A chronicle-based diagnosability approach for discrete timed-event systems: application to Web-Services

Yannick Pencolé and **Audine Subias**
CNRS; LAAS; 7 avenue du colonel Roche F-31077 Toulouse, France
Université de Toulouse; UPS,INSA,INP,ISAE; LAAS; F-31077 Toulouse, France
ypencole@laas.fr, subias@laas.fr

Abstract: This paper addresses the problem of diagnosability analysis in Web Services. In particular, it focuses on the analysis of the impact of time to the diagnostic capabilities in Web Service workflows. The diagnosability analysis that is proposed in this paper aims at determining the diagnostic capabilities of a previously developped algorithm for the diagnosis of Web Services. This diagnostic algorithm is based on chronicle recognitions. Faults that can occur during the execution of service workflows are described by means of chronicles.

To perform this diagnosability analysis, the problem is firstly defined as a language-based analysis which leads to the definition of exclusiveness tests between the languages represented by the chronicles. To deal with the time aspects inherent to the chronicles, we then propose to perform the automatic analysis by the use of time Petri nets. Exclusiveness tests are then defined on reachability graphs of time Petri nets which implicitly represent chronicle languages.

Key Words: Discrete-event systems, web service, workflow, diagnosis, diagnosability, chronicle, time Petri nets,

1 Introduction

Nowadays, Web Services appear as one promising solution for business integration. However, despite the growing interest, an important challenge still exists to make Web Services a dependable technology. Developing self-healing Web Services is still an open issue. A self-healing Web Service is able to monitor itself, to diagnose the causes of a failure and to recover from the failure, where a failure can be either the inability to provide a given service, or a loss in the service quality. The WS-DIAMOND [1] project is a step in this direction. The first goal of this project was to develop a framework for self-healing service execution of complex Web Services, where monitoring, detection and diagnosis of anomalous situations is carried on and repair/reconfiguation is performed so that reliability and availability of Web Services are guaranteed. The second goal of the project was to devise guidelines for designing services in such a way that they can be easily diagnosed and recovered during their execution. As the possibility to perform accurate diagnosis (fault identification or even only detection) during execution

[1] This work results from the WS-DIAMOND project funded by the European Commission, IST, FET-OPEN framework, under grant IST-516933.

depends on the available observations, this second objective supposes diagnosability analysis to provide information about the classes of behavior of the system under different anticipated fault situations. Diagnosability is the property of a system to generate observations that allow to detect and discriminate faults in a finite time after their occurrences.

The work presented in this article focuses on this diagnosability aspect in the field of Web Services. As observations and faults may be represented in different ways that depend on the considered diagnosis approach, diagnosability analyses are also related to the approach that is used to diagnose the system. In this paper, we study the impact of time to the diagnostic capabilities in Web Service workflows and thus focus our study on a particular model-based diagnosis approach called *monitoring approach* [D4.2, 2006]: this algorithm developed in the WS-DIAMOND framework and presented in [Cordier et al., 2007], is a *chronicle-based* approach; it determines whether faults have occurred in the execution of a workflow by the recognition of predefined chronicles (graph of events with temporal contraints).

Starting from this diagnostic algorithm, we propose in this paper to firstly define the diagnosability property that states whether the chronicle-based algorithm of [Cordier et al., 2007] will perform well or not on a given workflow. This diagnosability property is defined as a property over event languages that represent execution workflows and recognised chronicles. Then, we propose an automatic way to analyse the diagnosability of a given workflow by the use of time Petri nets which is a suitable formalism to implicitly represent the languages associated with execution of workflows and chronicles.

The paper is organized as follows. Section 2 briefly introduces the notion of monitoring and diagnosis of Web Services followed by a more detailed description of the underlying diagnostic approach based on chronicles. Before defining the diagnosability problem over this chronicle approach, Section 3 first recalls the general diagnosability problem for discrete event systems and then defines diagnosability for timed discrete event systems. Section 4 then shows how to test diagnosability in the framework defined by the chronicle-based diagnostic algorithm. Then, an exclusiveness test for analysing the workflow diagnosability is derived in Section 5. This section particularly shows that, due to the time aspects inherent to the use of chronicles, the set of behaviors to handle for diagnosability purposes are possibly infinite and thus require an adequate and finite representation. The use of time Petri nets to model chronicles are then discussed in this section. Finally, Section 6 presents a diagnosability checker tool. The use of this checker is illustrated on a foodshop Web Service [D4.2, 2006].

2 Monitoring and Diagnosis of Web Services

2.1 General principle

In the monitoring approach, the diagnosis relies on an event-based method. Workflows of web service orchestrations are thus seen as event-based systems.

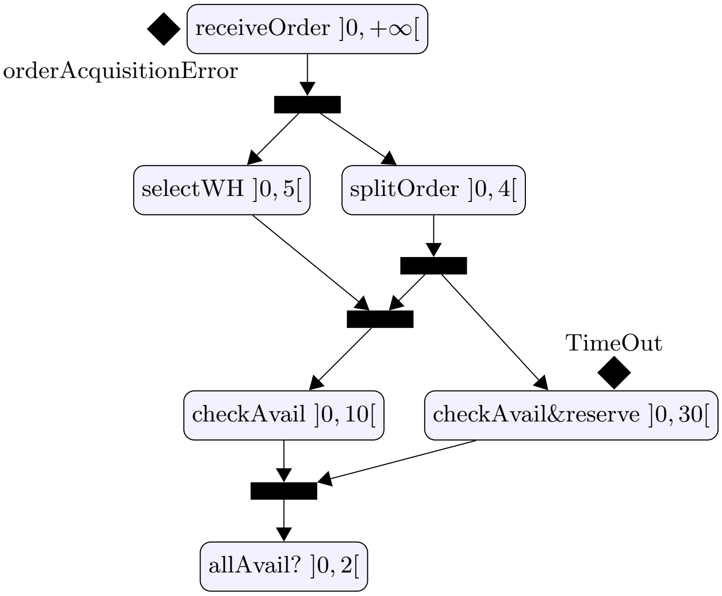


Figure 1: Extract workflow of a shop service

Figure 1 gives an illustrative example of workflow. The duration of each activity is given by a time interval. For instance, the activity *receiveOrder* waits till an order is received by the client, thus it may not end (unbounded time interval). Once this first activity has finished, the activities *selectWH* (select warehouse) and *splitOrder* start: *selectWH* lasts in the worst case 5 time units whereas *splitOrder* lasts in the worst case 4 time units.

A *workflow* defines then a partial order of events. A *workflow run* is a sequence of event instances:

$$(e_1, t_1) \dots (e_n, t_n)$$

where (e_1, t_1) is the first event instance of the run. A run can be either a *full run* in this case (e_n, t_n) represents the end of the workflow or a *partial run* that is not full.

The monitoring approach focuses on the time aspect of the workflow by taking into account that some activities may have a duration that can be pertinent to refine the diagnosis of the faults occurring during a workflow run. The basic idea [D4.2, 2006] of the method is to diagnose the occurrence of a fault by analysing the flow of observations (starts and ends of activities of the workflow) and matching this flow with a set of available *chronicles*. A chronicle is a partial order of events with time constraints and is associated with the occurrence of a fault. Once a chronicle is recognized, the diagnosis approach then automatically returns the associated fault as a diagnostic candidate that explains the observable part of the workflow. The set of chronicles used to perform the diagnosis is designed by expertise and with help of structural and temporal conformance analysis ([Eder et al., 2007]).

In this framework, faults are associated to activities. For example, in Figure 1, two faults are represented (illustrated by diamonds on activities). The first one is located in the activity *receiveOrder* and represents an order acquisition error. The second one is an unexpected delay (Timeout) in the *checkAvail&reserve* (check availability and reserve) activity (that requires an interaction with remote supplier services).

2.2 The chronicle-based diagnosis approach

The basic formal concepts of the chronicle-based approach applied to the monitoring of web services have been defined in WS-DIAMOND project [D4.2, 2006] [Cordier et al., 2007]. We recall the principle of this diagnostic algorithm in this section.

As said previously the monitoring approach is an event-based approach. An event is extended by the notion of *type*. The type of an event actually defines *what* is observed in the workflow. Depending on the available sensors and the reasoning accuracy that must be reached by the diagnosis algorithms, an event type in a workflow may be the name of an activity *act*: in this case, the diagnoser is able to observe that an activity of name *act* is actually running (e.g. *receiveOrder*). The type can also be the name of an activity augmented with the fact that the activity is starting (namely act^-) or ending (namely act^+): in this case the diagnoser is able to observe the event "*an activity act is starting*" and distinguish it from the event "*an activity act is ending*" (e.g. $receiveOrder^-$, $receiveOrder^+$). It may be also the case that some of the parameters of a given activity are observable. In this case, an event type may have the following form $act(?var_1 = val_1, \dots, ?var_n = val_n)$ that represents the event: "*The activity act is running with the parameters* $?var_1 = val_1$ *,...,* $?var_n = val_n$" (e.g. $receiveOrder(?items = "potatoes", ?custInfo = "Client23416")$). Finally, the type of event can also be the combination of previous types.

