Asynchronous Timed Web Service-Aware Choreography Analysis

Nawal Guermouche and Claude Godart

LORIA-INRIA-UMR 7503
F-54506 Vandoeuvre-les-Nancy, France
{Nawal.Guermouche,Claude.Godart}@loria.fr

Abstract. Web services are the main pillar of the Service Oriented Computing (SOC) paradigm which enables applications integration within and across business organizations. One of the important features of the Web services is the *choreography* aspect which allows to capture collaborative processes involving multiple services. In this context, one of the important investigations is the *choreography compatibility analysis*. We mean by the choreography compatibility the capability of a set of Web services of actually interacting by exchanging messages in a proper manner. Whether a set of services are compatible depends not only on their sequences of messages but also on *quantitative properties* such as *timed properties*. In this paper, we investigate an approach that deals with checking the *timed compatibility of a choreography* in which the Web services support *asynchronous timed communications*.

Keywords: Web service, Timed properties, Asynchronous timed Compatibility analysis.

1 Introduction

The evolution of computer science technologies gives life to many paradigms such as the Service Oriented Computing (SOC) paradigm. In this latter, Web services are the main pillar. Based on standard interfaces, Web services facilitate application-to-application interaction. This advantageous property gives rise to several important concepts such as the choreography concept. This feature offers the possibility to capture collaborative processes involving multiple services where the interactions between these services are seen from a global perspective. In this context, one of the important elements is the compatibility analysis. By compatibility we mean the capability of a set of services of actually fulfilling successful interactions by exchanging messages.

In the last few years, few works have investigated the compatibility problem of a client and a provider service [4,2,13,9,7]. In all these works, the authors deal with services that support *synchronous communications*. In that case, to characterize the compatibility class of two services, the authors check if each input (resp. output) message of a service corresponds to an output (resp. input)

P. van Eck, J. Gordijn, and R. Wieringa (Eds.): CAiSE 2009, LNCS 5565, pp. 364–378, 2009.

message of the other service in the same order (i.e., the services are synchronized over messages). However, the nature of distributed systems and particularly the Web services could be asynchronous, hence the problem of the applicability of these approaches in real application scenarios is still open. To overcome such limitations, in this paper, we tackle the problem of analyzing the compatibility of a choreography in which the Web services support *asynchronous* communications. In an asynchronous communication, when a message is sent, it is inserted to a message queue, and the receiver can consume it later from the queue.

It is commonly agreed that in general the interaction of Web services and in particular the compatibility of Web services depend not only on the supported sequences of messages but also on crucial *quantitative properties* such as *timed properties* [2,11,10,13,9,7,8]. We mean by *timed properties* the necessary delays to exchange messages (e.g., in an e-government system to manage handicapped pension requests, a prefecture must send its final decision to grant an handicapped pension to a requester after 168 hours and within 336 hours). There are some works that tried to consider timed properties when analyzing the compatibility of two synchronous services [2,13,9,7]. However, dealing only with synchronous services decreases considerably the feasibility and the applicability of these approaches.

In this paper, we propose a framework for analyzing the choreography compatibility. This framework supports asynchronous communicating services. In our framework we take into account data flow that can be involved when exchanging messages. Furthermore, we consider timed properties that specify the necessary delays to exchange messages. By studying the possible impacts of timed properties on a choreography, we remark that when the Web services are interacting together, *implicit timed dependencies* could be built between the different timed properties of the different services. Such dependencies could give rise to *implicit timed conflicts*. To discover such timed conflicts, we first study the possibility to apply the proposed compatibility approaches of synchronous services [2,13,9,7], and we have remarked that the existing approaches are inadequate to discover all the eventual timed conflicts since the authors rely on synchronizing the services over messages. In order to catch all the possible timed conflicts, in this paper we rely on the *clock ordering process* we have proposed in some earlier work on Web service composition [8]. The clock ordering process aims at making explicit the eventual implicit timed conflicts when services are interacting together.

To summarize, in this paper we make the following contributions: (1) we propose an asynchronous model of Web services that takes into account messages, data types and timed requirements. (2) unlike existing compatibility frameworks, we propose primitives for analyzing and characterizing the compatibility class of a choreography in which the services support *asynchronous timed communications*.

The reminder of the paper is organized as follows. Section 2 presents the e-government case study that we use to show the related issues of the proposed approach. In Section 3 we present how we model the timed behavior of

Web services. For better understanding, in section 4, we discuss informally and intuitively the timed compatibility problem of a choreography. Section 5 presents the formal choreography compatibility investigation we propose. An illustrative example using the e-government scenario is given in Section 6. In Section 7, we discuss related work. Finally section 8 concludes.

