Towards Timed Requirement Verification for Service Choreographies

Nawal Guermouche, Silvano Dal Zilio

▶ **To cite this version:**

Nawal Guermouche, Silvano Dal Zilio. Towards Timed Requirement Verification for Service Choreographies. 8th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing, Oct 2012, Pittsburgh, United States. pp.10. hal-00578436v2

HAL Id: hal-00578436

https://hal.science/hal-00578436v2

Submitted on 21 Mar 2012 (v2), last revised 20 Sep 2012 (v4)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

hal-00578436, version 2 - 21 Mar 2011

Formal Requirement Verification for Timed Choreographies *

Nawal Guermouche[1,2] and Silvano Dal Zilio[1,2]

[1] CNRS ; LAAS ; 7 avenue colonel Roche, F-31077 Toulouse, France
[2] Université de Toulouse ; UPS, INSA, INP, ISAE, UT1, UTM, LAAS ; F-31077 Toulouse, France

Abstract. Time plays a crucial role when reasoning about the composition of Web Services. Nonetheless, while the addition of temporal aspects in the specification of services improves expressiveness, it also makes reasoning about service composition much harder.
In this work, we propose an approach for analyzing and validating a composition of services with respect to time related properties. We consider services defined using an extension of the Business Process Execution Language (BPEL) where timing constraints can be associated to the execution of an activity or on the delay between events. The goal is to check whether a choreography obtained from the composition of timed services satisfies given real time requirements. Our approach is based on a formal interpretation of timed choreographies in the Fiacre verification language that defines a precise model for the behavior of services and their timed interactions. We also rely on a logic-based language for property definition to formalize real-time requirements and on specific tooling for model-checking Fiacre specifications.

Key words: Timed BPEL processes, choreography analysis, asynchronous services, real-time requirements, formal verification.

1 Introduction

Web Services are a set of standards that enable the definition of complex software systems based on the composition of autonomous and heterogeneous services. Web Services programming relies on a set of XML based standards, such as the Web Service Description Language (WSDL) [26], for the description of services interface, or the Business Process Execution Language (BPEL) [23], for defining service orchestration. The notion of *choreography* (see e.g. [2]) is useful to reason about the collaboration of services from a global viewpoint. Basically, a choreography provides a way to specify the overall behavior expected from the composition of services. Since time plays a crucial role when reasoning about business processes, we need to be able to express the time related attributes of a choreography and we need to be able to check timing requirements on them.

In this paper, we propose an approach for analyzing and validating choreographies with respect to time related properties. We consider *timed services* defined

* This work was partly supported by the JU Artemisia project CESAR and by the ANR project ITEmIS.

hal-00578436, version 2 - 21 Mar 2011

using an extension of BPEL with timing constraints. Our goal is to check whether a *timed choreography*, obtained from the composition of timed services, satisfies a given requirement. We choose a rich model for expressing timing information where constraints can be associated to the execution time of (basic and structured) activities as well as on the delay between two events. In our context, an event may be *local* to a given service—for instance an invocation or the end of an activity—or may be *global*, associated to a message crossing service boundaries. For instance, we can specify a delay between two asynchronous messages in a choreography. In our work, we take inspiration from the Business Process Modeling Notation (BPMN) as a graphical syntax for defining timed choreographies. We give an example of an annotated timed choreography in Fig. 1.

Taking into account temporal (quantitative) aspects in the specification of services improves expressiveness. However, it makes reasoning about service composition much harder and makes known problems more challenging. This is the case, for instance, with *timed compatibility* [16, 25], the problem of checking that every service request is eventually acknowledged (or equivalently that the system is deadlock-free). While compatibility is an important property for a distributed system, it is not the only useful requirement. For instance, it is also useful to check the "worst-case execution time" of a choreography or to check that a partial deadline is met. In our work, we define a formal model for the semantics of timed choreographies and propose a model-based approach for checking real-time requirements. In this context, requirements are defined using a logical-based formalism that is able to express real-time constraints between the occurrence of events; we use a property pattern language that defines a fragment of Metric Interval Temporal Logic (MITL), a real-time extension of Linear Temporal Logic. As a result, our framework can be used to check very general properties, that go beyond the mere absence of deadlocks. We give some examples of real-time requirements that can be checked automatically in Section 3.

The formal semantics of timed choreographies is defined using an interpretation of services in Fiacre [7, 8], a formal modeling language that can be used to represent the behavioral and timing aspects of a system. The use of Fiacre is well adapted for this context since it has been designed both as the target language of model transformation engines—interpretation have already been defined for system description languages such as SDL, UML or AADL—and as the source language for formal verification toolboxes, such as CADP or Tina [6], the TIme Petri Net Analyzer. In Section 5 we show how we can use our interpretation of timed BPEL processes in Fiacre and the model-checking tools provided in Tina in order to check real-time requirements of timed choreographies.

The paper is organized as follows. In Section 2 we describe related works in the domain of the formal analysis of Web Services and outline our contributions. In Section 3 we define an example of timed choreography that is used to give an informal overview of our framework and its expressiveness. Section 4 details our framework more formally and define our interpretation of timed processes in Fiacre. Before concluding, we report on some experimental results obtained using a prototype implementation of our framework.

hal-00578436, version 2 - 21 Mar 2011

2 Related Work

We list a series of works related to the formal analysis of Web Services. In [3, 4, 9], the authors address the problem of checking compatibility between two services. Their approach is restricted to synchronous services and is exclusively based on the sequences of messages that can be exchanged by a service. These assumptions are quite restrictive since, for instance, two services may engage in a successful conversation even if their behaviors do not have the same branching structure. On the other hand, compatible services may exhibit very different behaviors when time is taken into account. Compatibility between timed services has been first studied by Benatallah et al. [5]. This work has been latter extended in [24, 25]. In these approaches, it is only possible to specify the delay between two messages inside the same service, whereas we consider richer time constraints. Moreover, these works address only one specific requirement—absence of deadlocks—while we allow the verification of more general properties.