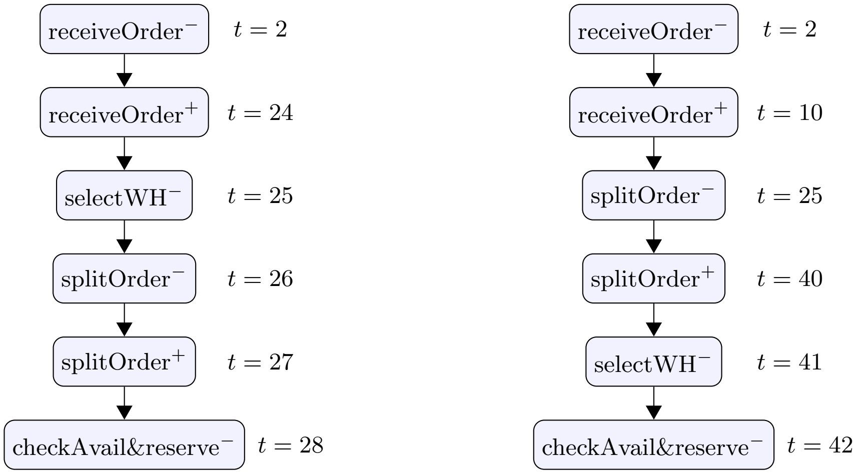


Figure 2: Partial normal and faulty runs of the workflow presented in Figure 1.

Figure 2 represents two partial runs of the workflow presented in Figure 1 where the considered type of events are the starting events and the ending events of each activity of the workflow. The first run is a normal run with no fault. Each event has an occurrence time that satisfies the time constraints defined in the workflow. The second run is a faulty run. An order acquisition error occurs in the *receiveOrder* activity that provokes an unexpected delay in the *splitOrder* activity (the duration of the activity *splitOrder* in this run is 15 whereas the normal duration for this activity is not greater than 4 (see Figure 1)).

In the following and without loss of generality, we will consider that the set of possible event types is available and is denoted $\mathcal{E}$. If the type of events simply represents the name of the activity, then $\mathcal{E}$ is obviously a finite set of types as the number of activity names in a given workflow is finite. If the types of events have the other forms then $\mathcal{E}$ may be an infinite enumerable set of types as the number of values associated to each variable may be infinite and enumerable.

An event is then stamped with the date of its occurrence. An *event* is a pair (e, t) where $e \in \mathcal{E}$ is an event type and t is the occurrence date of the event $t \in \mathbb{R}$. Instead of observing an event like "*The activity act has started*", here the considered event is "*The activity act started at time t*". The occurrence dates of the events are then of main importance for the chronicle definitions.

A chronicle model c is a pair $(\mathcal{S}, \mathcal{T})$ where $\mathcal{S}$ is a set of events and $\mathcal{T}$ a set of constraints between their occurrence dates. Figure 3 presents two temporal graphs, each of them represents a chronicle model. The first one is composed of four events. Time constraints between events are represented by time intervals.

An interval $]0, +\infty[$ is the way to represent an order between two events without any other time constraints. The second one is composed of two events.

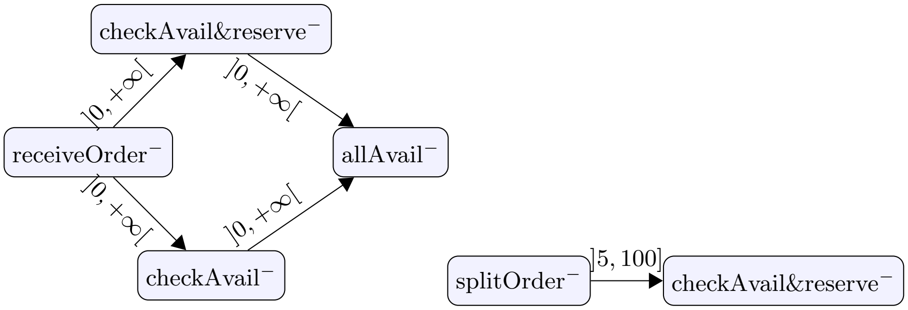


Figure 3: Two chronicles derived from the workflow (Figure 1).

Each chronicle is described in a specific formalism ([Cordier et al., 2007]). The descriptions for the previous chronicles is given below:

$\mathcal{S}_1$ = {(receiveOrder,?t1),
(checkAvail&Reserve$^-$,?t2),
(checkAvail$^-$,?t3),
(allAvail$^-$,?t4),
}

$\mathcal{T}_1 = \{?t1 <?t2, ?t1 <?t3, ?t2 <?t4, ?t3 <?t4\}$

and

$\mathcal{S}_2$ = {(splitOrder$^-$,?t1),
(checkAvail&Reserve$^-$,?t2), }

$\mathcal{T}_2 = \{5 <?t2 - ?t1 \leq 100\}$

3 Diagnosability of discrete event systems

3.1 Diagnosability Analysis: background

Diagnosability covers a set of properties that have been studied for many years in different fields [D5.1, 2007]. For continuous systems, diagnosability is stated in terms of detectability and isolability [Chen and Patton, 1994], [Basseville, 2001], [Travé-Massuyès et al., 2001], [Staroswiecki and Comtet-Varga, 1999]. In the discrete event systems (DES) context, the first definitions have been given in [Sampath et al., 1995]. Then, several extensions have been proposed for intermittent faults ([Contant et al., 2004]) and for more generic patterns [Pencolé, 2004], [Jéron et al., 2006]. Generally, the approaches for continuous systems adopt a state-based diagnosis (SBD) point of view in the sense that diagnosis is performed on a snapshot of observables, i.e. one observation at a given time point. DES approaches perform generally event-based diagnosis (EBD) and achieves state tracking, which means dynamic diagnosis reasoning achieved across time. Moreover, a unified diagnosability definition for state-based diagnosis and event-based diagnosis approaches have been proposed in [Cordier et al., 2006].

As presented previously, the monitoring approach using chronicles exploits the delays between two event instances for a better discrimination between diagnosis candidates. To study the diagnosis capabilities of such an approach, the classical event-based diagnosability approach is not sufficient as the delays between two event instances are not taken into account. In this subsection, an extension of the classical event-based framework for diagnosability checking is proposed. This extension introduces the notion of time and delay and is described in a way that is similar to the classical event-based diagnosability problem.

3.2 Diagnosability Analysis of discrete timed-event systems

Let us start now to define the diagnosability problem of activity workflows. From Section 2, it follows that a workflow can be interpreted as a language of words. Each word r is either a partial or a full run of the workflow. Let us denote by $\Sigma = \mathcal{E} \times \mathbb{R}$ the set of possible event instances of the workflow then a run r belongs to the Kleene closure of Σ (denoted $\Sigma^\star$):

$$r \in \Sigma^\star$$

The *workflow language* L_w is the prefix-closed language of all the possible runs of the workflow:

$$L_w \subseteq \Sigma^\star.$$

The workflow language explicitly characterizes the set of all possible runs of a given workflow. A run can be *normal* i.e. it results from the specification of the normal workflow (it can be a $BPEL$ Business Process Execution Language for Web Services specification for instance) or *abnormal* i.e. it results from the occurrence of a fault in the workflow. Generally, only a part of L_w is known (the normal part and some faulty parts). Depending on the available sensors, only a subset of activities in the workflow runs are observable. For a given run r, its observable part r_{OBS} can be defined by the sequence of observable events: $r_{OBS} = P_{OBS}(r)$. P_{OBS} is the classical projection operator that removes from any run r the unobservable events and keeps the observable one in r_{OBS}. With help of this operator, the observations OBS of the workflow are then defined as a language of observable events.

An *observation* σ of the workflow is a sequence of observable events resulting from the projection on the observable events of a sequence in the workflow, it is thus a word of the language:

$$OBS = P_{OBS}(L_w).$$

Figure 4 presents the observations of the runs presented in Figure 2. In this figure it is supposed that only the start of each activity of the workflow is observable; the events related to the end of the activities are not observable.

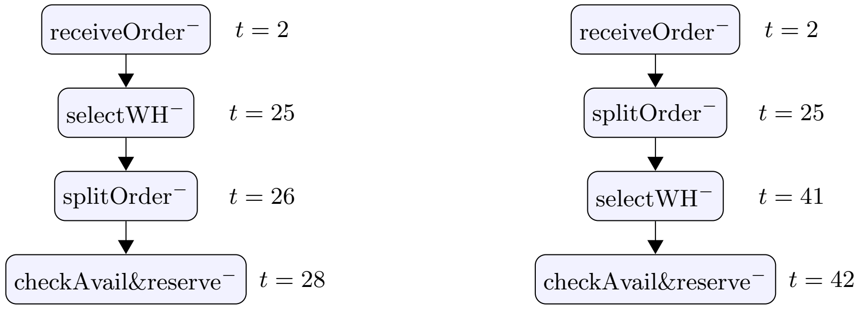


Figure 4: Observable part of the partial runs of Figure 2.

From this, follows the notion of signatures in this framework. Let $\mathcal{F} = \{f_1, \ldots, f_n\}$ be the set of basic faults that could occur in the workflow and that requires a diagnosis, let L_{f_i} be the sub-language of L_w that contains all the runs in which f_i holds: the fault signature $Sig(f_i)$ is the observable language:

$$Sig(f_i) = P_{OBS}(L_{f_i}) \subseteq OBS.$$

The second observable run of Figure 4 is typically part of the signature $Sig(orderAcquisitionError)$ (see Figure 1).