2 Case Study: e-Government Application

Let us present a part of an e-government application that we use at the end to illustrate our approach. The goal of the e-government application we consider is to manage handicapped pension request. Such a request involves three Web services: (1) *requester* service (*RS*), (2) *health authority* (*HS*) service, and (3) *prefecture* service (*PS*). The high level choreography model of the process is depicted on Fig. 1. To grant an handicapped pension to a requester, the process can be briefly summarized as follows. (1) via a requester service, a citizen deposits a file in the prefecture, (2) the citizen requests a medical file from the health authority service. (3) The health authority negotiates a date for an appointment to examine the citizen. (4) after the examination, the health authority service sends a medical report to the prefecture. (5) after studying the received file and the medical report, the prefecture sends the notification of the final decision to the citizen.

The interaction between these partners is constrained by timed properties. Below, we give some timed requirements.

- Once the health authority service proposes meeting dates to the citizen, the health authority service must receive the filled form within 24 hours.
- The prefecture requires at least 168 hours and at most 336 hours since receiving the file from the requester to notify the citizen with the final decision.
- Via the requester service, once the citizen obtains the medical form, he must send the filled form within 36 hours.

The Web services we consider could support asynchronous communications. The first issue we deal with is how to analyze the compatibility of a choreography in which the Web services are asynchronous? Moreover, the behavior of the Web services might be constrained by timed requirements. In order to manage

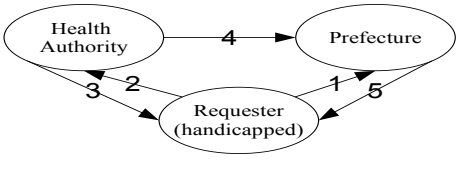

Fig. 1. Global view of the e-government application

the global interaction between the Web services (i.e., to ensure that the choreography is deadlock free), we need primitives that consider timed properties when analyzing the compatibility of Web services. Thus, the second issue we handle is how to consider timed properties when analyzing the compatibility of asynchronous services in a choreography?

3 Modeling Timed Behavior of Web Services

One of the important ingredient in a compatibility framework is the *timed conversational protocol* of Web services. In our framework, the timed conversational protocol specifies the sequences of messages a service supports, the involved data flow and the associated timed properties to exchange messages. We adopt a finite state machine based formalism to model the timed behavior of Web services (i.e., the timed conversational protocol). Intuitively, the states represent the different phases a service may go through during its interaction. Transitions enable sending or receiving a message. An output message is denoted by $!m$, whilst an input one is denoted by $?m$. A message involving a list of data types is denoted by $m(d_1, \ldots, d_n)$, or $m(\bar{d})$ for short. To capture the timed properties when modeling Web services, we propose to use the standard timed automata clocks [1]. The automata are equipped with a set of clocks. The values of these clocks increase with the passing of time. Transitions are labeled by timed constraints, called *guards*, and resets of clocks. The former represent simple conditions over clocks, and the latter are used to reset values of certain clocks to zero. The guards specify that a transition can be fired if the corresponding guards are satisfiable.

Let X be a set of clocks. The set of *constraints* over X, denoted $\Psi(X)$, is defined as follows:

true $\mid x \bowtie c \mid \psi_1 \wedge \psi_2$, where $\bowtie \in \{\leq, <, =, \neq, >, \geq\}$, $x \in X$, $\psi_1, \psi_2 \in \Psi(X)$, and c is a constant.

Definition 1. *(Timed conversational protocol)*
A timed conversational protocol of a Web service Q is a tuple (S, s_0, F, M, X, T) such that:

S is a set of states, s_0 is the initial state ($s_0 \in S$), F is the set of final states ($F \subseteq S$), M is a set of messages, X is the set of clocks, T is a set of transitions such that $T \subseteq S \times M \times \Psi(X) \times 2^X \times S$, with an exchanged message that involves data types ($?m(\bar{d})$: input message, $!m(\bar{d})$:output message), a guard over clocks, and the clocks to be reset.

A transition *(s, a, ψ, Y, s') is denoted by $s \rightarrow^{a}_{\psi, Y} s'$.*

A trace *is a sequence of transitions leading to a final state, denoted as follows: $s_0 \rightarrow^{\alpha_0}_{\psi_0, Y_0} s_1 \rightarrow^{\alpha_1}_{\psi_1, Y_1} \ldots \rightarrow^{\alpha_{n-1}}_{\psi_{n-1}, Y_{n-1}} s_n$ where s_n is a final state.*

The semantic of the former is defined using a transition relation over configurations made of a state and a clock valuation. The clock valuation is a mapping $u : X \rightarrow \mathbb{T}$ from a set of clocks to the domain of timed values. The mapping u_0

denotes the (initial) clock valuation, such that $\forall x \in X, u_0(x) = 0$. Initially, the queue of the services are empty.

A service remains in the same state s without triggering a transition, when the time increments, if there is no transition $(s, \alpha, \psi_X, Y, s')$ such that the timed constraints ψ_X are satisfied, where $\psi_X \subseteq \Psi(X)$ and α is either an output message $!m(\overline{d})$ or an input message $?m(\overline{d})$ which is available in the queue. In an asynchronous communication, when a message is sent, it is inserted to a message queue, and the receiver consumes (i.e. receives) the message while it is available in the queue.

