of both concepts to provide an enhanced modelling experience. They still only offer extended classical workflow or agent functionality and do not feature the possibilities of a full integration. To the best of our knowledge there are no examples of full integrations.

A full integration between agents and workflows exhibits a number of advantages to the system modeller. These assume an efficient and comprehensive implementation of a full integration system (see last paragraphs for outlook).

Abstraction The abstraction of the individual concepts into one unified entity enables a freedom to work with dynamic and hybrid constructs. Entities can operate as agents, workflows, or something in between. They can dynamically adapt to the requirements before, acting as an agent at one point and processing like a workflow at another. Entities naturally and directly incorporate any and all mechanisms, facilities, and properties of agents and workflows. Consequently, providing these characteristics in dynamic ways becomes far easier.

Flexibility Allowing a modeller to use agents and workflows on the same abstraction level, allows to model a system along the two dimensions *structure* and *behaviour*. Classically, only one of these dimensions was in the foreground, while modelling aspects of the other was heavily biased by the original dimension. This two-dimensional modelling enables a modeller to utilise the dynamic interaction between agents and workflows on a conceptual level.

Simplicity A full integration offers the combined capabilities of agents and workflows. It does so by providing simple-to-use and predefined constructs (entities) which allow a modeller to make full use of the strengths of agents **and** workflows. The entities in themselves can be used similarly to agents and workflows, but possess a larger spectrum of capabilities.

Expressiveness A full integration cannot necessarily express more than the classical paradigms. However, in the classical paradigms complex helper constructs might be necessary to implement more complicated structures available directly in a full integration. This means that a full integration is capable of expressing more constructs in a natural and simple way.

Enrichment The enrichment aspect, the main advantage of partial integration, is also applicable in a full integration. In fact, it is even more emphasised, since an entity can be improved from both its agent and workflow side.

Concerning future work the provision of a comprehensive implementation of a full integration is the main focus. Currently, the work is centred on establishing a working prototype as proof-of-concept. In conclusion, a full integration offers many beneficial advantages in comparison to classical systems. When extensively and efficiently implemented, it is a powerful tool for a system modeller to use.

References

1. P. Czarnul, M. Matuszek, M. Wójcik, and K. Zalewski. BeesyBees - efficient and reliable execution of service-based workflow applications for BeesyCluster using distributed agents. In *Proceedings of IMCSIT 2010*, pages 173–180, oct. 2010.
2. F. Hsieh. Collaborative Workflow Management in Holonic Multi-Agent Systems. In J. O'Shea et al., editors, *Agent and Multi-Agent Systems: Technologies and Applications*, volume 6682 of *LNCS*, pages 383–393. Springer Berlin Heidelberg, 2011.
3. A. Mislevics and J. Grundspenkis. Workflow based approach for designing and executing mobile agents. In *Digital Information Processing and Communications (ICDIPC), 2012 Second International Conference on*, pages 96–101, july 2012.

Cloud Transition for QoS Modeling of Inter-Organizational Workflows

Sofiane Bendoukha and Lawrence Cabac

University of Hamburg, Department of Informatics
<http://www.informatik.uni-hamburg.de/TGI/>

Abstract. In this paper we present an architecture for enabling complex workflow execution in Cloud-like environments. We focus mainly on modeling concepts and techniques to enhance accessibility to Cloud services by different kind of users.

Complex workflow tasks need in some cases to be mapped to distributed resources and involves the cooperation between several partners. Workflow management is critical to a successful long-term Cloud computing strategy. The notion of inter-organizational workflow still needs conceptual and technical support especially in complex and dynamic environments like Clouds. New ways to tackle this problem have to be found. Therefore, existing workflow architectures need to be adapted for the Cloud and workflow management systems (WfMS) should be integrated with Cloud infrastructure and resources [3].

In this paper we use Inter-Cloud Workflow Petri Nets (ICWPN), an approach for enabling workflows in an (Inter)-Cloud environment. A specialized *Cloud Task Transition* (CTT) is introduced to facilitate the connection to the Cloud and to support Quality of Service (QoS) management [1]. The CTT (see Fig. 1 (a)) is based on the Workflow Task Transition [2], which is the core of the workflow net formalism in RENEW¹ (**R**eference **N**et **W**orkshop). Workflow modelers specify their requirements as parameters to the CTT in form of tuples (S, Q, I), which correspond respectively to the Cloud service (S) that they want to use (it can be a storage or a compute service), the QoS constraints (Q) consisting of deadlines or costs and input data (I) consisting either of required files in case of a storage or scripts if they want to execute their codes on the Cloud. Synchronous channels are used to make the connection with the WfMS, which controls the completion of the task. It either initiates the firing or cancels it and all input parameters are put back onto the input places.