By extension, if we denote by $f_0 = norm$ the absence of any fault of $\mathcal{F}$ (i.e. $f_0 \equiv norm \equiv \neg f_1 \wedge \ldots \wedge \neg f_n$), and L_{f_0} the sub-language of L_w that contains all the runs in which none of the fault $f_i, i > 0$ holds, it follows the normal signature is the observable language:

$$Sig(f_0) = P_{OBS}(L_{f_0}) \subseteq OBS.$$

The first observable run of Figure 4 is typically part of the normal signature $Sig(f_0)$.

The characteristic fault signature $Sig_c(f_i)$ is the observable language:

$$Sig_c(f_i) = P_{OBS}(L_{f_i}) \setminus (\bigcup_{j \neq i} P_{OBS}(L_{f_j}))$$

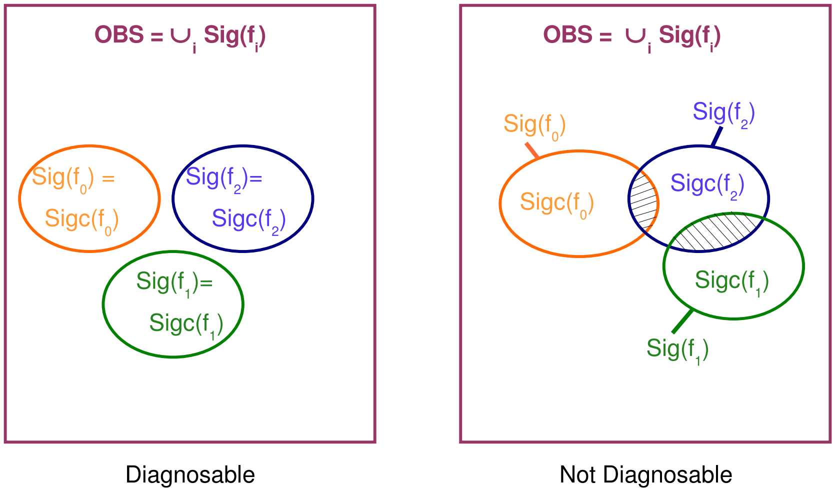


Figure 5: Diagnosable/Not diagnosable

If the observable part of a run belongs to $Sig_c(f_i)$, then the occurrence of f_i (or the absence of fault represented by f_0) during the run is certain, there is no ambiguity. Finally under the *single fault assumption*, the diagnosability definition can be refined in terms of languages as follows:

$f_0, f_1, \ldots, f_n$ are *diagnosable* in the workflow W iff the set

$$\{Sig(f_0), Sig(f_1), \ldots, Sig(f_n)\}$$

is a partition of OBS, in other words:

$$\forall i \in \{0, \ldots, n\}, Sig(f_i) = Sig_c(f_i).$$

Figure 5 illustrates two different configurations. On the left hand, the faults are diagnosable. Any possible observable run of a workflow characterizes either f_0 or f_1 or f_2. The three signatures are disjoint so they are characteristic. On the right hand of Figure 5, the faults are not diagnosable. $Sig(f_2)$ overlaps a part of $Sig(f_0)$ so there exists at least one observable run whose underlying run is affected by the fault f_2 or is not affected by any fault (f_0 holds). There are also some observable runs that make the presence of f_1 and f_2 ambiguous since $Sig(f_1) \cap Sig(f_2) \neq \emptyset$. Finally, as $Sig(f_1) \cap Sig(f_0) = \emptyset$, f_1 is said to be *detectable*: if f_1 is present then the observable run is certainly abnormal, hence a fault detection (but not a fault identification, as both f_1 and f_2 can still explain the run). f_2 may not be detectable as it can explain observable runs that are also explained by the absence of the f_2.

4 Diagnosability and chronicles

A chronicle is a set of observable events with some time constraints. A *good* chronicle is used to characterize a situation relying on its observable consequences. In a chronicle-based diagnosis approach, each abnormal situation (i.e. a fault f_i in our context) is usually associated to a set of chronicles, each chronicle recognizing a sub-part of the corresponding fault signature $Sig(f_i)$. The main positive point with these approaches is their high efficiency due to the symptom-to-fault knowledge they rely on. The counterpart is usually the difficulty of acquiring and updating the chronicle base.

Chronicle recognition proceeds as follows. Given a flow of observed event instances, a chronicle recognition system is in charge of checking on the fly whether the flow matched with some of the chronicles in the base. A chronicle model is said to be recognized if the flow of event instances contains an instance of every event in the chronicle model and these instances satisfy the time constraints defined in the chronicle model. More formally:

A chronicle model $c = (\mathcal{S}, \mathcal{T})$ is *recognized* in an observable sequence σ of event instances iff:

- there exists, for all event $(e_i, t_i) \in \mathcal{S} = \{(e_1, t_1), \ldots, (e_n, t_n)\}$, an event instance (e_i, t'_i) in σ and
- the constraint system $\{t_1 = t'_1, \ldots, t_n = t'_n\} \cup \mathcal{T}$ is satisfiable.

In the following, the predicate $recognition(c, \sigma)$ will denote the fact that the chronicle model c is recognized in the observable sequence σ. The chronicle on the right hand in Figure 3 is recognized by the second observable run of Figure 4 but not by the first one.

By construction, a chronicle model may be recognized in several observable runs σ of a workflow, these sequences define the *recognition language* of the chronicle model in the workflow.

Let c be a chronicle model and OBS the observations of the workflow W, the *recognition language* $L(c, OBS)$ is the set of observations $\sigma \in OBS$ such that $recognition(c, \sigma)$.

A chronicle model is designed in order to be associated to the occurrence of a fault. In the following, a chronicle model associated to a fault f is denoted $c(f)$. By construction, once a chronicle $c(f)$ is recognized, the monitoring approach provides the occurrence of the fault f as a possible diagnosis candidate that explains the current observations. It follows that a chronicle model $c(f)$ associated to the fault f is necessarily such that: $L(c(f), OBS) \subseteq Sig(f)$. The left-hand chronicle in Figure 3 is a chronicle that represents compatible time constraints with the normal workflow presented in Figure 1. In every normal run, this chronicle model is recognized, in this case, the recognition language of the chronicle includes $Sig(f_0)$. The right-hand chronicle in Figure 3 represents an unexpected delay in the *splitOrder* activity. As said previously, this is a possible consequence of a *dataAcquisitionError* fault in the activity *receiveOrder*. The recognition language of the chronicle is included in $Sig(\mathit{dataAcquisitionError})$.

Depending on the definition of a chronicle $c(f)$, it is possible that its recognition language is included in the characteristic signature of the fault f.

A chronicle $c(f)$ *characterizes* f iff:

$$L(c, OBS) \subseteq Sig_c(f) \subseteq Sig(f).$$

If $c(f)$ characterizes the occurrence of f, it means that the occurrence of the fault f during the observed run is certain.

Let $L \subseteq OBS$ be a set of observations, a chronicle model c *covers* L iff $L \subseteq L(c, OBS)$. A set of chronicle models $c_1, \ldots, c_n$ covers L iff $L \subseteq \bigcup_{i=1}^{n} L(c_i, OBS)$.

Let f be a fault, a set of chronicle models $c_1, \ldots, c_n$ covers f iff it covers $Sig(f)$.

Once we know that a set of chronicles covers a fault f, we know that, if the fault f occurs, one of the chronicle $c_1, \ldots, c_n$ will be at least recognized. Given these definitions, we can now present the relationship between fault diagnosability and chronicle modeling as a sufficient condition for diagnosability.

Property 1 *Let f be a fault in the workflow W, if there exists a set of chronicles $C(f) = \{c_1(f), \ldots, c_n(f)\}$ such that:*

1. *$C(f)$ covers f, and*
2. *$\forall j \in \{1, \ldots, n\}, c_j(f)$characterizes f,*

then f is diagnosable in the workflow W.

Proof: Firstly, condition 1 implies that $Sig(f) \subseteq \bigcup_{i=1}^{n} L(c_i(f), OBS)$. Secondly, condition 2 implies that $\bigcup_{i=1}^{n} L(c_i(f), OBS) \subseteq Sig_c(f)$. It follows that $Sig(f) = Sig_c(f)$, hence the result.

From Property 1, it follows that, under the *single fault assumption*, checking the diagnosability of a fault f relying only on a set of chronicles requires checking whether two chronicles $c(f)$ and $c(f')$ are *exclusive* or not (see more details in the next subsection). If $c(f)$ and $c(f')$ are not exclusive then they are not characteristic as $Sig(f) \cap Sig(f') \neq \emptyset$. The next subsections present the chronicle exclusiveness test as a mean to study the diagnosability of the workflow.

5 Exclusiveness test of chronicles

5.1 Problem statement

Two chronicles are exclusive if they cannot be recognized with the same flow of event instances. In the following and for the sake of simpler notations, the recognition language $L(c(f), OBS)$ of a chronicle model $c(f)$ is simply denoted by $\mathcal{C}$.