Definition 2. *(Semantic of timed conversational protocol)*
Let $P = (S, s_0, F, M, X, T)$ be a conversational protocol. The semantic is defined as a labeled transition $(\Gamma, \gamma_0, \rightarrow)$, where $\Gamma \subseteq S \times V_T$ is the set of configurations, such that V_T is a set of timed valuations, $\gamma_0 = (s_0, u_0)$ is the initial configuration, and $\rightarrow$ is defined as follows:

- Elapse of time*:* $(s, u) \overset{tick}{\rightarrow} (s, u + \delta)$
- Location switch*:* $(s, u) \overset{\alpha}{\rightarrow} (s', u')$*, if* $\exists t = (s, \alpha, \psi_X, Y, s')$ *such that* $u \wedge \psi_X$ *are satisfiable and* $\forall y \in Y, u'(y) = 0$, $\forall x \in X \backslash Y, u'(x) = u(x)$*, where* $Y \subseteq X$*, and*
  - *If* $\alpha = !m(\overline{d})$, $Que := Que + m(\overline{d})$
  - *If* $\alpha = ?m(\overline{d})$, *and* $m(\overline{d}) \in Que$, $Que := Que - m(\overline{d})$

The timed conversational protocols of the three services introduced in Section 2 are depicted on Fig. 2. Next, we will present the intuition behind the choreography compatibility problem.

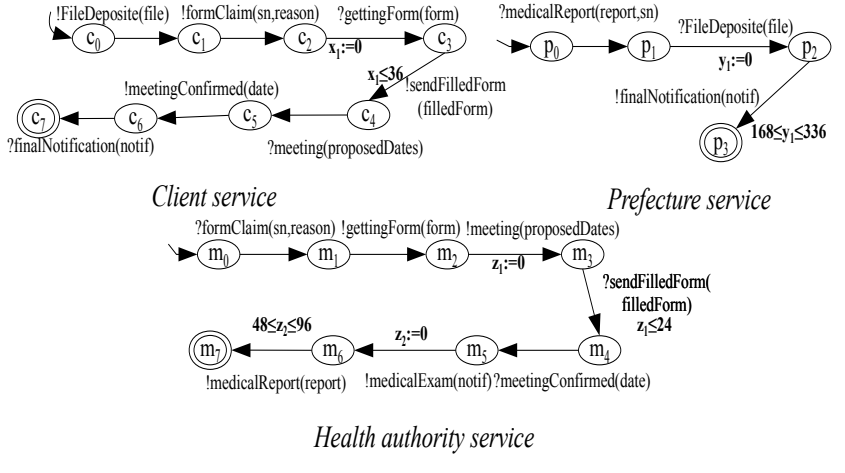

Fig. 2. Web services

4 Timed Compatibility Problem

In this section, by using examples, we discuss informally and intuitively the timed choreography compatibility problem and the related issues. Then, in the next section, we present the formal investigation we propose.

Example 1. Let us first consider the two *untimed* conversational protocols of the two services Q and Q' depicted on Fig. 3. In spite that both services cannot produce and consume their messages in the same order, the two asynchronous services are *fully compatible*. The service Q starts by sending the message $m_0(d_0, d_1)$ which becomes available in the queue of Q'. On the other side, Q' sends the message $m_2(d_3)$. After that, Q' consumes the message $m_0(d_0, d_1)$ then it sends the message $m_1(d_2)$. Once the message $m_1(d_2)$ is sent, it is added to the queue of Q. Therefore, Q can consume the message $m_1(d_2)$ and then the message $m_2(d_3)$. By using the existing work, these two services are considered as incompatible although they can succeed an execution. In fact, the proposed frameworks (e.g., see [4,2,13,9,7]) deal only with synchronous communicating services.

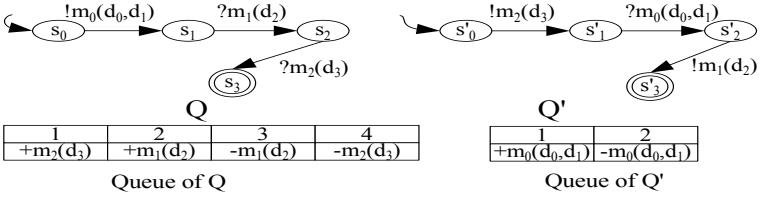

Fig. 3. Untimed asynchronous Web services

Augmenting the conversational protocol of asynchronous services by timed properties lays important challenges. Particularly, *the clocks used to define timed properties are local and mutually independent.* At the same time, in our work, we do not assume that the timed properties are synchronized over messages. Consequently, when services interact together, implicit timed conflicts can arise. To illustrate this issue, in the following we present an illustrative example.

