Models for the Supervision of Web Services Orchestration with Dynamic Changes*

Thomas Chatain Claude Jard

IRISA/ENS Cachan-Bretagne, Campus de Beaulieu,
F-35042 Rennes cedex, France
E-mail: {Thomas.Chatain, Claude.Jard}@irisa.fr

Abstract

Programming on the Web enlights some classical problems encountered on large distributed applications with a particular emphasis on dynamic changes. In that context, we are interested in the questions of supervision and diagnosis. Our approach is based on true-concurrency models and consists in building an unfolding of a model of the supervised system, that selects the histories that explain the observed alarms. In this paper we extend the notion of unfolding of high-level Petri nets to a model of dynamic systems that we define. This model is close to high-level Petri nets and allows us to model dynamicity. Finally we explain how to use unfoldings of dynamic nets for the diagnosis application.

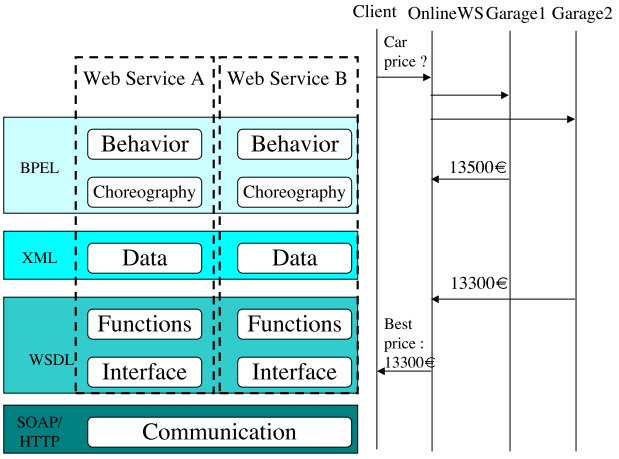


Figure 1. Business processes and Web services (left part) and an example of peer-to-peer orchestration (right part).

1. Introduction

Until now, the services provided on the Internet were based on the interaction between a client (browser) and a server through the HTTP protocol. More recently, the need to provide services that are usable by programs has launched the new model of *Web services*. A Web service is a self-describing, self-contained modular application that can be described, published, located, and invoked over a network, e.g. the World Wide Web. A Web service performs an encapsulated function ranging from a simple request-reply to a full business process. It appears that the description of Web services and their orchestration is very similar to the description of distributed workflows in the context of business processes. Figure 1 sketches the mapping between the business terms and the Web service technologies. A reference in this domain is the language WS-BPEL [5] (Business Process Language Language for Web Services).

In this document, we model services with high-level Petri nets. We model the steps in the procedure as transitions and precedence as arcs in the net. We assume that the reader has basic knowledge of Petri nets (see Figure 2 for an example). Data are treated using the symbolic aspects of high-level Petri nets. Those are also the basis for the representation of dynamic changes. The use of such kind of formal model has already been advocated in [6] and [9] in the context of workflow specifications. The main originality here is to consider a very powerful version of high-level Petri nets to deal with structural dynamic changes and to define a notion of unfolding for these nets for the purpose of on-line supervision. We have chosen to work in a Petri nets framework because our approach is based on unfoldings, which have been mainly developped in this context. Other formal models of concurrency could be used, like automata networks or process algebras, provided that they are executable and give an explicit notion of concurrency.

We are concerned with dynamic changes in the struc-

*This work is partially funded by the french RNRT Swan project.

ture of services; we are not concerned here with changes to the value of an application data variable. Dynamic means that we are required to make the change "on the fly" in the midst of continuous execution of the changing procedure. Web services must make structural changes such as adding a new service, removing an old service, or changing a given service behaviour depending on the current state of the system (self-adaptation with respect to intermittent resources, reconfiguration in case of QoS problem).