From a diagnosability point of view, the proposed exclusiveness analysis can be performed relying on two kinds of inputs:

- Check for the non exclusiveness of chronicles $\mathcal{C}_1 = c_1(f_1)$ and $\mathcal{C}_2 = c_2(f_2)$: that is $\mathcal{C}_1 \cap \mathcal{C}_2 \neq \emptyset$? As $\mathcal{C}_1 \subseteq Sig(f_1)$ and $\mathcal{C}_2 \subseteq Sig(f_2)$, if $\mathcal{C}_1 \cap \mathcal{C}_2 \neq \emptyset$ then f_1 and f_2 are not diagnosable.
- Check for the non exclusiveness between a chronicle $\mathcal{C} = c(f)$ and a non faulty model of the monitored system i.e. a non faulty model W_N of the workflow W: that is $\mathcal{C} \cap P_{OBS}(L_{W_N}) \neq \emptyset$? As $\mathcal{C} \subseteq Sig(f), f \neq f_0$, if $\mathcal{C} \cap P_{OBS}(L_{W_N}) \neq \emptyset$ then f is not diagnosable and more precisely f is not detectable.

Informally, the exclusiveness of chronicle models is illustrated as follows: let us consider the two simple chronicle descriptions:

1. c_1: event $\clubsuit$ followed by event $\diamondsuit$
2. c_2: event $\diamondsuit$ followed by event $\spadesuit$.

If the observed event flow is $\clubsuit$ $\clubsuit$ $\diamondsuit$ $\diamondsuit$, four instances of c_1 can be identified but if the event flow is extended by the event $\spadesuit$ two instances of c_2 will also be identified. This is illustrated on Figure 6.

As the events of the flow are ordered and time-stamped each instance of a chronicle can be associated to a *Time Support*. The Time Support of a chronicle instance is the necessary time interval leading to the chronicle recognition. The exclusiveness analysis of the chronicles can then be interpreted as an analysis of the time supports. Given a base of chronicles the time supports of the chronicles can be determined. Then for one observable run of the workflow it is possible to determine which chronicle will be recognized. Thus, it is possible at design stage to conclude the exclusiveness of the chronicles for some particular observable runs and the non exclusiveness for other observable runs. Moreover, in the case of non exclusiveness it is possible to determine for one specific run of the workflow which chronicle will be first recognized. This may be of interest to point out useless chronicles when several chronicles are supposed to identify the same fault. If the

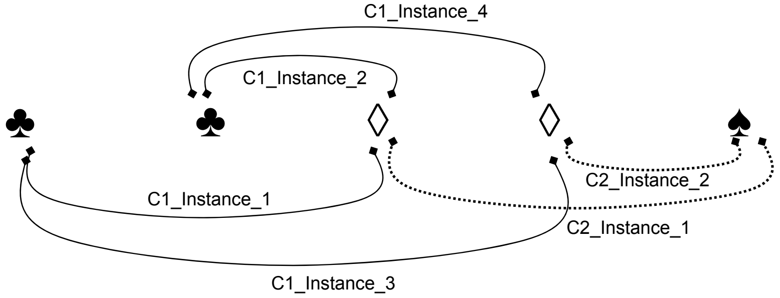


Figure 6: Instances of chronicles

set of all possible runs of the workflow i.e. the workflow language is known more general conclusions can be done.

Obviously, the difficulty is to determine for a chronicle all the possible Time Supports. Indeed, in a general way, a chronicle model describes not only logical relationships between the events (e.g. conjunction and disjunction) but also time relationships between events. These relationships are expressed through the constraints of the chronicle description. The time constraints (precedence constraint, interval constraint ...) are expressed with relative dates. That means that the exclusiveness analysis must deal with an important number of chronicle instances. Therefore, a complete enumeration of the possible instances of each chronicle is not realistic as the set of possible instances i.e. the state space is infinite. The challenge of the diagnosability analysis which is performed at design stage, is then to deal with the combinatorial explosion of the chronicle instances.

The approach proposed in this work is not to deal with all these instances i.e. with all the temporal behaviors associated to a chronicle but rather to consider classes of temporal schema. Our proposal is to generate a state space abstraction to realize the diagnosability analysis. More precisely we propose to work with a time abstraction of the different instances of a chronicle and to perform the exclusiveness analysis on this time abstraction. The aim of this time abstraction is to obtain a finite representation of the set of instances which is generally infinite in the case of time constraints.

5.2 Time abstraction

Several results exist in the field of state space abstraction in a general way without focusing on the chronicle notion. Results exist both in the automata or

Petri Nets (PN) world, as these two models are widespread. In the first case, Timed Automata are mainly used [Alur and Dill, 1994] and in the second case many Petri net based models with time extensions have been proposed.

Among these models, Time Petri Nets (TPN) [Merlin and Farber, 1976] is prominent as several effective analysis methods have been proposed (see for instance [Berthomieu and Menasche, 1983], [Berthomieu and Diaz, 1991]). Time Petri nets extend Petri nets with temporal intervals associated to transitions. Firing delay ranges are then associated to transitions.

A Time Petri Net is a tuple

$$< P, T, Pre, Post, m_0, I_s >$$

in which $< P, T, Pre, Post, m_0 >$ is a Petri net with P the set of places, T the set of transitions, m_0 the initial marking and Pre, $Post$ the forward and backward incidence functions. $I_s : T \rightarrow I^+$ is a function called the Static Interval function that associates a time interval to each transition of the net. I^+ is the set of non empty real intervals with non-negative rational end-points. The left-end-point and the right-end-point of the interval associated to a transition t are respectively the static earliest firing date of t, and the the static latest firing date of t.

A state of TPN [Berthomieu and Diaz, 1991] is a couple $s = (m, I)$ in which m is a marking and $I : T \rightarrow I^+$ is a function that associates a time interval to each transition enabled by m. The initial state is $s_0 = (m_{0,I_0})$ with I_0 the restriction of I_s to the transitions enabled by m_0. Every enabled transition must be fired in the associated interval. This interval is relative to the enabling date of the transition and depends on the date of the last transition firing. Firing a transition t after a delay θ from state $s = (m, I)$ is possible iff:

$$m \geq Pre(t) \wedge \theta \in I(t)$$

$$\wedge(\forall t' \neq t), (m \geq Pre(t') \Rightarrow \theta \leq sup(I(t'))).$$

The state $s' = (m', I')$ reached by firing the transition t is given by:

- the new marking m' classically defined by $m' = m - Pre(t) + Post(t)$
- for every transition t' enabled by m':
  1. $I'(t') = I(t') - \theta$ if $t' \neq t$ and $m - Pre(t) \geq Pre(t')$: the firing of t' is not affected by the firing of t
  2. $I'(tr) = I_s(tr)$ otherwise.

For TPN several techniques are available to generate state space abstraction in order to obtain a finite representation of the behavior. These state space abstractions are based on this notion of state of TPN. Let us consider that a

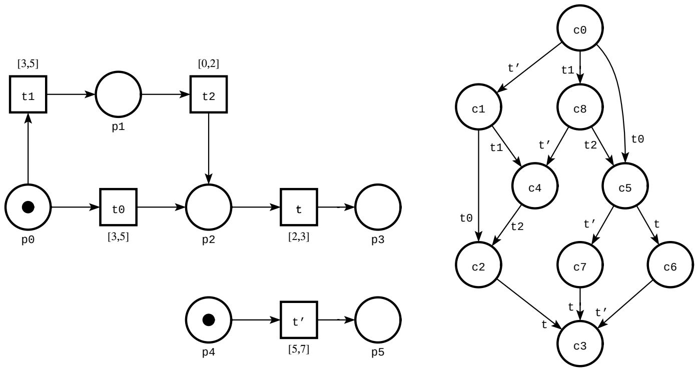


Figure 7: Time Petri Net and State Class graph from [Berthomieu et al., 2004]

$c0$ $p0,\ p4$ $5 \leq t' \leq 7$ $3 \leq t0 \leq 5$ $3 \leq t1 \leq 5$	$c1$ $p0,\ p5$ $0 \leq t0 \leq 0$ $0 \leq t1 \leq 0$	$c2$ $p2,\ p5$ $2 \leq t \leq 3$	$c3$ $p3,\ p5$	$c4$ $p1,\ p5$ $0 \leq t2 \leq 2$
$c5$ $p2,\ p4$ $2 \leq t \leq 3$ $0 \leq t' \leq 4$	$c6$ $p3\ , p4$ $0 \leq t' \leq 2$	$c7$ $p2\ , p5$ $0 \leq t \leq 3$	$c8$ $p1\ , p4$ $0 \leq t' \leq 4$ $0 \leq t2 \leq 2$	

Figure 8: Classes: marking and firable domain from [Berthomieu et al., 2004]

transition t is enabled for the last time in the state $s = (m, I)$ at the date α. t cannot be fired from the state s before the date (α + earliest firing date of t) and must be fired before the date (α + latest firing date of t). The firing of t leads to a new state of the TPN. As transitions in a TPN can be fired at any date in their time interval each state in a TPN may have an infinite number of successors. The state space abstraction is obtained by the grouping of the TPN states in terms of *State Classes*. Several kinds of grouping have been defined according to the properties of the state space they preserve and according to the considered type of TPN (with priorities or not).

Figure 7 and Figure 8 directly issued from [Berthomieu et al., 2004] give an example of a TPN and its associated State Class Graph. The State Class Graph allows to abstract the time from the behavior of a Time Petri Nets, therefore the transitions are not labeled with time information.