Eder and Tahamtan introduce the notion of *time conformance* in [12]. This is defined as the problem of checking whether a timed orchestration satisfies a given timed choreography. In this framework, time constraints are limited to expressing the execution time of basic activities. This approach is different in nature from the one followed in our work. Indeed, our goal is to check properties directly on the choreography; we do not suppose that the choreography "is correct" and that parts of its implementation—a timed orchestration—should conform to this specification. More generally, the distinction is the same than between a model-checking (our case) and a behavior conformance approach. We should point out that, in our case, we solve a model-checking problem for a real-time extension of temporal logic, since we consider timed models, and for a dense time model. Another instance of a "behavior conformance" problem is studied by Héam et al [18, 19] that focus on the problem of substituting a service by another. These works address both the temporal and financial costs of operations but are restricted to choreographies and do not consider time constraints over structured activities or across service boundaries.

Kazhamiakin et al. [20] adopt a formalism closely related to timed automata to model the behavior of a timed orchestration. This work is based on a discrete-time variant of the Duration Calculus for expressing requirements, while we work with a dense model of time. Furthermore, they consider—other than timeouts—the temporal cost of manual activities and deal with synchronous services, while, in our framework, we propose a richer temporal model. Another difference is that we allow the declaration of more expressive real-time requirements, like for example *absence patterns*, that states that parts of a system's execution are free of certain events. Our approach extends previous works based on timed automata [15, 17] that are concerned with the compatibility analysis of timed services. In this work, verification is restricted to checking the absence of deadlock in a composition of services. Moreover, the model can only express temporal constraints on the delays between the exchange of messages and not on the duration of activities.

Our Contributions. our framework provides a rich temporal model, able to express constraints more expressive than those found in the related work: delays

hal-00578436, version 2 - 21 Mar 2011

between messages; duration of structured activities; timeouts; etc. As a consequence, our first contribution lies in the definition of a rich model for timed service composition. This model integrates efficiently the real-time and concurrent characteristics of timed services and is compatible with both synchronous and asynchronous services.

Our second contribution consists in defining new class of real-time requirements that goes far beyond the mere absence of deadlock. In Section 3 we illustrate the use of this model in the context of a healthcare scenario and show how time constraints and requirements can be expressed. We also explain where this information could be reasonably stored: WSDL and BPEL for local constraints and Service Level Agreement (SLA) contracts for global constraints.

Our final contribution is the definition of an interpretation of timed business processes into Fiacre, a formal verification language with native support for expressing concurrency and time-constrained interactions. This interpretation has been implemented in a tool that takes a collection of annotated BPEL processes as input and returns the associated Fiacre model. Our tool is based on EasyBPEL (http://easybpel.petalslink.org/), a library that provides a BPEL 2.0 engine to orchestrate services. With this approach, we are able to apply model-checking tools in order to automatically check the timed compatibility of a composition of services as well as complex real time requirements defined by the users.

3 Global Overview of the Modeling Framework

We describe the different features of our framework with the help of an example of timed choreography. Our modeling framework relies on a syntax for expressing services (based on BPMN for the choreography aspects); temporal constraints annotations; and a requirement language based on real-time property specification patterns. The verification framework is defined in the next section. We opted for a scenario from the healthcare domain related to patient handling during a medical examination. The scenario involves three entities, each one managed by a service: (1) the medical consultation clinic; (2) a medical analysis laboratory; and (3) a pharmacy.

The choreography is depicted in Fig. 1, where each service is in a different swimlane. With the aid of the *medicalConsultationService* (MCS), a doctor can check the social security number (ssn) of a patient. If the ssn is valid, then the MCS may ask the *medicalAnalysisService* (MAS) to perform some medical analyses and, in parallel, asks for a radiography. Once the medical analyses are fulfilled—and after analyzing the different medical data—a medical report is compiled and drugs can be ordered.

3.1 Temporal Constraints

The services may interact using asynchronous and synchronous exchange of messages. As usual, the choreography imposes constraints on the order of these messages. It also defines temporal constraints through the use of annotations. We consider two kinds of temporal constraints in our framework, local and global.