We have been developing for several years a diagnosis approach for distributed systems [2], based on dynamic unfoldings of Petri nets [7], which explicits causalities and concurrencies linking the observed events (e.g. alarms) in the explanations. But simple Petri nets are too limited to deal with data aspects, even for very simple systems. High-level Petri nets [8] provide an interesting extension. In [4], we have defined and used the notion of symbolic unfolding of high-level nets to supervise partially observable distributed systems.

This model is relevant for static systems, but must be extended to deal with dynamic systems whose the structure evolves. The proposed extensions are in the same vein that those described in [3]. In addition, they include "reading arcs" [1] to model non-destructive accesses to data. They also play a role to represent dynamicity. We propose a notion of unfolding for this model. We show that the diagnosis problem can be expressed as the computation of an unfolding constrained by the observations, in order to retain only trajectories that explain them.

The paper is organized as follows. We first present our model of dynamic net, illustrated with a simple example of Web service. We then define the notion of unfolding for this type of net. The last section before conclusion presents the way to use the unfolding in order to select the different partial order trajectories that explain a given set of observations.

2. Dynamic nets

The system state is represented as a multiset of tokens, called marking. Each token carries a value. Transitions are used to change the current state. When it fires, a transition removes tokens from the current marking and create new tokens. It is also possible to only test the presence of tokens without consuming then by using read arcs. In standard high-level Petri nets, the set of transitions is static. In contrast, in our dynamic model, transitions are particular tokens included in the current marking. As any token, they can be created or removed. This allows us to model dynamic systems, in which the set of system components can evolve during execution. Furthermore, our dynamic model does not use the notion of place: the presence of a token of value c in the place p in a standard high-level Petri net can be simulated by the presence of a token of value (c, p) in the marking. This avoids the difficulty to consider the dynamicity of places.

2.1. Definition

For a set X, we denote by 2^X the set of subsets of X, and by $X^{\oplus}$ the finite multisets on X. We write the multisets between the separators $\{\!|\ \cdot\ |\!\}$ and we denote by $\oplus$ and $\ominus$ the sum and difference between multisets.

A *dynamic net* (or net) is a tuple $N \stackrel{\text{def}}{=} (\mathit{Tok}, \mathit{PAR}, \iota, W)$, where *Tok* is a set of tokens, *PAR* is a set of values used as parameters and ι and W are maps such that:

- $\iota \in \mathit{Tok} \longmapsto (2^{\mathit{Tok}} \times \mathit{Tok}^{\oplus} \times \mathit{PAR}) \longmapsto \textbf{bool}$
- $W \in \mathit{Tok} \longmapsto (2^{\mathit{Tok}} \times \mathit{Tok}^{\oplus} \times \mathit{PAR} \longmapsto \mathit{Tok})^{\oplus}$

A marking is a multiset of tokens $M \in \mathit{Tok}^{\oplus}$. A token $t \in \mathit{Tok}$ is called *transition* if it exists $r \in 2^{\mathit{Tok}}$, $c \in \mathit{Tok}^{\oplus}$ and $p \in \mathit{PAR}$, such that $\iota(t)(r, c, p)$. Furthermore, if $(\{t\} \cup r) \oplus c$ is included in M, then transition t can fire with the parameter p, reading the tokens of r, and consuming the tokens of c. We obtain a new marking $M' \stackrel{\text{def}}{=} M \ominus c \oplus W(t)(r, c, p)$ (we write $W(t)(r, c, p)$ instead of $\{\!|w(r, c, p) \mid w \in W(t)|\!\}$.

We make the assumption that all transitions consume tokens. To start the execution of the dynamic net, we add an initial transition, $\perp$, which does not read nor consume any token. For simplicity we assume that $\perp$ can fire only with the value $\bullet$ as parameter. The initial transition is not considered as a token and never appears in the marking. The set of transitions is denoted by T.

2.2. Example of a Web services orchestration

We propose to illustrate our model and the diagnosis application by a simple example. It is graphically drawn in Figure 2. Its formal description is given under the figure.