Example 2. Let us consider the two timed conversational protocols of the two services Q and Q' depicted on Fig. 4. The service Q starts by sending the message $m_0(d_0, d_1)$. So this latter becomes available in the queue of Q'. On the other hand, Q' can send the message $m_2(d_3)$ that can be stored in the queue of Q. The service Q remains blocked, since the message $m_1(d_2)$ is not yet available. But Q' can consume the message $m_0(d_0, d_1)$ which has been already sent by Q. Once consumed, Q' sends the message $m_1(d_2)$ after 20 and within 40 units of time from consuming the message $m_0(d_0, d_1)$ (i.e., $20 \leq x \leq 40$). Consequently, the message $m_1(d_2)$ becomes available in the queue of Q after 20 units of time from consuming the message $m_0(d_0, d_1)$. In that case, Q will be able to consume

the message $m_1(d_2)$ after 20 units of time. Finally, Q must consume the message $m_2(d_3)$ within 10 units of time. However, this message can be consumed only after consuming the message $m_1(d_2)$, i.e., after 20 units of time. In fact, the message $m_1(d_2)$ can be sent (becomes available) by Q' after 20 units of time. So, the message $m_2(d_2)$ must be consumed within 10 units of timed and at the same time it is possible to consume it after 20 units of time which represents a timed conflict.

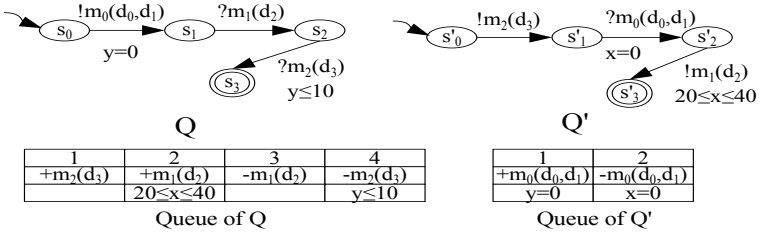

Fig. 4. Incompatible timed asynchronous services

In order to catch the eventual timed conflicts in a choreography, we propose the formal approach described in the following section.

5 Formal Compatibility Analysis

In the previous section, we have shown the need of formal primitives of analyzing the timed choreography compatibility of asynchronous services. In general a compatibility framework should be able to characterize the compatibility class of Web services. But in addition, in case of compatibility, it is quite important to characterize the deadlock free interaction schema of a choreography. When the different services are interacting together, timed dependencies could be created between their different timed properties. Therefore, we need mechanisms that allow to catch the eventual implicit timed conflicts. In the following sections we present respectively how we compute the interaction schema of a set of Web services and how to discover the eventual timed conflicts when computing this schema.

5.1 Building the Timed Choreography Interaction Schema

By having the set of conversational protocols of the involved services, our aim is to build a global timed conversational protocol that specifies an executable Timed Choreography Interaction Schema (TCIS). In order to build this TCIS, we introduce the concept of configuration that represents the states of the TCIS at a given time. A configuration defines the evolution of the services states when they are interacting together. In the initial configuration, all the services are

in their initial states. Given a source configuration, the TCIS reaches a new configuration when there exists one service that changes its state by exchanging a message so that no timed conflicts arise. The process of discovering the eventual implicit timed conflicts is presented in Section 5.2.

Definition 3. *(Timed Choreography Interaction Schema (TCIS))*
Let $Q_i(S_i, s_{0_i}, F_i, M_i, X_i, T_i)$ *to be a set of Web services for* $i = \{1, \ldots, n\}$. *The Timed Choreography Interaction Schema TCIS of* Q_i *is defined as a tuple* (S, s_0, F, M, X, T) *such that*
$S = S_1 \times \ldots \times S_n$, $s_0 = s_{0_1} \times \ldots \times s_{0_n}$, $F = F_0 \times \ldots \times F_n$, $M = M_0 \cup \ldots \cup M_n$, $X = X_0 \cup \ldots \cup X_n$, $T \subseteq S \times M \times \Psi(X) \times 2^X \times S$ *is defined as follows:*
- $(s_1 \ldots s_i \ldots s_n, m(\overline{d}), \psi'_X, Y, s_1 \ldots s'_i \ldots s_n) \in T$ *if* $(s_i, m(\overline{d}), \psi_X, Y, s'_i) \in T_i$

When we build a TCIS, we simulate the transactions of the queues of the services by using one queue. By using the built TCIS, we can characterize each queue transaction of each service. In order to build a TCIS, we propose the algorithm 1.