hal-00578436, version 2 - 21 Mar 2011

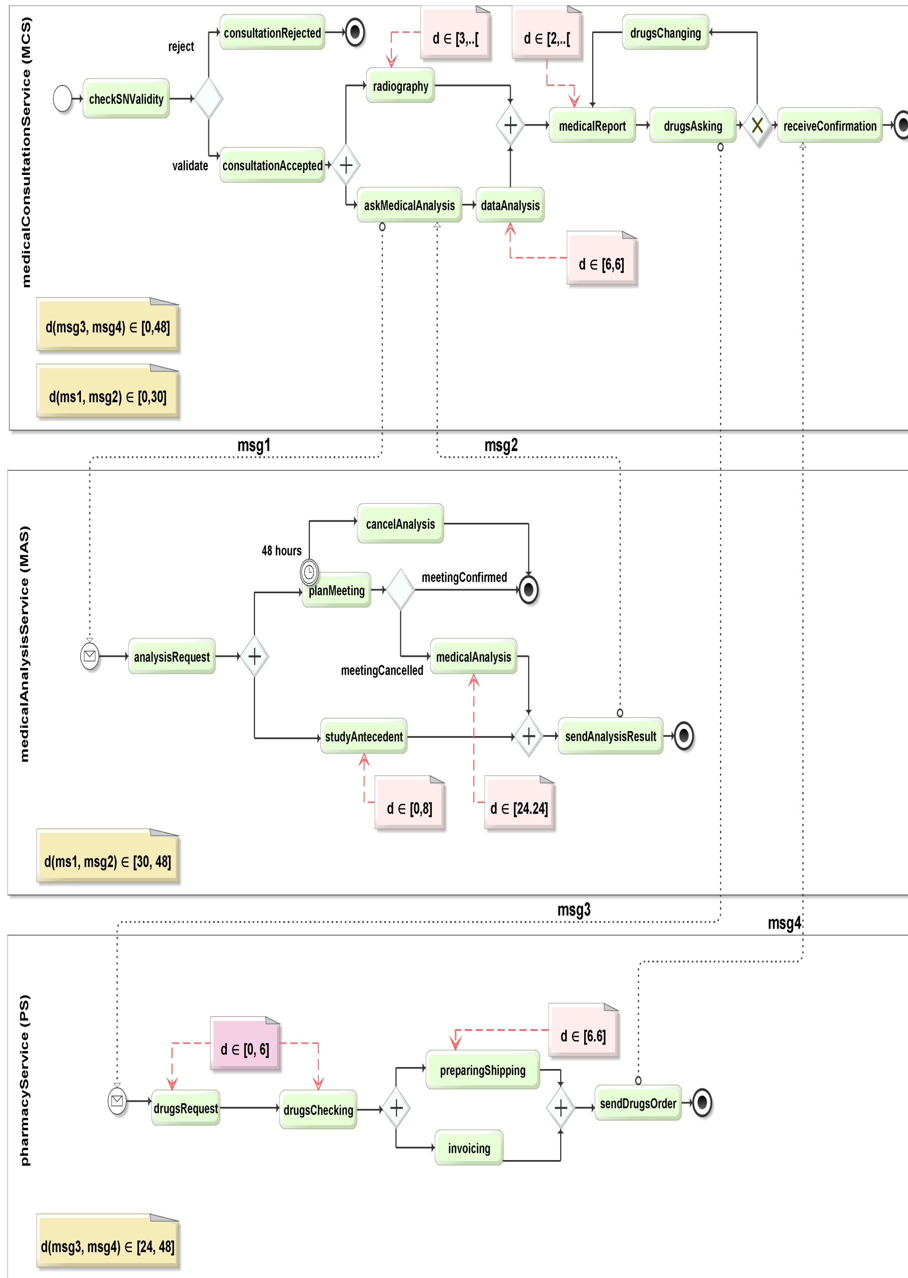


Fig. 1. Global view of the health-care application

Local temporal constraints are associated to the execution of a service. They are used to specify the duration and/or the delays required to perform an activity. We consider two kinds of local constraints:

hal-00578436, version 2 - 21 Mar 2011

(1) *temporal costs* are used to define the estimated execution time of an activity, say A, and are of the form $d(A) \in I$, where I is a time interval[1]. For instance, the medical report operation in the MCS requires at least 2 hours: $d(\mathit{medicalReport}) \in [2, ..[$. The most suitable place to store these constraints is in the WSDL file that describes the operation;

(2) *temporal delays* are used to specify the expected delay between two activities and are of the form $d(A_1, A_2) \in I$. For instance, the Pharmacy Service performs the *drugsChecking* activity between 6–12 hours after the start of the service: $d(\mathit{drugsRequest}, \mathit{drugsChecking}) \in [6, 12]$. A temporal delay expresses a commitment from the service, much like a timeout; it is not a requirement that should be checked a posteriori. (Note that we also allow the use of timeouts, such as in the activity *planMeeting* of MAS) The most suitable place to store temporal delays is to add annotations in the BPEL file that describes the service.

Global temporal constraints are used to specify the temporal contract of a service and are associated to pair of messages (dotted lines in our diagram) that cross service boundaries. For example, the Pharmacy Service promises that the drug order is delivered within 24–48 hours of the drug request: $d(\mathit{msg}_3, \mathit{msg}_4) \in [24; 48]$. Global constraints live at the same level than Service Level Agreement contracts.

3.2 Real-Time Requirements

This section gives a description of the specification patterns available in our framework. A complete description of the language is given in [1]. Our language extends the property specification patterns of Dwyer et al. [11] with the ability to express time delays between the occurrences of events. The result is expressive enough to define properties like the compliance to deadline, bounds on the worst-case execution time, etc. The advantage of this approach is to provide a simple formalism to non-experts for expressing properties. Another benefit is that properties expressed with this pattern language can be directly used with our model-checking tools (see our experiments in Sect. 5). The pattern language follows the classification introduced in [11], with patterns arranged in categories such as occurrence or order patterns. In the following, we study examples of *response* and *absence* patterns.

Absence pattern with delay. This category of patterns can be used to specify delays within which activities must not occur. A typical pattern in this category can be used to assert that an activity, say A_2, cannot occur between d_1–d_2 units of time after the occurrence of an activity A_1. This requirement corresponds to a basic absence pattern in our language:

$$\texttt{Absent}\ A_2\ \texttt{after}\ A_1\ \texttt{within}\ [d_1; d_2]\ . \qquad \text{(absent)}$$

An example of use for this pattern is the requirement that we cannot have two medical analyses for the patient in less than 10 days (240 hours):

$$\texttt{Absent}\ \mathit{MAS.medicalAnalysis}\ \texttt{after}\ \mathit{MAS.medicalAnalysis}\ \texttt{within}\ [0; 240]\ .$$

[1] Time intervals may be open, like $]2.5, 3]$, or unbounded, like $[1, ..[$.