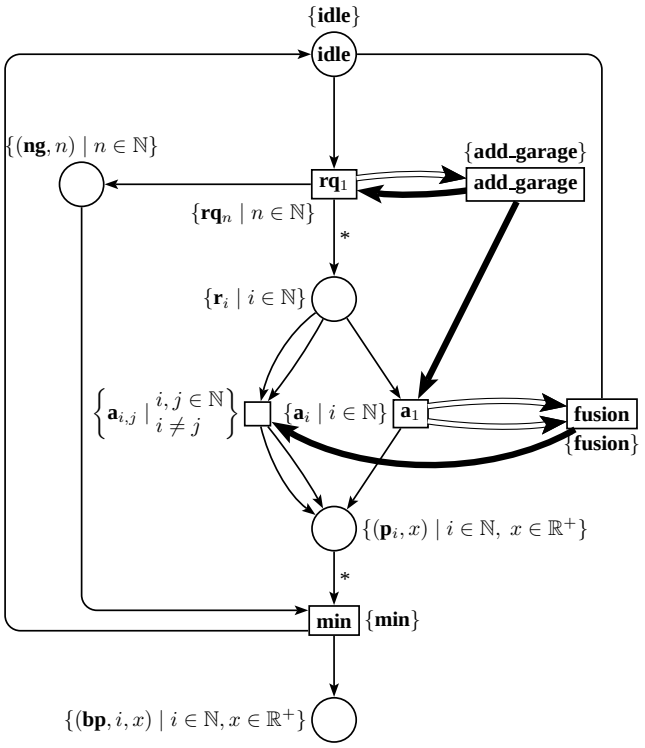
For the graphical representation, we have split the set of tokens in two categories: the transitions and the others. Each square box represents a set of transitions and each circle represents a set of other tokens. Tokens currently present in the net are figured inside the boxes and the circles. Arrows figure consumption and creation of tokens by the transitions. An arrow is labelled by a star if it may represent the consumption or creation of several tokens. An arrow, linking a transition t to a transition t' can figure either the consumption of t by t', or the creation of t' by t. We remove the ambiguity by drawing a fat empty arrow in the first case and a full one in the second case.

Figure 2 shows a model for a distributed system where a server selects the best price for a car rent from several garages each time a client asks for it. There is initially one garage, but new garages can be added dynamically at any time. We detail this aspect later. For the moment consider we are in a situation where there are n garages. When a request arrives, the server sends a request to each of the n garages. This action is represented by the transition $\mathbf{rq}_n$. Then all the garages are supposed to answer concurrently and give their price. The answer of the i^{th} garage is represented by the transition $\mathbf{a}_i$, that creates a token $\mathbf{p}_i$. When all the garages have answered, the server selects the best price (transition **min**, that creates a token $(\mathbf{bp}, i)$ where i is the number of one of the garages that gave the best price). At any time a new garage may be added. The transition **add_garage** models this: it creates a transition $\mathbf{a}_{n+1}$ that represents the answer of the new garage (number $(n+1)$). Additionally, the transition $\mathbf{rq}_n$ is replaced by $\mathbf{rq}_{n+1}$, so that the next time a client asks for the price of a car, all the garages (including the new one) will be requested. Because a garage can be added even while some garages are answering a request, the number of requested garages is stored (token $(\mathbf{ng}, n)$ in the model) when the requests are sent. Thus before selecting the best price, the server can check that all the requested garages have answered, and it will not wait for answers of new garages.

In addition to the behaviour that was described, any two garages, say the i^{th} and the j^{th}, may choose to agree on the price. In this case they synchronize to give their answer together. In our model, the synchronization is represented by the transition **fusion**, which deletes the transitions $\mathbf{a}_i$ and $\mathbf{a}_j$ and replaces them by a transition $\mathbf{a}_{i,j}$ which gives the two answers simultaneously with the same price. The read arc from the **idle** token from the **fusion** transition models the fact that the fusion cannot happen when the server is waiting for answers from the garages. Figure 3 shows the structure of the system with two garages in two different situations where they have not or have respectively agreed on the price.