5.3 Computation of the proposed exclusiveness test

The computation of the exclusiveness test, that is proposed in this paper, is then based on several steps:

- **Generation of the chronicles descriptions and of the workflow**. This step is of main importance and the acquisition and the update of the set of chronicles is not trivial. Currently several researches address this problem using notably learning techniques [Mayer, 1998] [Quiniou et al., 2001]. In this article the set of chronicles is supposed to be available and described according to the formalism presented in Section 2.2. Moreover, this set is composed of chronicles associated to faults that can occur during the workflow execution but it also includes chronicles that describe the normal behavior of the workflow, that means the normal behavior of the Web Service that is described by the workflow.

- **Chronicle modeling with TPN**. This step aims at modeling the behaviors associated to chronicle descriptions by Time Petri Nets. The events of a chronicle are associated to the transitions of the TPN, delays are assigned to the transitions if necessary. Precedence constraints are implicitly modeled through the chaining place/transition/place. As said previously an interval $[0, \infty[$ is the way to represent an unconstrained occurrence date of an event. The initial marking is a waiting state and final places can be defined as *the chronicle is recognized.* The other places represent intermediate stages during the chronicle recognition and can be interpreted as *the chronicle is partially recognized.*

  Figure 9 shows the model of two chronicles c_1 and c_2 by Time Petri nets. The description of c_1 and c_2 is as follows: $(\mathcal{S}_1 = \{\spadesuit, \clubsuit, \diamondsuit\}$ and $\mathcal{T}_1 = \{(\diamondsuit - \clubsuit) \leq 4, \spadesuit < \clubsuit < \diamondsuit\})$ and $(\mathcal{S}_2 = \{\clubsuit, \diamondsuit\}$ and $\mathcal{T}_2 = \{(\diamondsuit - \clubsuit) \leq 7, \clubsuit < \diamondsuit\})$.

  For some chronicles a final place defined as *the chronicle is dead* can be added. This place is marked when the chronicle cannot be recognized whatever the next events are. This is the case for instance when the chronicle description explicitly forbids an event. The description of the chronicle is then done using a *no-event* clause [D4.2, 2006]. For example, let us suppose that a new constraint is added to the chronicle c_2 specifying that no event $\diamondsuit$ must occur in addition of the event $\diamondsuit$ expected after the event $\clubsuit$ in the interval $[0, 7]$. The corresponding Time Petri Net model is shown in Figure 10. In this case, the model is obtained by Time Petri Nets with Priorities $PrTPN$ [Berthomieu et al., 2006]. $PrTPN$ is an extension of TPN in which a priority relation on transitions is defined. Indeed, in Time Petri Nets the fact that the time elapse can only increase the number of firable transitions. Note that this is one important difference between Timed Automata

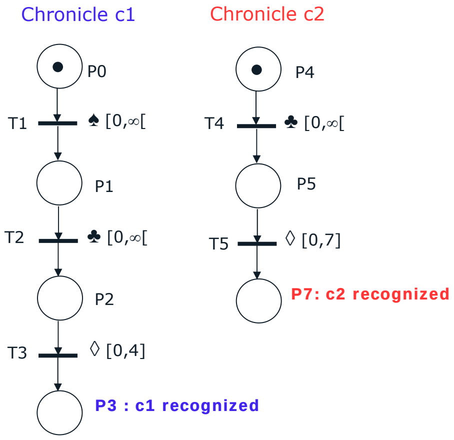


Figure 9: Time Petri Net of chronicles descriptions

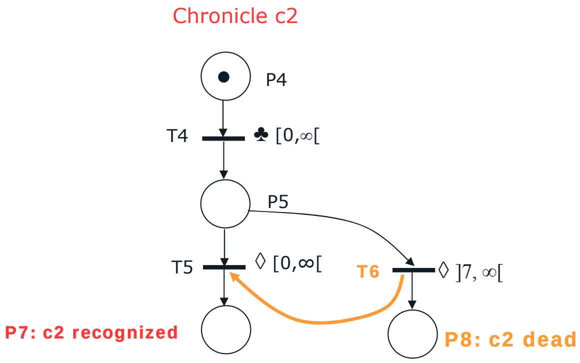


Figure 10: Time Petri Nets with Priorities for no-event modeling

and Time Petri Nets. That is the firing of a transition cannot be forbidden by the time elapse. Then priorities are used to complete firing conditions of a transition. In $PrTPN$ a transition t may fire from a state $s = (m, I)$ if t is enabled by the marking m, firable instantly and if no transition with higher priority satisfies these conditions. Let us consider the example shown in Figure 10. Transition $T6$ is silent and firable at any time greater than 7. Transition $T5$ may fire at any time less than or equal to 7, but no later since transition $T6$ has priority over $T5$, which is modeled by a directed arrow from $T6$ to $T5$.

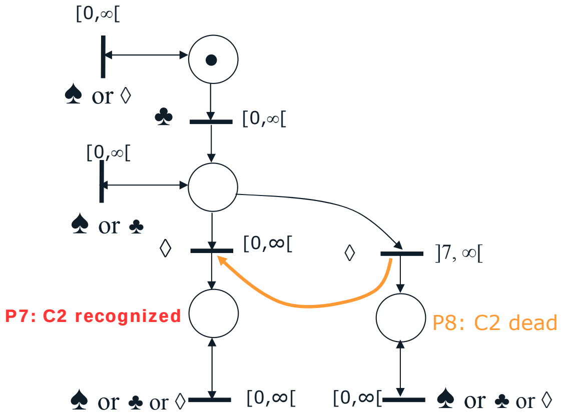


Figure 11: Generator of the chronicle language

For a given chronicle, it is important to say that the aim of this modeling step is to model the chronicle description only and not the recognition language of the chronicle [Bertrand et al., 2007] [Boufaied et al., 2002] [Dousson, 1996]. To illustrate this, let us come back to the example of the chronicle c_2: event ♣ followed by ♢ and *no-event* ♢ in $[0, 7]$ after the event ♣. Let us suppose the set of possible event types E={♣, ♢, ♠}. Figure 11 gives the Time Petri Net with Priorities (TPN) of what is classically called *a generator* G. The *generator* is the Petri net model from which the chronicle language can be obtained. Indeed, the set of all the recognized instances corresponds to the Petri Net Language $L_m(G)$ where $L_m(G)$ is the marked behavior (or terminal language) defined as the set of words generated by firing sequences that reach to a final marking [Jantzen, 1987]. All the recognized instances of a chronicle are then given by all the marked behaviors of the Petri Net and represented by the reachability tree (if it exists) associated to the Petri net. Currently, these modeling issues are addressed by several works. Chronicle recognition techniques rely on multiple types of modeling tools such as colored Petri nets, or Petri nets with counters [Bertrand et al., 2007].

Figure 12 finally gives a last example of TPN corresponding to the chronicles of Figure 3.

By modeling the chronicle description, the purpose is to only model the relevant abstraction of the recognised language that is useful for the diagnosability analysis. Indeed, the chronicle description gives the most discriminative part of the words belonging to the recognition language that are considered

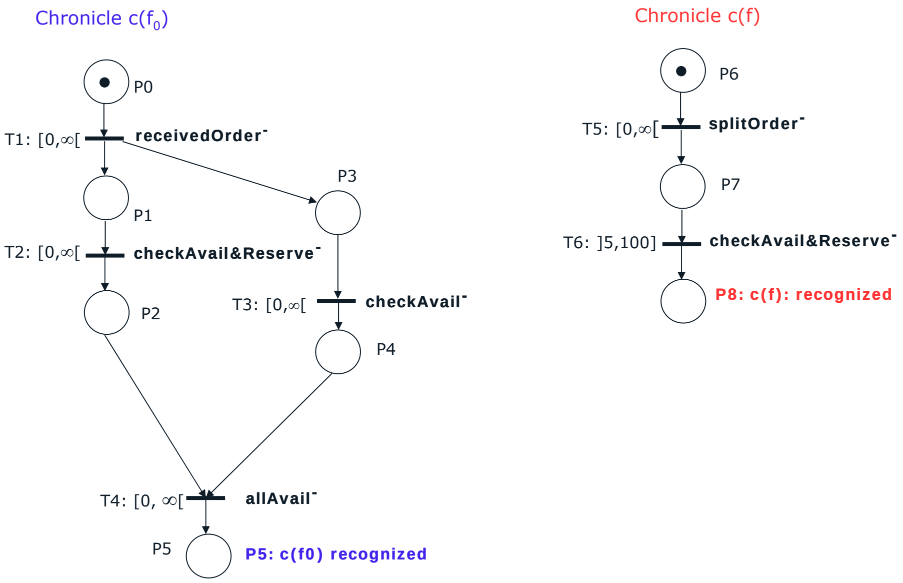


Figure 12: Time Petri Net model for the chronicles of Figure 3

as a fault manifestation. The modeling of these relevant parts is sufficient for exclusiveness analysis between chronicles. If the set of available chronicles covers the set of faults, then if a fault occurs at least one of these chronicles must be recognized.