hal-00578436, version 2 - 21 Mar 2011

A more complicated example of requirement is to impose that if a doctor does not cancel a drug order within 6 hours, then it should not cancel drugs for another 48 hours. This requirement can be expressed using the composition of two absence patterns:

$$(\texttt{absent}\ MCS.\mathit{drugsChanging}\,\texttt{after}\ MCS.\mathit{drugsAsking}\,\texttt{within}\,[0;6])$$
$$\Rightarrow (\texttt{absent}\ MCS.\mathit{drugsChanging}\,\texttt{after}\ MCS.\mathit{drugsAsking}\,\texttt{within}\,[0;54])\ .$$

Response pattern with delay. This category of patterns can be used to express delays between events, like for example constraints on the execution time of a service. The typical example of response pattern states that every occurrence of an event, say e_1, must be followed by an occurrence of an event e_2 within a time interval I. This pattern is denoted:

$$e_1\,\texttt{leadsto}\ e_2\,\texttt{within}\ I\ . \qquad \text{(leadsto)}$$

In our framework, we use the notation $S.\mathit{init}$ and $S.\mathit{end}$ to refer to a start, respectively an end, event in the service S. Hence, we can check that the execution time of the service S is less than d units of time with the requirement $S.\mathit{init}$ `leadsto` $S.\mathit{end}$ `within` $[0;d]$. We can use the same pattern to express requirements on an activity. For instance that drugs must be delivered within 48 hours of the medical examination start:

$$MCS.\mathit{init}\,\texttt{leadsto}\ PS.\mathit{sendDrugsOrder}\,\texttt{within}\,[0;48]\ .$$

Using a composition of patterns with the conjunction operator, we can bound the time between the start of the initiating service (the client) and the end of the choreography (if any). For instance, in our motivating example, we would like to check that the medical examination last less than 60 hours:

$$\begin{aligned} & MCS.\mathit{init}\,\texttt{leadsto}\ MCS.\mathit{end}\,\texttt{within}\,[0,60] \\ \wedge\ & MCS.\mathit{init}\,\texttt{leadsto}\ MAS.\mathit{end}\,\texttt{within}\,[0,60] \\ \wedge\ & MCS.\mathit{init}\,\texttt{leadsto}\ PS.\mathit{end}\,\texttt{within}\,[0,60]\ . \end{aligned}$$

More generally, we can check that a service, say S_2, always terminates its execution after service S_1 within a duration d with the requirement: $S_1.\mathit{end}$ `leadsto` $S_2.\mathit{end}$ `within` $[0;d]$

Local and global compatibility. To conclude this section, we study how the *timed compatibility* problem [5, 15, 25] translates in our setting. Using the interpretation of BPEL processes defined in the next section, we can apply model-checking techniques to analyze each service separately and check whether its local temporal constraints are not mutually contradictory and in accordance with its "contracts." This is obtained by checking the absence of deadlocks on the interpretation of the service. In this case we say that the services are locally coherent.

A choreography is obtained by the composition of services. Even though each service is locally coherent, service composition may still break the local and global temporal constraints of a service. Again, proving that the composition is sound with respect to all these constraints can be reduced to checking a simple reachability property on the interpretation of the choreography—like the absence of deadlocks. This is the equivalent of timed compatibility in our setting.

hal-00578436, version 2 - 21 Mar 2011

4 Verification Framework

In this section, we consider the framework used to define the formal semantics of timed choreographies. Our approach is based on an interpretation of services in Fiacre [7, 8], a formal verification language that can model the behavioral and timing aspects of a system. This method has some advantages compared to related work where the semantics of services is given using a dedicated formalism (see e.g. [21]) or a low-level formalism, like timed automata [15]. Indeed, Fiacre provides high-level operators, special support for different concurrency paradigm and a hierarchical (component-based) structure that simplify the encoding of system description languages. Moreover, the language comes equipped with a set of dedicated tooling: compilers to different model-checking tool suite (like CADP or Tina [6]); support for the real-time requirement language described in the previous section [1]; and support for Model-Driven Engineering. In particular, Fiacre is the intermediate language used for model verification in Topcased [7, 13]—an Eclipse based toolkit for critical systems (`http://www.topcased.org/`)—where it used as the target of model transformation engines from various languages, such as SDL, UML or AADL.

4.1 The Fiacre Language

Fiacre offers a formal representation of both the behavioral and timing aspects of systems for formal verification and simulation purposes. The design of the language is inspired from decades of research on concurrency theory and real-time systems theory. For instance, its timing primitives are borrowed from Time Petri nets [22], while the integration of time constraints and priorities into the language can be traced to the BIP framework [10]. Fiacre processes can interact both through synchronization (message-passing) and access to shared variables (shared-memory). A formal definition of the language is given in [8].

Fiacre programs are stratified in two main notions: *processes*, which are well-suited for modeling structured activities, and *components*, which describes a system as a composition of processes, possibly in a hierarchical manner. The language is strongly typed, meaning that type annotations are exploited in order to guarantee the absence of unchecked run-time errors.

A *process* is defined by a set of *control states* (say $\mathtt{s}_0, \mathtt{s}_1, \dots$) and parameters, each associated with a set of *complex transitions*, which are programs specifying how parameters are updated and which transitions may fire. For example, the process declaration:

$$\textbf{process}\ \mathtt{P}[\mathtt{q} : \textbf{none}](\&\mathtt{v} : \textbf{nat})\ \textbf{is}\ \dots$$

expresses that `P` is a process that uses the port `q` for synchronization—the port carries no data—and has one parameter, `v`, that is a (reference to a) shared variable holding natural values. Complex transitions are built from expressions and deterministic constructs available in typical programming languages (assignments, conditionals, while loops and sequential compositions), nondeterministic constructs (nondeterministic choice and assignments) and communication events on ports. For example, the transition:

```
from s0 select v :=v+1; to s1
            [] on(v=0); to s0
       end
```

states that, in state s_0, the process may choose nondeterministically between two alternatives. Either increments the value of the variable `v` and moves to s_1, or loops to s_0 if `v` is nil.