3. Unfolding

Our approach of the supervision problem is to consider that some events of the system are observed and correspond to some transitions of the formal model. The problem is to infer the causal dependencies between these observed events. This is achieved by a kind of execution of the model, guided by the observations, and theoretically based on what is called a net unfolding, which we define formally in this section. The unfolding of our example is shown in figure 4.



$\iota(\mathbf{fusion})(r,c,p) \stackrel{\text{def}}{=} r = \{\mathbf{idle}\} \wedge c = \{\!|\mathbf{a}_i, \mathbf{a}_j|\!\} \wedge p = \bullet$ (with $i, j \in \mathbb{N}$ and $i \neq j$)
$W(\mathbf{fusion})(\{\mathbf{idle}\}, \{\!|\mathbf{a}_i, \mathbf{a}_j|\!\}, p) \stackrel{\text{def}}{=} \{\!|\mathbf{a}_{i,j}|\!\}$

$\iota(\mathbf{add\_garage})(r,c,p) \stackrel{\text{def}}{=} r = \emptyset \wedge c = \{\!|\mathbf{rq}_n|\!\} \wedge p = \bullet$ (with $n \in \mathbb{N}$)
$W(\mathbf{add\_garage})(\emptyset, \{\!|\mathbf{rq}_n|\!\}, p) \stackrel{\text{def}}{=} \{\!|\mathbf{rq}_{n+1}, \mathbf{a}_{n+1}|\!\}$

$\iota(\mathbf{rq}_n)(r,c,p) \stackrel{\text{def}}{=} r = \emptyset \wedge c = \{\!|\mathbf{idle}|\!\} \wedge p = \bullet$
$W(\mathbf{rq}_n)(\emptyset, \{\!|\mathbf{idle}|\!\}) \stackrel{\text{def}}{=} \{\!|(\mathbf{ng}, n), \mathbf{r}_1, \ldots, \mathbf{r}_n|\!\}$

$\iota(\mathbf{a}_i)(r,c,p) \stackrel{\text{def}}{=} r = \emptyset \wedge c = \{\!|\mathbf{r}_i|\!\} \wedge p = \bullet$
$W(\mathbf{a}_i)(r,c,p) \stackrel{\text{def}}{=} \{\!|\mathbf{p}_i|\!\}$

$\iota(\mathbf{a}_{i,j})(r,c,p) \stackrel{\text{def}}{=} r = \emptyset \wedge c = \{\!|\mathbf{r}_i, \mathbf{r}_j|\!\} \wedge p = \bullet$
$W(\mathbf{a}_{i,j})(r,c,p) \stackrel{\text{def}}{=} \{\!|\mathbf{p}_i, \mathbf{p}_j|\!\}$

$\iota(\mathbf{min})(r,c,i) \stackrel{\text{def}}{=} \begin{cases} r = \emptyset \wedge c = \{\!|(\mathbf{ng}, n), \mathbf{p}_1, \ldots, \mathbf{p}_n|\!\} \\ \wedge i \in \{1, \ldots, n\} \end{cases}$ (with $n \in \mathbb{N}$)
$W(\mathbf{min})(\emptyset, \{\!|(\mathbf{ng}, n), \mathbf{p}_1, \ldots, \mathbf{p}_n|\!\}) \stackrel{\text{def}}{=} \{\!|(\mathbf{bp}, i)|\!\}$

$\iota(\bot)(r,c,p) \stackrel{\text{def}}{=} r = \emptyset \wedge c = \emptyset \wedge p = \bullet$
$W(\bot)(\emptyset, \emptyset, p) \stackrel{\text{def}}{=} \{\!|\mathbf{idle}, \mathbf{rq}_1, \mathbf{a}_1, \mathbf{min}, \mathbf{add\_garage}, \mathbf{fusion}|\!\}$

Figure 2. A dynamic WS orchestration where a server selects the best price for a car rent from several garages.