- **Time abstraction generation**: from the Time Petri Net models, it is possible to generate the time abstraction using *State Class* techniques. The result is given by a *State Class Graph*. In order to point out the intersection of chronicles, the solution is to work on the synchronized product of the TPN models to generate the time abstraction. By considering the synchronized product, the size of the *State Class Graph (SCG)* is limited. The product construction for labeled Petri nets is quite similar to the product of automata [Hack, 1976][Berthomieu et al., 2006]. For Petri nets, the product is compositional i.e. the behavior of the PN product (noted $\aleph 1 \| \aleph 2$) is the product of those noted $\aleph 1$ and $\aleph 2$. For Time Petri nets this property holds under the condition that all synchronized transitions have interval $[0, \infty[$. This aspect must be taken into account during the modeling step of the chronicles. In the case, the chronicle descriptions only include precedence constraints ($[0, \infty[$) between events, the modeling by Time Petri Nets of the chronicle descriptions and the generation of the synchronized product can be easily performed as previously presented. In other cases, to ensure the com-

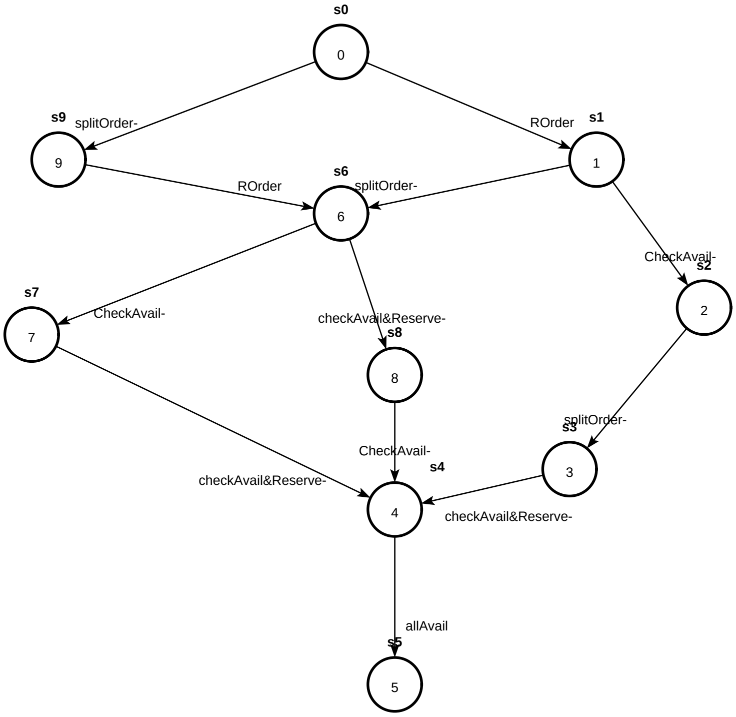
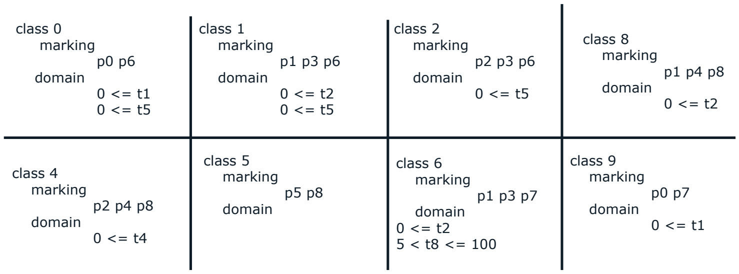


Figure 13: State Class Graph from synchronized TPN

positionnality some additional model transformations must be performed. It is possible for this to consider Time Petri Nets with priorities ($PrTPN$). The use of priorities allows to put the time constraints in the local parts of the nets and to have all the synchronized transitions with intervals $[0, \infty[$. The Product of $PrTPN$ is defined in [Berthomieu et al., 2006]. Another solution is to generate the product without taking into account time intervals and to put the time constraints on the synchronized transitions after the product construction. This part of the work presented in this article is currently under research.

Figure 13 gives the State Class Graph (10 classes and 12 transitions) generated from the synchronized part of the TPN model issued from Figure 12.

- **Checking non exclusiveness:** the test may involve two chronicles associated to faults or one chronicle associated to a fault and another one associated to nominal behaviors of the workflow. The idea of this step is to verify that there exists in the State Class Graph describing the synchro-



```
class 0
   marking
           p0 p6
  domain
           0 <= t1
           0 <= t5

class 1
   marking
           p1 p3 p6
  domain
           0 <= t2
           0 <= t5

class 2
   marking
           p2 p3 p6
  domain
           0 <= t5

class 8
   marking
           p1 p4 p8
  domain
           0 <= t2

class 4
   marking
           p2 p4 p8
  domain
           0 <= t4

class 5
   marking
           p5 p8
  domain

class 6
   marking
           p1 p3 p7
  domain
0 <= t2
5 < t8 <= 100

class 9
   marking
           p0 p7
  domain
           0 <= t1
```

Figure 14: Extract of the classes associated to the synchronized behavior

nized behavior at least one path leading both to the recognition of several chronicles of faults or leading to the recognition of the chronicles and of the workflow. This path represents the possibility to associate a same instance of events with more than one chronicle and corresponds to the fact that the two chronicles can be recognized by the same flow of events with the same Time Support. It may also exist independent transitions in the Petri Nets associated to the chronicles, thus it may also exist in the SCG some paths leading to the recognition of both chronicles but in a sequential way. The instances of events associated to each chronicle are simply ordered so that both chronicles are partially recognized during a specific time interval.

For the previous example, the path defined by the sequence of classes $C0$, $C1$, $C6$, $C8$, $C4$, and $C5$ corresponds to a synchronized behavior (see Figure 14). This path represents an intersection of the Time supports of $c(f)$ and $c(f_0)$: the two chronicles can have some instances that are recognized from a shared instance of events, so they are not exclusive. As f_0 stands for the absence of fault, it means that in this example, the fault f is actually not detectable (see Section 5.1 for details).

For example σ is a partial observed flow of instances corresponding to such a path: $\sigma = \{(receivedOrder^-, 3)$, $(splitOrder^-, 8)$, $(checkAvail\&reserve^-, 15)$, $(checkAvail^-, 25)$, $(allAvail^-, 30)\}$.

6 A diagnosability checker Tool

This section presents a diagnosability checker Tool that demonstrates what is presented in the previous sections in the context of the WS-DIAMOND project. The development of this checker is motivated by the need of an automatic tool to check the diagnosability of the workflows relying on monitoring approach for diagnosis [D4.2, 2006]. Moreover, this software partly relies on the use of

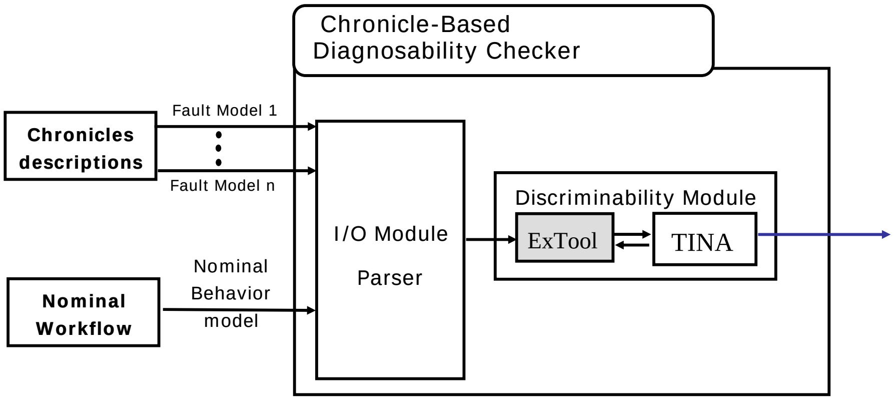


Figure 15: Diagnosability checker description

off-the-shelf techniques with their associated tools. Therefore, the developed tool is based on TINA (TIme Petri Net Analyzer, (http://www.laas.fr/tina)) [Berthomieu et al., 2004]. TINA not only proposes common graphical editing facilities but also construction of representation behavior of PN and TPN. Currently several investigations on TPN are grouped in the software TINA. Figure 15 illustrates the description of the diagnosability tool. The checker is mainly composed by an input/output parser and a discriminability module. The input/output parser allows to transform textual descriptions and workflow model to Time Petri Nets. The discriminability analysis module performs the test of exclusiveness. The Extool analyses also the State Class Graphs Generated by TINA to conclude on the exclusiveness of chronicles. Finally, TINA allows the graphical edition of the chronicles descriptions under the Time Petri Net formalism and generates the State Class Graph.

Seven tabs appear. The purpose of each onglet is to present one view of the demonstration.