A *component* is defined as the parallel composition of processes and/or other components, expressed with the operator **par** ... || ... **end**. While components are the unit of composition, they are also the unit for process instantiation and for ports and shared variables creation. The syntax of components allows to restrict the access mode and visibility of shared variables and ports, to associate timing constraints with communications and to define priority between communication events. For example, the following declaration states that `C` is a component with a private port `r`—synchronization over `r` is in time 0—and two fresh instances of the process `P`.

```
component C(x : nat) is
   port r:none in [0,0]
   var v1:nat :=x, v2:nat :=3
   par P[r](v1) || P[r](v2) end
```

4.2 Interpretation of BPEL in Fiacre

We define an interpretation of timed BPEL processes where a service is modeled by a Fiacre component and such that service invocation (both synchronous and asynchronous) is modeled using shared variables. In this approach, the interpretation of a choreography is simply the parallel composition of its services. We only detail the encoding of a representative subset of activities. Moreover, this simple encoding does not take into account the expected behavior in case of faults or communication failures.

Interpretation of communication. We model communication using a shared variable that acts as a buffer counting the number of messages exchanged between services; each WSDL message, say msg_i, is represented by an integer variable, $\texttt{msgVar}_i$, with initial value 0. Then (an asynchronous) message emission is encoded by incrementing the variable and reception by decrementing it. For example, reception of the message *msg* is encoded by the Fiacre expression: **on**(`msgVar`>0); `msgVar`:=`msgVar`−1 (we use the **on** expression to test that the "message channel" is not empty). This simple approach can be extended to take into account the values exchanged in a message, as long as the number of values stay finite.

Interpretation of activities as processes. A service S is encoded by a component `S`. We encode each activity A_i in S by a Fiacre process $\texttt{A}_i$ with two specific ports: `ps` for signaling the start of the activity and `pe` for signaling its end. A `<flow>` activity in BPEL is used to execute sub-activities,

hal-00578436, version 2 - 21 Mar 2011

hal-00578436, version 2 - 21 Mar 2011

$A_1, \ldots, A_n$, in parallel. In our encoding, a flow activity inside the service S is turned into a parallel composition $\mathtt{A}_1$(ps,pe) || ... || $\mathtt{A}_n$(ps,pe), such that instances of the $\mathtt{A}_i$'s are synchronized on their start and end event. Likewise, a <sequence> activity can be encoded with a parallel composition $\mathtt{A}_1$(ps,$\mathtt{ps}_1$) || $\mathtt{A}_2$($\mathtt{ps}_1$,$\mathtt{ps}_2$) || ... || $\mathtt{A}_n$($\mathtt{ps}_{(n-1)}$,pe), such that the "end" of $\mathtt{A}_i$ is synchronized to the start of the process $\mathtt{A}_{i+1}$. We describe an example of encoding for the <pick> activity in the paragraph on the interpretation of timeouts. We define the processes corresponding to the basic activities <receive>, <reply> and <invoke>.

```
process Receive[ps:none, pe:none](&msgVar:nat) is
  states start, s1, s2, s3 init to start
  from start ps; to s1
  from s1 on(msgVar>0);wait[0,0];msgVar:=msgVar-1;to s2
  from s2 pe; to s3
```

```
process Reply[ps:none, pe:none](&msgVar:nat) is
  states start, s1, s2, s3 init to start
  from start ps; to s1
  from s1 wait[0,0];msgVar:=msgVar+1;to s2
  from s2 pe; to s3
```

```
process Invoke[ps:none, pe:none](&msgOutVar,&msgInVar:nat)is
  states start, s1, s2, s3, s4 init to start
  from start ps; to s1
  from s1 wait[0,0];msgOutVar:=msgOutVar+1;to s2
  from s2 on(msgInVar>0);wait[0,0];msgInVar:=msgInVar-1;to s3
  from s3 pe; to s4
```

Interpretation of basic activities (optimization). We propose another interpretation for basic activities that are not executed in parallel. In this case, we map every activity to a state in a single Fiacre process, say P. The goal is to obtain a more efficient encoding (with less processes and states). In this context, an activity A_i is mapped to a state $\mathtt{s}_i$ in P and its effect is implemented by a transition from the state $\mathtt{s}_i$ to the state $\mathtt{s}_j$ such that A_j is the next in line from A_i in the sequence.

<receive>	**from** $\mathtt{s}_i$ **on** (MsgVar>0); **wait**$[0,0]$; msgVar:=msgVar−1; **to** $\mathtt{s}_j$
<reply>	**from** $\mathtt{s}_i$ **wait**$[0,0]$; msgVar:=msgVar+1; **to** $\mathtt{s}_j$
<invoke>	**from** $\mathtt{s}_i$ **wait**$[0,0]$; msgOutVar:=msgOutVar+1; **to** s'_i **from** s'_i **on**(msgInVar>0); **wait**$[0,0]$; msgInVar:=msgInVar−1; **to** $\mathtt{s}_j$

Interpretation of temporal constraints. We start by describing the interpretation of temporal cost. An activity that can be executed within a delay is modeled as a transition that takes place after a timed interval. We consider two categories of activity that can have a temporal cost—*one-way operation* and *request-response operation*—and define the encoding in the optimized case. In this context, we map an operation A_i to a pair of states $\mathtt{s}_i$ and $\mathtt{s}'_i$. The effect of A_i is implemented

hal-00578436, version 2 - 21 Mar 2011

by a sequence of two transitions from the state $\mathtt{s}_i$ to the state $\mathtt{s}_j$, where A_j is the activity that logically follows A_i. We give below the encoding of one-way and request-response `<invoke>` operations with an execution time of d.