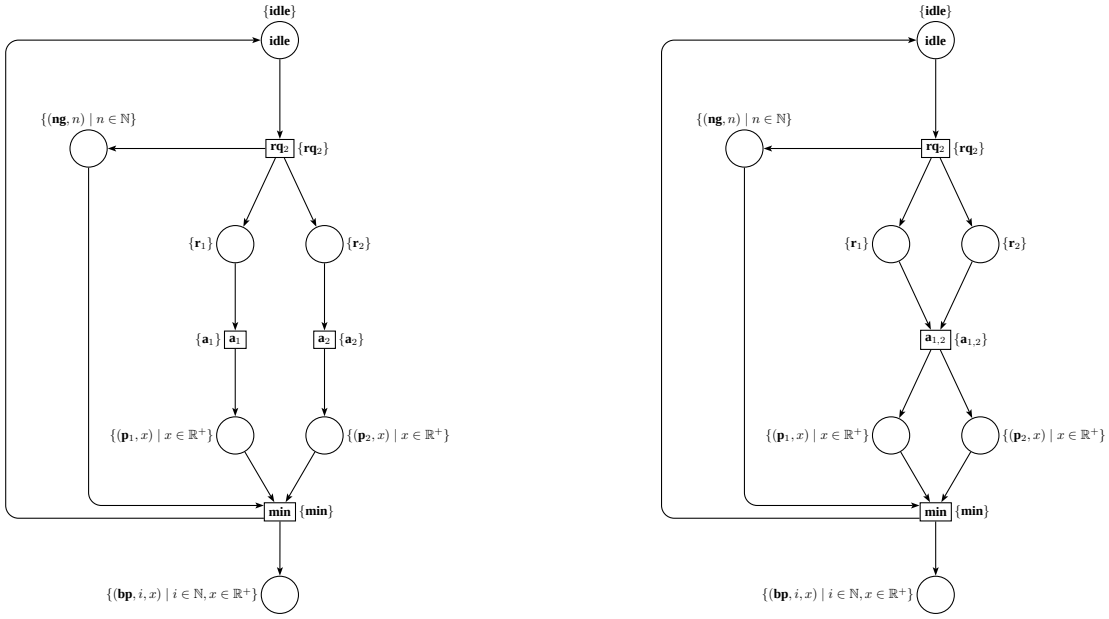


Figure 3. Two independent garages (left part) and two garages that have agreed on the price (right part).

3.1. Events

For a given dynamic net $N \stackrel{\text{def}}{=} (\mathit{Tok}, \mathit{PAR}, \iota, W)$, we define inductively the set of events E by:

- the initial event $e_\perp \in E$. We denote $\tau(e_\perp) \stackrel{\text{def}}{=} \perp$ and $\underline{e_\perp} = {}^\bullet e_\perp \stackrel{\text{def}}{=} \emptyset$; for all $w \in W(\perp)$, we denote $\mathit{val}(e_\perp, w) \stackrel{\text{def}}{=} w(\emptyset, \emptyset, \bullet)$.
- let $p \in \mathit{PAR}$ and $b, r_1, \ldots, r_n, c_1, \ldots, c_m$ be pairs under the form (f, w), where $f \in E$ and $w \in W(\tau(f))$. If $\iota(\mathit{val}(b))(\{\mathit{val}(r_1), \ldots, \mathit{val}(r_n)\}, \{\!|\mathit{val}(c_1), \ldots, \mathit{val}(c_m)|\!\}, p)$ then $e = (b, \{r_1, \ldots, r_n\}, \{c_1, \ldots, c_m\}, p) \in E$. We denote $\tau(e) \stackrel{\text{def}}{=} \mathit{val}(b)$, $\underline{e} \stackrel{\text{def}}{=} \{b, r_1, \ldots, r_n\}$ and ${}^\bullet e \stackrel{\text{def}}{=} \{c_1, \ldots, c_m\}$. For all $w \in W(\mathit{val}(b))$, we denote $\mathit{val}(e, w) \stackrel{\text{def}}{=} w(\{\mathit{val}(r_1), \ldots, \mathit{val}(r_n)\}, \{\!|\mathit{val}(c_1), \ldots, \mathit{val}(c_m)|\!\}, p)$

A pair (e, w), with $e \in E$ and $w \in W(\tau(e))$ is called *condition* and codes the presence of a token of value $\mathit{val}(e, w)$ at a given time of the history (between its creation and its consumption).