Algorithm 1: TCIS_Computing

Input: A set of Web services $Q_i = (S_i, s_{0_i}, F_i, M_i, X_i, T_i)$, for $i = \{1, .., n\}$. Empty queue *Que*.
Output: TCIS $= (S, s_0, F, M, X, T)$
begin
  $computedTr = T_1 \times \ldots \times T_n$ for $i = \{1, .., n\}$
  while $computedTr \neq \emptyset$ **do**
    incompatibility=false
    $currentTr = \{t_1, \ldots, t_n\}$ where $t_i \in T_i$ and $\{t_1, \ldots, t_n\} \in computedTr$
    $computedTr = computedTr - currentTr$
    for *each transition* $(s_i, m(\overline{d}), \psi_i, Y_i, s'_i)$ *of each trace* $t_i \in currentTr$ **do**
      if *Cycle_ Checked*$((s_i, m(\overline{d}), \psi_i, Y_i, s'_i))$ **then**
        /*If the message m is an input message, then $polarity(m) =?$ else $polarity(m) =!$*/
        if $(polarity(m) =' !')$ *or* $(polarity(m) =' ?'$ *and* $m(\overline{d}) \in Que\ )$ **then**
          $t_{candidate} = (s_1 \ldots s_i \ldots s_n, m(\overline{d}), \psi, Y, s_1 \ldots s'_i \ldots s_n)$
          if *Clock_ Order*$(t_{candidate})$ **then**
            $T = T \cup t_{candidate}$
            if $polarity m(\overline{d}) =!$ **then**
              $Que = Que + m(\overline{d})$
            else
              $Que = Que - m(\overline{d})$
          else
            there is a timed conflict, $t_{candidate}$ is not accepted,incompatibility=true
        else
          if $polarity(m) =' ?'$ *and* $m(\overline{d}) \notin Que$ **then**
            The current message is not yet available. We choose another transition of another service.
      else
        The current message is not yet available. We choose another transition of another service.
    if *not incompatibility* **then**
      /*there are good traces*/
      if $Que \neq \emptyset$ **then**
        There is an extra message. The current traces combination of the services are not compatible
      else
        The current traces combination of the services are compatible
    else
      The current traces combination of the services are not compatible
end

In the worse case, the algorithm 1 is exponential in time. In fact, in order to check if a set of services are compatible, the cartesian product of the services traces could be parsed.

The algorithm 1 considers cycles when analyzing the compatibility of a choreography thanks to the algorithm 2.

Algorithm 2 : Cycle_Checked

```
Input: a transition (s, m(d̄), ψ, Y, s')
Output: boolean
begin
    if s' is already visited then
        if polarity(m)=! then
            /*The message could be sent infinitely*/
            mark the message m(d̄)
            return true
        else
            if m(d̄) ∈ Queue and m(d̄) is marked then
                return true
            else
                return false
    else
        return true
end
```

5.2 Making Explicit the Implicit Timed Constraints Dependencies

In order to make explicit the dependencies between the timed properties when building the TCIS, we use the *clock ordering process* we proposed in our previous work on Web service composition directed by client data [8]. The clock ordering process aims at defining an order between the different clocks of the different services when they are interacting together. The idea behind the *clock ordering* process is to define a total order between the different clocks of the services for each new TCSA transition. To explain the idea behind the clock ordering process, we use the following example.

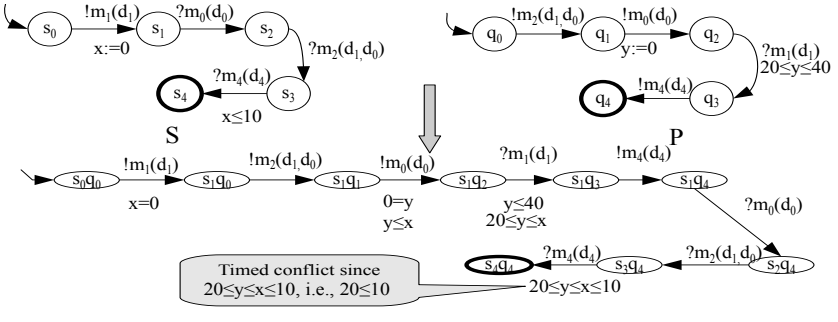

Fig. 5. Making explicit the implicit timed conflicts

Example 3. Let us consider the two timed conversational protocols of the services S and P depicted on Fig. 5. The service S can send the message $m_1(d_1)$ and resets the clock x. So we build the TCIS transition $(s_0q_0, !m_1(d_1), x = 0, s_1q_0)$. Then, P sends the message $m_2(d_1, d_0)$. We build the TCIS transition $(s_1q_0, !m_2(d_1, d_0), s_1q_1)$. After that, P sends the message $m_0(d_0)$ and resets the clock y. Since the clock x is reset before the clock y, hence we can define the order $y \leq x$. We build the corresponding TCIS transition $(s_1q_1, !m_0(d_0), y = 0, y \leq x, s_1q_2)$. As the message $m_1(d_1)$ has been already sent by S, so P can consume it so that $20 \leq y \leq 40$. By propagating the order $y \leq x$ defined above, we built the TCIS transition $(s_1q_2, ?m_1(d_1), 20 \leq y \leq x, y \leq 40, s_1q_3)$. Once the message $m_1(d_1)$ is consumed, P sends the message $m_4(d_4)$. On the other side, S consumes the message $m_0(d_0)$ that has been already sent by P. We build the $TCIS$ transition $(s_1q_4, ?m_0(d_0), s_2q_4)$. Then S consumes the message $m_2(d_1, d_0)$ that has been already sent by P. We build the TCIS transition $(s_2q_4, ?m_2(d_1, d_0), s_3q_4)$. Finally, S must consume the message $m_4(d_4)$ within 10 units of time from sending the message $m_1(d_1)$. By propagating the order $20 \leq y \leq x$ we defined above, we build the TCIS transition $(s_3q_4, ?m_4(d_4), 20 \leq y \leq x \leq 10, s_4q_4)$. The order $20 \leq y \leq x \leq 10$ presents a timed conflict, i.e., $20 \leq 10$ and it is not possible to fire the transition $(s_3q_4, ?m_4(d_4), 20 \leq y \leq x \leq 10, s_4q_4)$ (i.e., the message $m_2(d_3)$ cannot be consumed). As remarked, without the timed propagation process, the timed conflict could not be detected.