– Workflow tab: this tab displays the considered workflow of Web Services (Figure 16). The benchmark chosen is a Web Service of a foodshop. The following description of the foodshop is directly issued from [Team, 2008]. When a customer places an order, the Shop service selects the Warehouse that is closest to the customers address, and that will thus take part in process execution. Ordered items are split into two categories: perishable

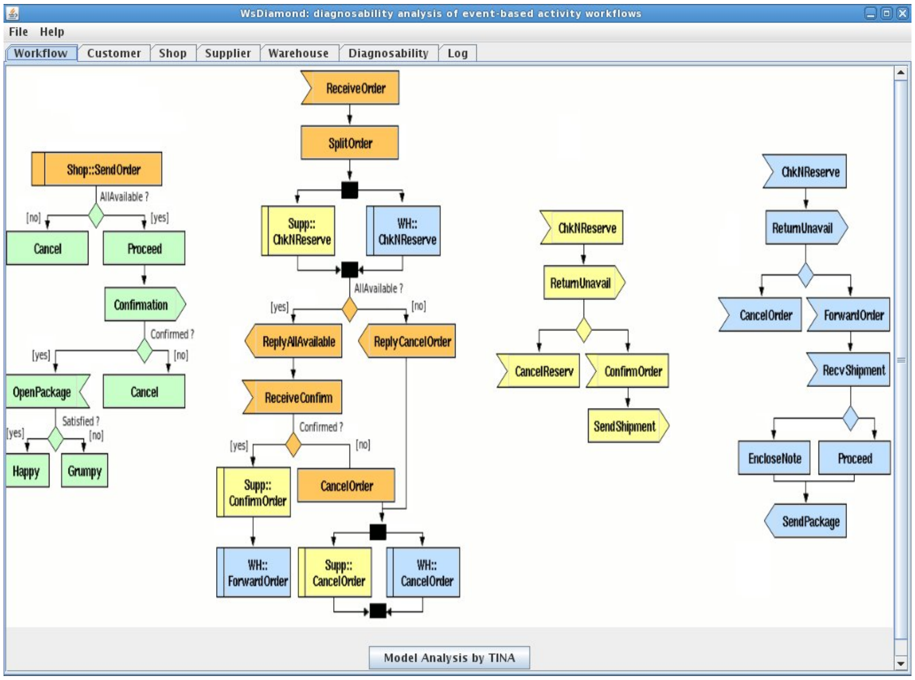
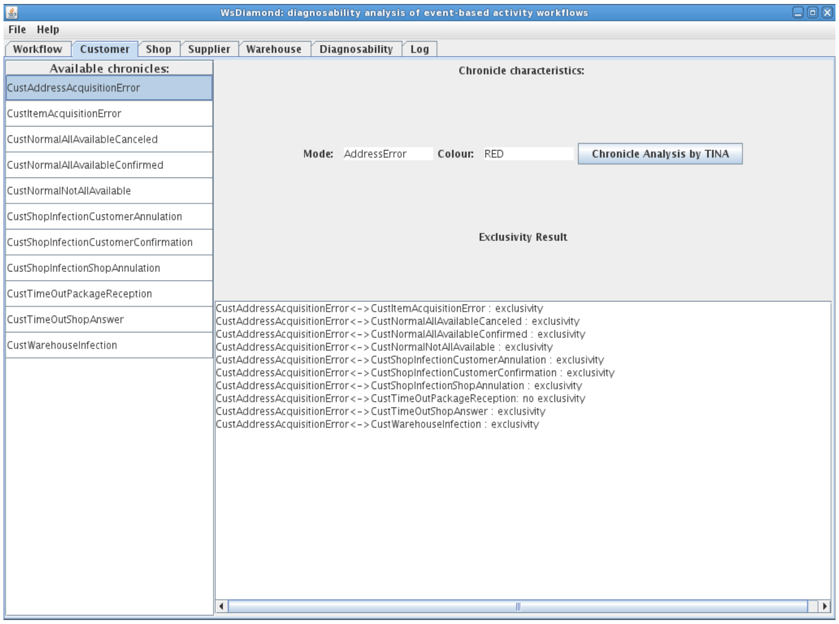


Figure 16: Workflow of the foodshop example of the WS-DIAMOND project

(they cannot be stocked, so the warehouse will have to order them directly) and nonperishable (the warehouse should have them in stock). Perishable items are handled directly by the Shop, exploiting the services of a Supplier, whereas nonperishable items are handled by the Warehouse; all of them are eventually collected by the Warehouse in order to prepare the parcel for the customer. The Shop checks whether the ordered items are available, either in the Warehouse or from the Supplier. If they are, they are temporarily reserved in order to avoid conflicts between orders. Once the Shop receives all the information on item availability, it can decide whether to give up on the order (to simplify, this happens whenever there is at least one unavailable item) or to proceed. In the former case, all item reservations are canceled and the business process ends. If the order proceeds, the Shop computes the total cost (items plus shipping) with the aid of the Warehouse, which provides the shipping costs, depending on its distance from the customer location and the size of the order. Then it sends the bill to the customer, who can decide whether to pay or not. If the customer does not pay, all item reservations are canceled and, again, the business process terminates. If the customer pays, then all item reservations are confirmed and all the Suppliers (in the cases of perishable and out-of-stock items) are asked to send their goods to the Warehouse. The Warehouse will then assemble a package and send it to the customer.

By clicking on Model Analysis by TINA, the Petri Net representing the nominal model of this workflow is loaded in TINA. TINA can then be used to perform several analyses on the nominal model (simulation, computation of the marking graph,..).


WsDiamond: diagnosability analysis of event-based activity workflows
File
Help
Workflow
Customer
Shop
Supplier
Warehouse
Diagnosability
Log
Chronicle Analysis by TINA

Available chronicles:

CustAddressAcquisitionError
CustItemAcquisitionError
CustNormalAllAvailableCanceled
CustNormalAllAvailableConfirmed
CustNormalNotAllAvailable
CustShopInfectionCustomerAnnulation
CustShopInfectionCustomerConfirmation
CustShopInfectionShopAnnulation
CustTimeOutPackageReception
CustTimeOutShopAnswer
CustWarehouseInfection

Chronicle characteristics:

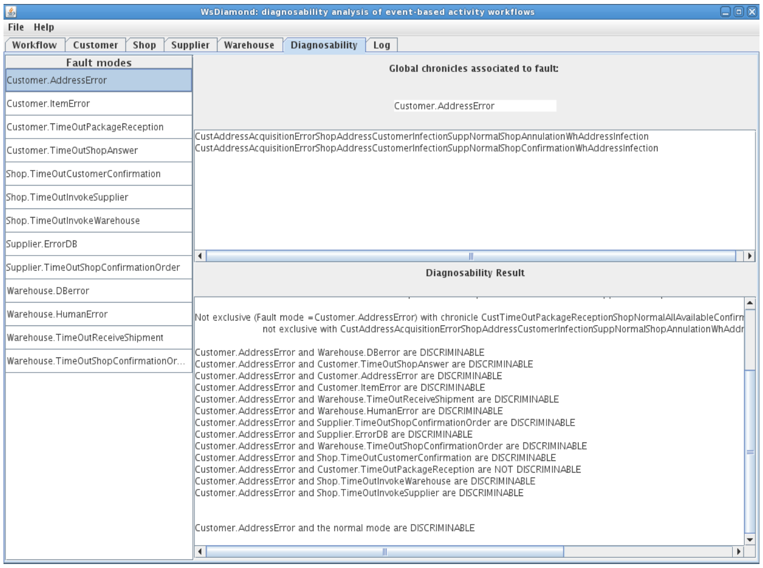
Mode: AddressError Colour: RED

Exclusivity Result

```
CustAddressAcquisitionError<->CustItemAcquisitionError : exclusivity
CustAddressAcquisitionError<->CustNormalAllAvailableCanceled : exclusivity
CustAddressAcquisitionError<->CustNormalAllAvailableConfirmed : exclusivity
CustAddressAcquisitionError<->CustNormalNotAllAvailable : exclusivity
CustAddressAcquisitionError<->CustShopInfectionCustomerAnnulation : exclusivity
CustAddressAcquisitionError<->CustShopInfectionCustomerConfirmation : exclusivity
CustAddressAcquisitionError<->CustShopInfectionShopAnnulation : exclusivity
CustAddressAcquisitionError<->CustTimeOutPackageReception: no exclusivity
CustAddressAcquisitionError<->CustTimeOutShopAnswer : exclusivity
CustAddressAcquisitionError<->CustWarehouseInfection : exclusivity
```

Figure 17: Chronicles for the Customer

- Service tab: the foodshop example contains four services: Customer, Shop, Supplier and Warehouse. To each service is associated a tab. This tab presents the chronicles that are available in the service. Figure 17 shows the exclusiveness tests of the chronicle 'CustAddressAcquisitionError' in the service 'Customer' with all the available chronicles of the service 'Customer'. In this example, 'CustAddressAcquisitionError' and 'CustTimeOutPackageReception' shares common runs in the service 'Customer'. So, there is a *local ambiguity* between the occurrence of the fault 'AddressError' and the fault 'TimeOutPackageReception' in the service customer. Note that, for


WsDiamond: diagnosability analysis of event-based activity workflows
File
Help
Workflow
Customer
Shop
Supplier
Warehouse
Diagnosability
Log
Fault modes
Customer.AddressError
Customer.ItemError
Customer.TimeOutPackageReception
Customer.TimeOutShopAnswer
Shop.TimeOutCustomerConfirmation
Shop.TimeOutInvokeSupplier
Shop.TimeOutInvokeWarehouse
Supplier.ErrorDB
Supplier.TimeOutShopConfirmationOrder
Warehouse.DBerror
Warehouse.HumanError
Warehouse.TimeOutReceiveShipment
Warehouse.TimeOutShopConfirmationOr...
Global chronicles associated to fault:
Diagnosability Result

Global chronicles associated to fault: Customer.AddressError

```
CustAddressAcquisitionErrorShopAddressCustomerInfectionSuppNormalShopAnnulationWhAddressInfection
CustAddressAcquisitionErrorShopAddressCustomerInfectionSuppNormalShopConfirmationWhAddressInfection
```