Each event e creates a set of conditions defined by $e^\bullet \stackrel{\text{def}}{=} \{(e, w) \mid w \in W(\tau(e))\}$. An event $e = (b, R, C, p)$ codes the firing of transition $\mathit{val}(b)$ with the parameter p, reading the conditions of R and consuming the conditions of C. Events are depicted by black rectangles. The input arrows link the consumed conditions ${}^\bullet e$, while the read conditions $\underline{e}$ are linked with lines (without arrow). The conditions created by the event e are linked with the output arrows. In Figure 3, the only initial event is depicted on the top. Its output arrows point to conditions. Some of them are transitions: they are written inside white rectangles. The other conditions are written within ellipses. For each condition (e, w), the value $\mathit{val}(e, w)$ is written in the corresponding rectangle or ellipse.

3.2. Causality and conflict

Let us consider the relations $\rightarrow$, $\rightsquigarrow$ and $\nearrow$ on E, defined as follows:

- $e \rightarrow e'$ iff $e^\bullet \cap (\underline{e'} \cup {}^\bullet e') \neq \emptyset$
- $e \rightsquigarrow e'$ iff $\underline{e} \cap {}^\bullet e' \neq \emptyset \vee (e \neq e' \wedge {}^\bullet e \cap {}^\bullet e' \neq \emptyset)$
- $e \nearrow e'$ iff $e \rightarrow^+ e' \ \vee \ e \rightsquigarrow e'$

We say that two events e and e' are *causally related* if $e \rightarrow^* e'$, where $\rightarrow^*$ denotes the reflexive transitive closure of $\rightarrow$. For an event $e \in E$, we define the *past* of e: $\lceil e \rceil \stackrel{\text{def}}{=} \{f \in E \mid f \rightarrow^* e\}$ and for all $F \subseteq E$, $\lceil F \rceil \stackrel{\text{def}}{=} \bigcup_{f \in F} \lceil f \rceil$.

Note that the unfolding contains all the possible histories of the system. We say that a set of events are in *conflict* if