(one-way) **from** $\mathtt{s}_i$ **on**(msgVar>0);**wait**$[0,0]$;msgVar:=msgVar−1;**to** $\mathtt{s}'_i$
from $\mathtt{s}'_i$ **wait**$[d,d]$;**to** $\mathtt{s}_j$

(request-response) **from** $\mathtt{s}_i$ **on**(msgVar>0);**wait**$[0,0]$;msgVar:=msgVar−1;**to** $\mathtt{s}'_i$
from $\mathtt{s}'_i$ **wait**$[d,d]$;msgVar:=msgVar+1; **to** $\mathtt{s}_j$

The interpretation of temporal delays and global constraints are similar. We associate to every delay constraint of the form $d(A_1, A_2) \in I$ a process in Fiacre, say `TObs`. The role of this process is to observe the delay between the end of A_1 (synchronization on the port $\mathtt{pe}_1$) and the start of A_2 (synchronization on $\mathtt{ps}_2$). The encoding is very similar for a delay between messages. We give below the "time observer" process corresponding to a constraint of the form $d(A_1, A_2) \in [0; d]$ (the **unless** operator is used to state that the transition to `err` has an higher priority). We can test if the constraint is violated by checking whether the process `TObs` can reach the state `err`.

```
process TObs[pe1, ps2:none]() is
  states start, s1, s2, err init to start
  from start pe1; to s1
  from s1 select ps2;to s2
          unless wait[d,d];to err
          end
```

Interpretation of timeouts (onAlarm). We describe our interpretation of a timer-based alarm, `<onAlarm for ="d">`, using the example of a simple `<pick>` activity A_i such that:

A_i = `<pick><onMessage name="m"/>`
`<onAlarm for="d">`A_k`</onAlarm></pick>`

meaning that the activity will select the activity A_k after d unit of time unless it receives the message `m` before. The activity A_i may be encoded using the following transition (we consider the case of the optimized encoding):

```
from si select wait[0,0];on(msgVar>0);msgVar:=msgVar−1;to sj
          unless wait[d,d];to sk
          end
```

4.3 Interpretation of the Healthcare Scenario

We take the example of the healthcare scenario to study the result of our interpretation on an example. The architecture of the Fiacre component corresponding to the Pharmacy Service is displayed in Fig. 2. The complete Fiacre source code for the example can be found in a long version of this paper [14].

The Pharmacy Service (PS) has three temporal constraints: (c_1) the activity *drugsChecking* must be done within 6–12 hours of receiving the drugs request

(a local, delay constraint); (c_2) the execution time of activity *preparingShipping* is 6 hours (a local, temporal constraint); and (c_3) the message sent by activity *sendDrugsOrder* should be emitted within 24–48 hours from receiving the drugs request (this is a global constraint). The PS is built from a `<sequence>` activity that contains two operations that should be performed in parallel (inside a `<flow>` activity): *preparingShipping* and *invoicing*. Therefore, in our (optimized) interpretation, the component `PS` contains the parallel composition of three processes—one for the sequence and one for each activity inside the `<flow>`—added to a "time observer" process for the two constraints c_1 and c_3. These four processes appear in the diagram of Fig. 2 with an explicit naming convention.

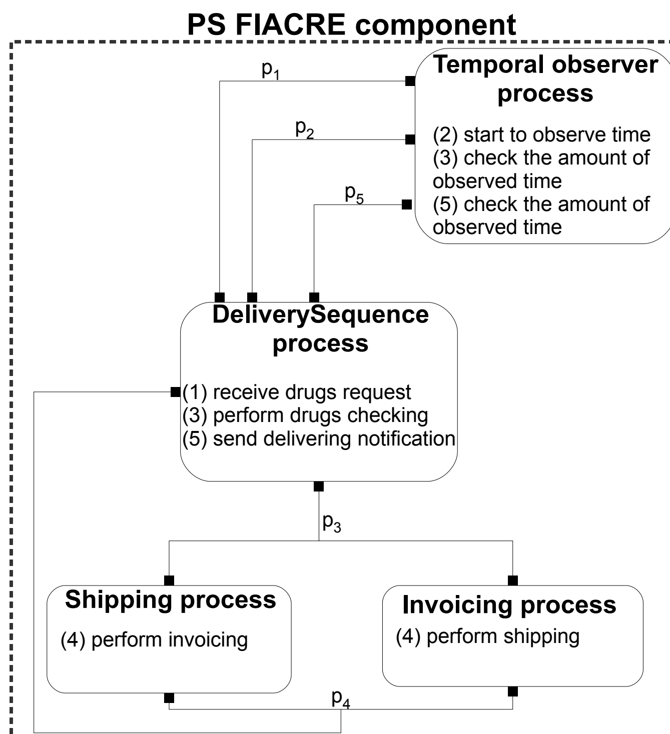


Fig. 2. Architecture of the Fiacre process for the Pharmacy Service

We can describe the possible interactions of the `PS` component as follows. First, the `deliverySequence` process starts by receiving a drugs request. This message is labeled msg_3 in the diagram of Fig. 1 and is modeled using a shared variable $\texttt{msgVar}_3$ that is a parameter of `PS`. This event eventually triggers a transition in the process `deliverySequence` that corresponds to the end of the activity *drugsRequest* and also prompts the time observer process (via synchronization on port p_1) to start monitoring the elapsed time. The next state in line corresponds to the start of the *drugsChecking* activity that must be ful-

hal-00578436, version 2 - 21 Mar 2011

filled within 6–12 unit of times unless the time observer enter its error state `err` (synchronization with the temporal observer on port p_2). At this point, the `deliverySequence` synchronizes on the "start" port of the `<flow>` process (port $\mathtt{p}_3$ in Fig. 2). After the completion of the `<flow>` process (synchronization on port $\mathtt{p}_4$) the `deliverySequence` gains hand again and the computation moves to the fulfillment of the reply activity *sendDrugsOrder*. Before concluding, the process interact again with the time observer (synchronizarion on port p_5) if the delay for sending the message $\mathtt{msg}_4$ comply with the global constraint (c_3).