To see how the CTT is used in practice, we introduce a Cloud-based workflow architecture, it is depicted in Fig. 1 (b). It includes three basic layers from top to bottom: *user applications layer* (UL), *middle-ware layer* (ML) and the *resource layer* (RL), which consists mainly of Cloud services. In our approach we view the process of executing an application in an Inter-Cloud environment as a 6-phase process: (1) Users use the offered modeling tools consisting mostly of RENEW and the introduced CTT to specify the requirements (Cloud services, QoS constraints, specific input data) for their applications using Petri nets models. (2)

¹ Renew is available at <http://www.renew.de>

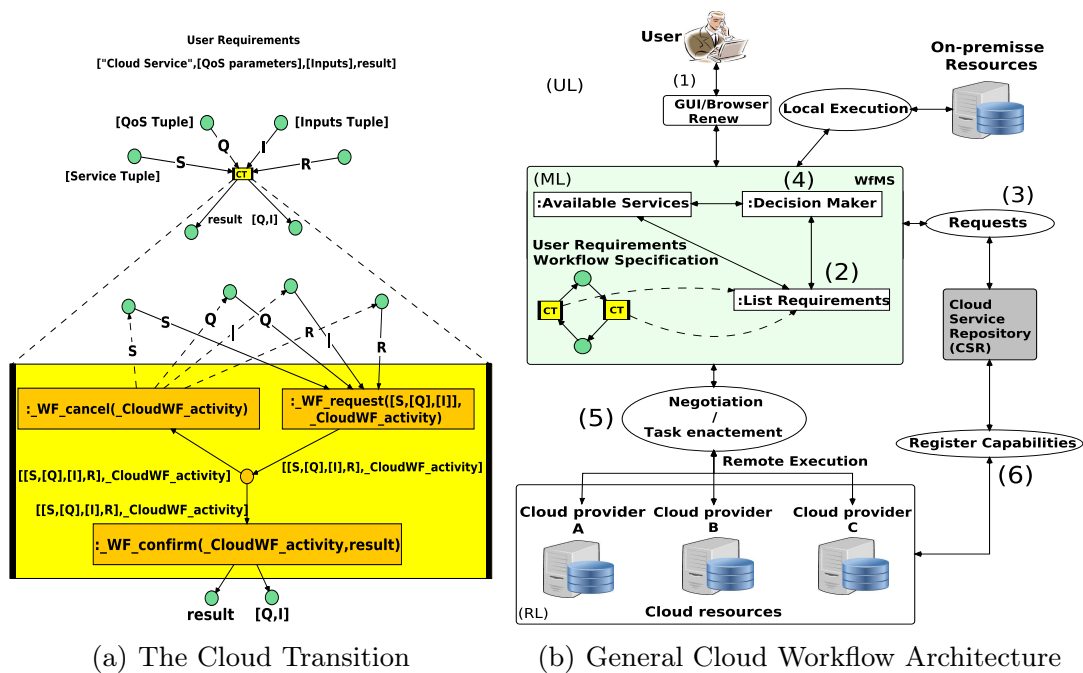


Fig. 1. Cloud-based Workflow Management

A list of requirements is created consisting of required services as well as their related QoS constraints. (3) Make a request to the *Cloud Service Repository* (CSR) which is accessible by the WfMS to achieve workflow tasks (4) Based on the above steps (2-3) a decision is made by the *Decision Maker* who determines whether the workflow tasks will be executed locally or using Cloud resources. (5) After that the workflow tasks are mapped to the adequate resources. (6) When the workflow is deployed, information about Cloud providers and the state of their services are constantly updated.

Here we focused primarily on Cloud technologies. Nevertheless, the introduced model (see Fig. 1(b)) can be also applicable to other dynamic domains where distributed resources are shared and dynamically allocated and usually priced.

References

1. Sofiane Bendoukha and Thomas Wagner. Cloud transition: Integrating cloud calls into workflow Petri nets. In Lawrence Cabac, Michael Duvigneau, and Daniel Moldt, editors, *PNSE*, volume 851 of *CEUR Workshop Proceedings*, June 2012.
2. Thomas Jacob, Olaf Kummer, Daniel Moldt, and Ulrich Ultes-Nitsche. Implementation of workflow systems using reference nets – security and operability aspects. In Kurt Jensen, editor, *Fourth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, August 2002.
3. Suraj Pandey, Dileban Karunamoorthy, and Rajkumar Buyya. *Workflow Engine for Clouds*, pages 321–344. John Wiley & Sons, Inc., 2011.