In order to define the clock ordering process, we are using the algorithm 3.

Algorithm 3: Clock_Order

```
Input: a transition (s_i, m_i(d̄), ψ_i, Y_i, s'_i)
Output: boolean
begin
    if s_i is the initial state then
        return true
    else
        /*propagation of the constraints of the form x ≥ v (resp. x > v) of a predecessor transition over the
        current transition*/
        for each ϱ_{i-1} ∈ ψ_{i-1}, such that ϱ_{i-1} = x ≥ v or ϱ_{i-1} = x > v of (s_{i-1}, m_{i-1}(d̄), ψ_{i-1}, Y_{i-1}, s'_{i-1}) do
            ψ_i = ψ_i ∪ ϱ_{i-1}
            /*The value of the clocks reset in the current transition is smaller than the value of a clock reset in
            the predecessor transition */
            for each y = 0 ∈ Y_i and z = 0 ∈ Y_{i-1} do
                ψ_i = ψ_i ∪ y ≤ z
            /*We propagate the order defined in the predecessor transition over the current transition*/
            for each z_1 ≤ z_2 ∈ ψ_{i-1} do
                ψ_i = ψ_i ∪ z_1 ≤ z_2
        if ∃ v ≤ y_0 ≤ ... ≤ y_n ≤ v' ∈ ψ_i where v' < v then
            return false
        else
            return true
end
```

5.3 Characterization of Compatibility Classes

Previously, we have shown how we can build a deadlock free TCIS. In this section, we present how we can characterize the compatibility class of a set of asynchronous Web services. Before that, let us present the *subsumption* and *crossing* relations of protocols.

We say that a protocol Q_i is subsumed by a given TCIS if each transition of each trace of the protocol Q_i belongs to the given TCIS. In the context of our work, this means that for each transition $(s_i, \alpha_i, \psi_i, Y_i, s'_i)$ of Q_i, there exists a transition $(s_1 \dots s_i \dots s_n, \alpha'_i, \psi'_i, Y_i, s_1 \dots s'_i \dots s_n)$ of TCIS which can be preceded by a sequence of messages.

Definition 4. *(Protocol subsumption $\subseteq_{tcis}$)*
Let $TCIS = (S, s_0, F, M, X, T)$ *be a computed TCIS and* $Q_i = (S_i, s_{0_i}, F_i, M_i, X_i, T_i)$ *be a protocol of a Web service. We say that the TCIS subsumes* Q_i*, denoted* $Q_i \subseteq_{tcis} TCIS$ *if for each trace* $s_{0_i} \rightarrow^{\alpha_0}_{\psi_0, Y_0} s_{1_i} \rightarrow^{\alpha_1}_{\psi_1, Y_1} \dots s_{n-1_i} \rightarrow^{\alpha_{n-1}}_{\psi_{n-1}, Y_{n-1}} s_{n_i}$ *in* Q_i*, there exists a trace t:* $(s_0 \dots s_n) \stackrel{\vartheta_0}{\rightarrow} (s_0, \dots s_{0_i} \dots s_n) \rightarrow^{\alpha_0}_{\psi'_0, Y_0} (s_0, \dots s_{1_i} \dots s_n) \dots \rightarrow (s_0, \dots s_n) \stackrel{\vartheta_{n-1}}{\rightarrow} (s_0, \dots s_{n-1_i} \dots s_n) \rightarrow^{\alpha_{n-1}}_{\psi'_{n-1}, Y_{n-1}} (s_0, \dots s_{n_i} \dots s_n) \stackrel{\vartheta_n}{\rightarrow} (s_0, \dots s_n)$ *of* $TCIS$ *such that for* $i = 0, \dots, n$, $(s_0 \dots s_n) \stackrel{\vartheta_i}{\rightarrow} (s_0 \dots s_n)$ *is a (possibly empty) message sequence of* $TCIS$*.*

We say that a protocol Q_i *crosses* a given TCIS if there is at least one trace of Q_i which is subsumed by this TCIS. Next, we present the formal definition of the *crossing* relation.