5 Experimentation

We have developed a prototype implementation of a compiler from annotated BPEL processes to Fiacre in Java, named Bpel2fcr, that is based on the interpretation defined in the previous section. The architecture of our transformation is depicted in Fig. 3. The input of our prototype is a set of BPEL processes, their corresponding WSDL and Service Level Agreement contracts that list the local and global constraints of the timed choreography. Our tool relies on EasyBPEL (http://easybpel.petalslink.org/), a library that provides a BPEL 2.0 engine to orchestrate services.

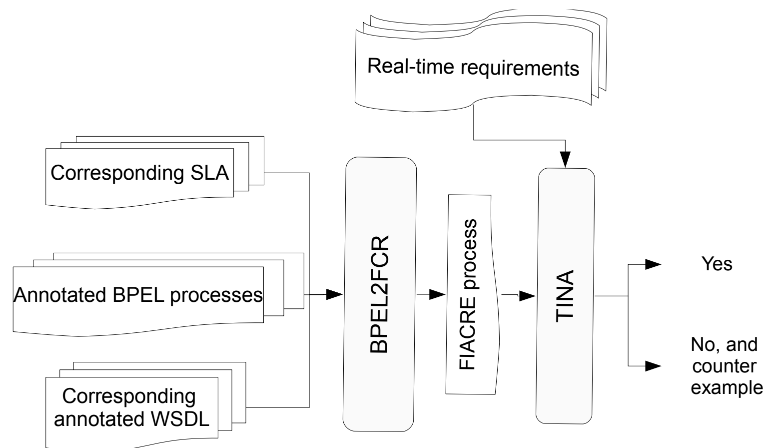


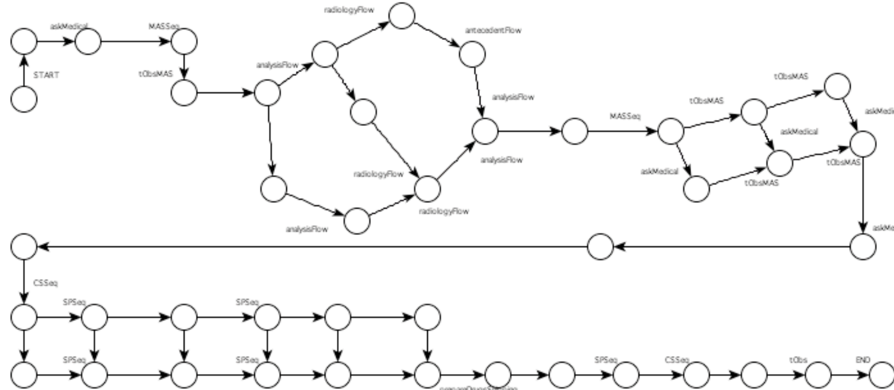
Fig. 3. Bpel2Fcr: architecture of the transformation

The Fiacre specification obtained as an output of Bpel2Fcr can be used with model-checking tools provided by the Tina verification toolbox. Model-checking is used to analyze the compatibility of the choreography as well as to check real-time requirements expressed by the users.

We give some results obtained with the analysis of our running example. The state graph for the HealthCare example, displayed in the diagram below, has only 43 states and 51 transitions. This is obtained using our optimized encoding (the graph for the unoptimized encoding is about three times as big). The generation

hal-00578436, version 2 - 21 Mar 2011

of the Fiacre specification and its corresponding state space takes less than a second. For examples of this size, the verification time for checking a requirement is negligible; in the order of a couple of milliseconds.



This model is very small due to the almost lack of concurrency in the scenario; only two `<flow>` activities and three services. For more complex examples, we have tested our tools on an extended version of the healthcare scenario with seven different services (see [14] for more details). The resulting example has 886 states and 2476 transitions. With this larger example of timed choreography, the verification of simple requirements takes in the order of half a second. As an example, we give the time and the validity of some real-time properties:

Property	*Result*	*Time (s)*
MCS.*init* `leadsto` MCS.*end* `within` [0,60]	true	0,638
MCS.*init* `leadsto` MCS.*end* `within` [0,20]	false	0,645
MAS.*end* `leadsto` MCS.*end* `within` [0,30]	true	0,531
PS.*end* `leadsto` MCS.*end* `within` [0,10]	false	0,637

6 Conclusion

We describe a new framework for modeling and analyzing timed choreographies obtained through the composition of annotated BPEL processes. We most particularly focus on the problem of checking real-time requirements on a choreography. In this context, we have proposed a rich formal model for timed services composition that captures efficiently several kind of temporal constraints associated to the concurrent nature of services. The framework we propose has been implemented into a tool that automatically translates timed BPEL processes into a Fiacre specification. In addition, we have shown an associated model-based verification approach to check real-time requirements on timed choreographies. In this context, we have defined classes of new real-time requirements which are specified using a logical-based formalism.

hal-00578436, version 2 - 21 Mar 2011

Work is still ongoing to improve and optimize the transformation implemented in our verification toolchain. We also have plans for future work. In particular, our goal is to extend our approach by checking more complex requirements that relate to automatic reconfiguration of service composition. We should be assisted in that by the fact that our interpretation is compositional.