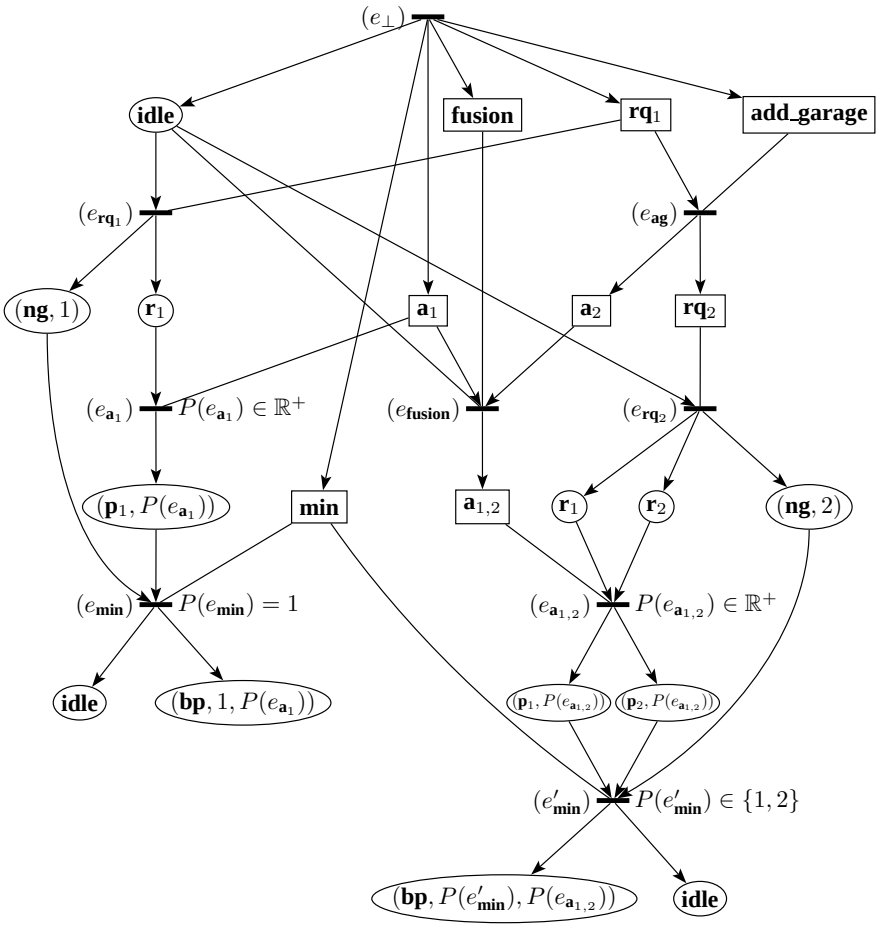


Figure 4. A prefix of the unfolding of the model of Figure 2.

all of them cannot occur in the same history. The events of the set $F \subseteq E$ are in *conflict* if

$$\exists e_0, e_1, \ldots, e_n \in \lceil F \rceil \quad e_0 \nearrow e_1 \nearrow \ldots \nearrow e_n \nearrow e_0$$

The *unfolding* of N is the set of events e such that $\{e\}$ is not in conflict.

3.3. Algorithm

The algorithm proceeds by non-deterministic iterations, after the placement of the initial event $e_\perp$. At each iteration, it chooses a transition $t \in T$, a condition b such that $val(b) = t$, a parameter p and the sets of conditions that are read (R) and consumed (C). Then it places an event e corresponding to the firing of transition t, if $\{e\}$ is not in conflict and if the guard $\iota(t)(\{val(b) \mid b \in R\}, \{\!|val(b) \mid b \in C|\!\}, p)$ is satisfied.

Notice that in the application to supervision, the net we try to unfold is constrained by the observations (see Section 4). This implies that the unfolding is finite and ensures the termination of the algorithm if the model does not contain any loop of non-observable transitions.

3.4. Example

Figure 4 shows a prefix of the unfolding of the model of Figure 2. The token **idle**, which indicates that no request is en route, and the transitions $\mathbf{rq}_1$, $\mathbf{a}_1$, **min**, **add_garage** and **fusion** are in the initial marking. In the unfolding, arrows indicate that they are created by the initial event $e_\perp$, represented in top of the figure. In the initial state, a client may ask for the price of a car (event $e_{\mathbf{rq}_1}$). The unique initial garage is requested and answers (event $e_{\mathbf{a}_1}$). The system also allows the addition of a new garage (event $e_{\mathbf{add\_garage}}$). If the event $e_{\mathbf{rq}_1}$ occurs in the same history, then it must occur before $e_{\mathbf{add\_garage}}$. The asymmetric conflict that appears in the unfolding between $e_{\mathbf{rq}_1}$ and $e_{\mathbf{add\_garage}}$ forces this ordering.

After the occurrence of $e_{\mathbf{add\_garage}}$, the transition $\mathbf{rq}_1$ does not exist any more but is replaced by the transition $\mathbf{rq}_2$, which sends a request to two garages. Moreover, the

transition $\mathbf{a}_2$ is added. If $\mathbf{rq}_2$ fires (event $e_{\mathbf{rq}_2}$), it will send a request to both the first and the second garage. They may answer concurrently (this is not represented on Figure 3) or answer simultaneously (event $e_{\mathbf{a}_{1,2}}$) provided that they have agreed about the price (event $e_{\mathbf{fusion}}$).