Diagnosability Result

```
Not exclusive (Fault mode =Customer.AddressError) with chronicle CustTimeOutPackageReceptionShopNormalAllAvailableConfirm
              not exclusive with CustAddressAcquisitionErrorShopAddressCustomerInfectionSuppNormalShopAnnulationWhAddr

Customer.AddressError and Warehouse.DBerror are DISCRIMINABLE
Customer.AddressError and Customer.TimeOutShopAnswer are DISCRIMINABLE
Customer.AddressError and Customer.AddressError are DISCRIMINABLE
Customer.AddressError and Customer.ItemError are DISCRIMINABLE
Customer.AddressError and Warehouse.TimeOutReceiveShipment are DISCRIMINABLE
Customer.AddressError and Warehouse.HumanError are DISCRIMINABLE
Customer.AddressError and Supplier.TimeOutShopConfirmationOrder are DISCRIMINABLE
Customer.AddressError and Supplier.ErrorDB are DISCRIMINABLE
Customer.AddressError and Warehouse.TimeOutShopConfirmationOrder are DISCRIMINABLE
Customer.AddressError and Shop.TimeOutCustomerConfirmation are DISCRIMINABLE
Customer.AddressError and Customer.TimeOutPackageReception are NOT DISCRIMINABLE
Customer.AddressError and Shop.TimeOutInvokeWarehouse are DISCRIMINABLE
Customer.AddressError and Shop.TimeOutInvokeSupplier are DISCRIMINABLE


Customer.AddressError and the normal mode are DISCRIMINABLE
```

Figure 18: Diagnosability results

this example 49 chronicles have been defined (customer(11) warehouse(11), shop(20), supplier(7)) associated to the normal behavior of the foodshop and to 13 faults.

– Diagnosability tab: this is the main tab of the software. It presents the global view and the diagnosability analysis of the workflow. On the left are listed the set of faults prefixed with the service in which the fault occurs. 'Customer.AddressError' means that the customer provided a wrong address, 'Customer.itemError' means that the customer provided a wrong item to order. 'errorDB' are database errors in different services. Figure 18 shows the diagnosability analysis applied to the fault 'Customer.AddressError': Exclusiveness tests that answer 'No' are printed on the screen, they provide the reason why two faults are not discriminable. In this example, the fault 'Customer.AddressError' is discriminable from the nominal behavior of the Web Services but not from the fault 'TimeOutpackageReception'.

7 Conclusion

This paper fully addresses the problem of analyzing the diagnosability of Web Services workflow by analyzing the performance of a preexisting diagnostic approach based on chronicle recognitions. The first contribution of this paper is a complete characterization of the diagnosability problems induced by this diagnostic algorithm. The main idea consists in translating the chronicle recognition problem into an event-based diagnosability analysis. We then propose to reduce the diagnosability problem to a set of exclusiveness tests. As opposed to a classical diagnosability problems on discrete-event system, one main difficulty here is to deal with the delays between events defined in the chronicle models. Thus a second contribution of the paper in order to solve this issue is on the modeling aspects of the chronicles. We propose to translate chronicle models into Time Petri nets to benefit of the time analysis capabilities. The use of Time Petri nets allows to generate a time abstraction to obtain a finite representation of the chronicle behaviors. A diagnosability checker has been derived from this study and tested in the framework of the European project WS-DIAMOND. We propose to extend this work in the following manner: currently, the approach only deals with simple fault, it is necessary to consider the case when multiple and independent faults can occur during on one run of the workflow. The modeling aspects could also be extended by the use of Time Petri nets with priority for a simpler generation of synchronized behaviors. Finally, the challenge is to extend this analysis to help in the design of workflow by pointing out where the discriminability problems are.

References

[Alur and Dill, 1994] Alur, R. and Dill, D. (1994). A theory of timed automata. *Theoretical Computer Sciences*, 126:183–235.

[Basseville, 2001] Basseville, M. (2001). On fault detectability and isolability. *European Journal of Control*, 7(8):625–637.

[Berthomieu and Diaz, 1991] Berthomieu, B. and Diaz, M. (1991). Modeling and verification of time dependent systems using time Petri nets. *IEEE Transactions on Software Engineering*, 17(3):259–273.

[Berthomieu and Menasche, 1983] Berthomieu, B. and Menasche, M. (1983). An enumerative approach for analyzing time Petri nets. In *IFIP Congress Series*, volume 9, pages 41–46. Elsevier Science Publ. Comp.

[Berthomieu et al., 2006] Berthomieu, B., Peres, F., and Vernadat, F. (2006). Bridging the gap between timed automata and bounded time Petri nets. In *Proceedings of Formal Modeling and Analysis of Timed Systems (FORMATS'06), LCNS 4202.*

[Berthomieu et al., 2004] Berthomieu, B., Ribet, P.-O., and Vernadat, F. (2004). The tool TINA – construction of abstract state spaces for Petri nets and time Petri nets. *International Journal of Production Research*, 42(14).

[Bertrand et al., 2007] Bertrand, O., Carle, P., and Choppy, C. (2007). Chronicle modelling using automata and colored Petri nets. In *18th Internation Workshop on Principles of Diagnosis (DX'07)*, pages 243–248, Nashville, TN, USA.

[Boufaied et al., 2002] Boufaied, A., Subias, A., and Combacau, M. (2002). Chronicle modeling by Petri nets for distributed detection of process failures. In *IEEE International conference on systems, man and cybernetics (SMC'02)*, pages 243–248, Hammamet, Tunisia.

[Chen and Patton, 1994] Chen, J. and Patton, R. (1994). A re-examination of fault detectability and isolability in linear dynamic systems. In *Proceedings of the 2nd Safeprocess Symposium.*

[Contant et al., 2004] Contant, O., Lafortune, S., and Teneketsis, D. (2004). Diagnosis of intermittent faults. *Journal of Discrete Event Dynamic Systems: Theory and Applications*, 14:171–202.

[Cordier et al., 2007] Cordier, M.-O., Guillou, X. L., Robin, S., Rozé, L., and Vidal, T. (2007). Distributed chronicles for on-line diagnosis of web services. In Biswas, G., Koutsoukos, X., and Abdelwahed, S., editors, *18th International Workshop on Principles of Diagnosis*, pages 37–44.

[Cordier et al., 2006] Cordier, M.-O., Travé-Massuyès, L., and Pucel, X. (2006). Comparing diagnosability in continous and discrete-event systems. In González, C., Escobet, T., and Pulido, B., editors, *17th International Workshop on Principles of Diagnosis*, pages 55–60.

[D4.2, 2006] D4.2 (2006). Specification of diagnosis algorithms for web-services - phase 1. Technical report, WS-DIAMOND European project.

[D5.1, 2007] D5.1 (2007). Characterization of diagnosability and repairability for self-healing web services. Technical report, WS-DIAMOND European project.

[Dousson, 1996] Dousson, C. (1996). Alarm driven supervision for telecommunication networks: *ii*- on line chronicle rcognition. *Annales des Télécommunications*, 51 (9-10):501–508.

[Eder et al., 2007] Eder, J., Lehmann, M., and Tahamtan, A. (2007). Conformance test of federated choreographies. In *Proc. of the 3rd International Conference on Interoperability for Enterprise Software and Applications.*

[Hack, 1976] Hack, M. (1976). Petri net languages. Technical report, MIT, Cambridge, Massuchets, USA.

[Jantzen, 1987] Jantzen, M. (1987). Language theory of Petri nets. *lecture notes in computer Sciences*, 254-I:397–412.

[Jéron et al., 2006] Jéron, T., Marchand, H., Pinchinat, S., and Cordier, M.-O. (2006). Supervision patterns in discrete event systems diagnosis. In *WODES06: Workshop on Discrete Event Systems.*

[Mayer, 1998] Mayer, E. (1998). Inductive learning of chronicles. In *ECAI'98: Proceedings of European Conference on Artificial Intelligence*, pages 471–472.

[Merlin and Farber, 1976] Merlin, P. and Farber, D. (1976). Recoverability of communication protocols: Implication of a theoretical study. *IEEE Trans. Comm.*, 24(9):1036–1043.

[Pencolé, 2004] Pencolé, Y. (2004). Diagnosability analysis of distributed discrete event systems. In *ECAI04: Proceedings of European Conference on Artificial Intelligence*, pages 43–47.

[Quiniou et al., 2001] Quiniou, R., Cordier, M.-O., Carrault, G., and Wang, F. (2001). Application of ilp to cardiac arrhythmia characterization for chronicle recognition. *Lecture Notes in Computer Science*, (2157):220–227.

[Sampath et al., 1995] Sampath, M., Sengputa, R., Lafortune, S., Sinnamohideen, K., and Teneketsis, D. (1995). Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40:1555–1575.

[Staroswiecki and Comtet-Varga, 1999] Staroswiecki, M. and Comtet-Varga, G. (1999). Fault detectability and isolability in algebraic dynamic systems. In *Proceedings of the European Control Conference.*

[Team, 2008] Team, W.-D. (2008). Ws-diamond web services - diagnosability, monitoring and diagnosis. Technical Report 08005, LAAS-CNRS.

[Travé-Massuyès et al., 2001] Travé-Massuyès, L., Escobet, T., and Milne, R. (2001). Model-based diagnosability and sensor placement application to a frame 6 gas turbine subsystem. In *IJCAI01: Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, volume 1, pages 551–556.