Definition 5. *(Protocol crossing $\cap_{tcis}$)*
Let $TCIS = (S, s_0, F, M, X, T)$ *be a computed TCIS and* $Q_i = (S_i, s_{0_i}, F_i, M_i, X_i, T_i)$ *be a protocol of a Web service. We say that* Q_i ***crosses*** *the TCIS, denoted* $Q_i \cap_{tcis} TCIS$ *if* $\exists\ s_{0_i} \rightarrow^{\alpha_0}_{\psi_0, Y_0} s_{1_i} \rightarrow^{\alpha_1}_{\psi_1, Y_1} \dots s_{n-1_i} \rightarrow^{\alpha_{n-1}}_{\psi_{n-1}, Y_{n-1}} s_{n_i}$ *in* Q_i*, such that there exists a trace t:* $(s_0 \dots s_n) \stackrel{\vartheta_0}{\rightarrow} (s_0, \dots s_{0_i} \dots s_n) \rightarrow^{\alpha_0}_{\psi'_0, Y_0} (s_0, \dots s_{1_i} \dots s_n) \dots \rightarrow (s_0, \dots s_n) \stackrel{\vartheta_{n-1}}{\rightarrow} (s_0, \dots s_{n-1_i} \dots s_n) \rightarrow^{\alpha_{n-1}}_{\psi'_{n-1}, Y_{n-1}} (s_0, \dots s_{n_i} \dots s_n) \stackrel{\vartheta_n}{\rightarrow} (s_0, \dots s_n)$ *of* $TCIS$ *such that for* $i = 0, \dots, n$, $(s_0 \dots s_n) \stackrel{\vartheta_i}{\rightarrow} (s_0 \dots s_n)$ *is a (possibly empty) message sequence of* $TCIS$*.*

We say that a set of Web services constitutes a *full compatible choreography* if each protocol of each service is subsumed by the TCIS. However, when there are some traces that are subsumed by the TCIS and there are some traces that are not subsumed by the TCIS, we say that the set of Web services constitutes a *partial compatible choreography*. But, when the TCIS is an empty protocol, thus we say that the set of Web services constitutes a *full incompatible choreography*.

Definition 6. *(Choreography compatibility classes)*
Let $TCIS = (S, s_0, F, M, X, T)$ *be a computed TCIS of a set of Web services* $Q_i = (S_i, s_{0_i}, F_i, M_i, X_i, T_i)$ *for* $i \in \{1, .., n\}$

- *A set of Web services Q_i for $i = \{1, .., n\}$ are said to be fully compatible if* $\forall i \in \{1, .., n\}$, $Q_i \subseteq_{tcis} TCIS$
- *A set of Web services Q_i are said to be partially compatible if* $\exists i \in \{1, .., n\}$, $Q_i \nsubseteq_{tcis} Q$ *and* $Q_i \cap_{tcis} TCIS$
- *A set of Web services Q_i are said to be fully incompatible if* $TCIS = \emptyset$.

6 Illustrative Example

By using the e-pension application we introduced in Section 2, let us now present an illustrative example of how analyzing the compatibility of the corresponding choreography. Initially, the three services client service (CS), prefecture service (PS) and health authority service (HS) are in their initial states. That means, the first $TCIS$ configuration is $(c_0p_0m_0)$. From this configuration, CS enables the transition $(c_0, !FileDeposite(file), c_1)$. We build the $TCIS$ transition $(c_0p_0m_0, !FileDeposite(file), c_1p_0m_0)$. From the configuration $c_1p_0m_0$, CS enables the transition $(c_1, !formClaim(sn, reason), c_2)$. We build the $TCIS$ transition $(c_1p_0m_0, !formClaim(sn, reason), c_2p_0m_0)$. HS can consume the message $formClaim(sn, reason)$ which has been already sent by CS. Thus, we build the $TCIS$ transition $(c_2p_0m_0, ?formClaim(sn, reason), c_2p_0m_1)$. The new configuration becomes $c_2p_0m_1$. From this latter, HS enables the transition $(m_1, !gettingForm(form), m_2)$. We can build the $TCIS$ transition $(c_2p_0m_1, !gettingForm(form), c_2p_0m_2)$. After sending the message $gettingForm(form)$, HS sends the message $meeting(proposeDates)$ and resets a clock z_1. In that case, we build the $TCIS$ transition $(c_2p_0m_2, !meeting(proposeDates), z_1 = 0, c_2p_0m_3)$. From the new configuration $c_2p_0m_3$, CS can consume the available message $gettingForm(form)$ which is already sent by HS. Once consumed, the clock x is reset. Since the clock z_1 is reset before the clock x, hence we define the clock order $x \leq z_1$. Then, we build the $TCIS$ transition $(c_2p_0m_3, ?gettingForm$