References

[1] Nouha Abid et al. *The Fiacre Real-Time Specification Patterns Language*. Tech. rep. LAAS, 2010.

[2] A. Airkin et al. "Web Service Choreography Interface (WSCI) 1.0". In: *W3C Working Group* (2002).

[3] B Benatallah, F Casati, and F Toumani. "Analysis and Management of Web Service Protocols". In: *23rd International Conference on Conceptual Modeling* (2004).

[4] Boualem Benatallah, Fabio Casati, and Farouk Toumani. "Web Service Conversation Modeling: A Cornerstone for E-Business Automation". In: *IEEE Internet Computing* 8.1 (2004), pp. 46–54.

[5] Boualem Benatallah et al. "On Temporal Abstractions of Web Service Protocols". In: *The 17th Conference on Advanced Information Systems Engineering (CAiSE '05). Short Paper Proceedings*. 2005.

[6] Bernard Berthomieu, Pierre O Ribet, and François Vernadat. "The tool TINA – Construction of Abstract State Spaces for Petri Nets and Time Petri Nets". In: *International Journal of Production Research* 42.10 (2004).

[7] Bernard Berthomieu et al. "Fiacre: an intermediate language for model verification in the TOPCASED environment". In: *Embedded Real Time Software (ERTS)*. 2008.

[8] Bernard Berthomieu et al. "The syntax and semantics of FIACRE". In: *Repport LAAS N 07264* (2007).

[9] Lucas Bordeaux et al. "When are Two Web Services Compatible?" In: *Technologies for E-Services, 5th International Workshop (TES)*. 2004, pp. 15–28.

[10] Marius Dorel Bozga, Vassiliki Sfyrla, and Joseph Sifakis. "Modeling synchronous systems in BIP". In: *Proceedings of the seventh ACM international conference on Embedded software*. EMSOFT '09. ACM, 2009.

[11] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. "Patterns in Property Specifications for Finite-State Verification". In: *International Conference on Software Engineering (ICSE'99)*. 1999, pp. 411–420.

[12] Johann Eder and Amirreza Tahamtan. "Temporal Conformance of Federated Choreographies". In: *Proceedings of the 19th International Conference on Database and Expert Systems Applications(DEXA'08)*. Turin, Italy, September 1-5, 2008.

[13] Patrick Farail et al. "the TOPCASED project: a Toolkit in OPen source for Critical Aeronautic SystEms Design". In: *Embedded Real Time Software (ERTS)*. 2006.

[14] Nawal Guermouche and Silvano Dal Zilio. "Formal Requirement Verification for Timed Choreographies". In: *http://homepages.laas.fr/nguermou/sources/BPMExtendedVersion.pdf* (2011).

hal-00578436, version 2 - 21 Mar 2011

[15] Nawal Guermouche and Claude Godart. "Asynchronous Timed Web Service-Aware Choreography Analysis". In: *Proceedings of the* 21st *International Conference on Advanced Information Systems Engineering (CAiSE'09).* Amsterdam, The Netherlands, June 8-12, 2009. Pp. 364–378.

[16] Nawal Guermouche and Claude Godart. "Timed Conversational Protocol based Approach for Web services Analysis". In: *Proceedings of the* 8th *International Conference on Service Oriented Computing (ICSOC'10).* San Francisco, California, USA, December 7–10, 2010.

[17] Nawal Guermouche and Claude Godart. "Timed Model Checking Based Approach for Web Services Analysis". In: *IEEE International Conference on Web Services (ICWS'09).* Los Angeles, CA, USA, 6-10 July 2009, pp. 213–221.

[18] Pierre-Cyrille Héam, Olga Kouchnarenko, and Jérôme Voinot. "Component simulation-based substitutivity managing QoS and composition issues". In: *Sci. Comput. Program.* 75.10 (2010), pp. 898–917.

[19] Pierre-Cyrille Héam, Olga Kouchnarenko, and Jérôme Voinot. "How to Handle QoS Aspects in Web Services Substitutivity Verification". In: *WETICE'07 – IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises.* 2007, pp. 333–338.

[20] Raman Kazhamiakin, Paritosh K. Pandya, and Marco Pistore. "Representation, Verification, and Computation of Timed Properties in Web Service Compositions". In: *Proceedings of the IEEE International Conference on Web Services (ICWS).* 2006, pp. 497–504.

[21] Raman Kazhamiakin, Paritosh K. Pandya, and Marco Pistore. "Timed Modelling and Analysis in Web Service Compositions". In: *Proceedings of the The First International Conference on Availability, Reliability and Security, ARES.* IEEE Computer Society, 2006, pp. 840–846.

[22] P. M. Merlin and D. J. Farber. "Recoverability of Communication Protocols: Implications of a Theoretical Study." In: *IEEE Trans. Comm.* 24.9 (1976).

[23] "OASIS. Web Services Business Process Execution Language Version 2.0." In: *http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html* (April, 2007).

[24] Julien Ponge. "A New Model For Web services Timed Business Protocols". In: *Atelier (Conception des systèmes d'information et services Web) SIWS-Inforsid).* 2006.

[25] Julien Ponge et al. "Fine-Grained Compatibility and Replaceability Analysis of Timed Web service Protocols". In: *the 26th International Conference on Conceptual Modeling (ER).* 2007.

[26] W3C. "Web Service Description Language (WSDL)". In: *http ://www.w3.org/TR/WSDL/* (2001).