4. Diagnosis

The unfolding represents the different possible histories of the system. For the purpose of diagnosis, we want to select only the histories that explain the set of observations. This can be achieved by synchronizing the different observations with the corresponding transitions of the model (see [4]). The alarms become new tokens. The unfolding of the constrained net can be computed by the algorithm of Section 3.3.

In our example two sensors A and B record alarms in the system. When the server sends requests to the garages (transition $\mathbf{rq}_n$, for any n), an alarm ρ is emitted. When a garage answers a request (transition $\mathbf{a}_i$, for any i) or when two garages answer synchronously (transition $\mathbf{a}_{i,j}$, for any i, j), an alarm α is emitted. When the server selects the best price (transition **min**), it emits an alarm μ. All these alarms are observed by sensor A. Only the addition of a new garage (transition **add_garage**) emits an alarm γ which is observed by sensor B. The fact that two garages agree about the price (transition **fusion**) is not observable.

Consider that sensor A has observed the alarms ρ, α, μ and that sensor B has observed one alarm γ. There are two explanations to these observations. Both appear in Figure 3: possibly the new garage was added after the request was sent to the first garage. Then the answer comes from the first garage. Otherwise the new garage was added earlier and the requests were sent to both garages. In this case they must have agreed about the prices, as a single answer has been observed. The garage that is selected as the best can be either the first or the second. Both appear in the unfolding.

5. Conclusion

We have considered the new context of orchestrations of Web services. In that context, dynamic changes in Web services become an important aspect. We have presented an approach for a possible supervision and diagnosis of such systems. It is based on the use of unfoldings of dynamic Petri nets. The unfolding, guided by a finite set of observations allows the diagnoser to select the different possible explanations, knowing that not all the events are observable.

In the example given in this article, the alarms give quite few information on the system. Thus there are multiple explanations even for short observations. This allows us to illustrate the method with a small unfolding. But it would also be possible to supervise systems in which the alarms give much richer information. For example in realistic systems, the structural changes are often observable. As a result there are few explanations even for long observations.

In this paper we have used a notion of unfolding where we create a distinct event for each possible value for the parameter of a transition. This is possible only when each transition can fire with a finite (and preferably small) number of parameters. The reverse would force us to use symbolic unfoldings [4]. For simplicity we have not used them in this article as they were not necessary. The algorithm to compute the symbolic unfolding of a dynamic net has to decide the satisfiability of predicates associated with the events. This is possible if the transition guards and the values of the created tokens are described using a language of a weak enough theoretical power. Several frameworks can be envisioned. For instance, we can restrict to the Presburger arithmetics (arithmetics without multiplication).

We are currently implementing the algorithms and we plan to apply them on examples in the field of telecommunications.

References

[1] P. Baldan, A. Corradini, and U. Montanari. Contextual petri nets, asymmetric event structures, and processes. *Inf. Comput.*, 171(1):1–49, 2001.

[2] A. Benveniste, S. Haar, E. Fabre, and C. Jard. Distributed monitoring of concurrent and asynchronous systems. In *CONCUR*, pages 1–26, 2003.

[3] M. G. Buscemi and V. Sassone. High-level petri nets as type theories in the join calculus. In *FoSSaCS*, pages 104–120, 2001.

[4] T. Chatain and C. Jard. Symbolic diagnosis of partially observable concurrent systems. In *FORTE*, pages 326–342, 2004.

[5] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business process execution language for web services, version 1.1. Technical report, BEA Systems, IBM and Microsoft, May 2003.

[6] C. A. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In *COOCS*, pages 10–21, 1995.

[7] J. Engelfriet. Branching processes of petri nets. *Acta Inf.*, 28(6):575–591, 1991.

[8] K. Jensen. *Coloured Petri nets: basic concepts, analysis methods and practical use*. Springer-Verlag, 1995.

[9] G. Joeris and O. Herzog. Managing evolving workflow specifications. In *CoopIS*, pages 310–321, 1998.