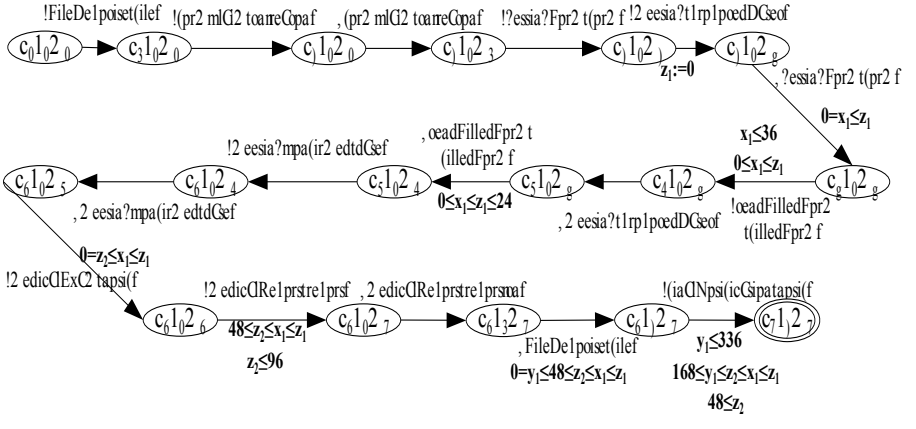

Fig. 6. TCIS of the e-pension application

$(for- \ m)$, $x = 0, x \leq z_1, c_3p_0m_3)$. After that, CS can send the message $sendFilled- \ Form(filledForm)$ within 36 units of time ($x \leq 36$). Regarding the clock order $x \leq z_1$ we have defined above, we build the $TCIS$ transition $(c_3p_0m_3, !sendFill- \ edForm(filledForm), 0 \leq x \leq z_1, x \leq 36, c_4p_0m_3)$.

When the configuration $c_5p_0m_3$ is reached, HS can consume the message $sendFilledForm(filledForm)$ which is already sent (regarding the built $TCIS$) by CS. HS can consume the message $sendFilledForm(filledForm)$ within 24 units of time ($z_1 \leq 24$). Regarding the order we have defined above, we define the order $0 \leq x \leq z_1 \leq 24$ that we associate to the $TCIS$ transition $(c_5p_0m_3, ?sendFilledForm(filledForm), 0 \leq x \leq z_1 \leq 24, c_5p_0m_4)$. By applying the same steps, we build the deadlock free TCIS depicted on Fig. 6.

According to this TCIS, the three services CS, PS, and HS are fully compatible, since each protocol of each service is subsumed by the built TCIS. For example, according to the trace $p_0 \rightarrow^{?medicalReport(report,sn)} p_1 \rightarrow^{?FileDeposite(file)}_{y_1=0} p_2 \rightarrow^{!finalNotification(notif)}_{168 \leq y_1 \leq 336} p_3$ of PS, we can remark that each transition belongs to the trace of the $TCIS$. For example, if we consider the transition $(p_0, ?medicalReport(report, sn), p1)$, we can see that from the TCIS initial configuration $c_0p_0m_0$, we can reach the configuration $c_6p_0m_7$ that allows to fire the transition $(c_6p_0m_7, ?medicalReport(\ report, sn), c_6p1m_7)$.

7 Related Work

Checking and analyzing in general the Web services features is an important investigation [4,3,2,5,12,11,10]. Particularly, in this paper we are interested in the compatibility analysis of a choreography in which the services support asynchronous communicating services. In general, the compatibility problem is based on analyzing message exchange sequences (conversations). In practice, other metrics affect the Web services compatibility, such as the kind of communication (synchronous or asynchronous) the services support. Besides, quantitative properties such as timed constraints plays a crucial role in Web services interaction.

In [4,3], the authors consider the sequence of messages that can be exchanged between two synchronous Web services. But, considering only message exchange sequences is not sufficient. To succeed a conversation, other metrics can have an impact such as timed properties which are not considered in [4,3]. Another important remark is that in [4,3], the authors consider synchronous Web services. Such assumption is very restrictive since the nature of Web services can be asynchronous. To overcome this limitation, we propose a compatibility checking approach for timed asynchronous services.

The compatibility framework presented in [12,13], that is an extension of the framework presented in [2], considers a more expressive timed constraints model. Although powerful, in some cases, the compatibility framework cannot detect some timed conflicts due to non-cancellation[1] constraints. In fact, the authors

[1] In [13] the non-cancellation constraints are called *C-Invoke*. They specify a time window within which a given message can be fired. Outside the window, the transition is disabled (exchanging the message results in an error).

deal only with synchronous communicating services. Thus, to discover timed conflicts, the authors are based on synchronizing the corresponding timed properties over messages. Therefore, this framework cannot be applied to discover the eventual timed conflicts in case of asynchronous Web services.

In [6], the authors handle the timed conformance problem which consists in checking if a given timed orchestration satisfies a global timed choreography. In this framework, the authors propose to deal with timed cost (i.e., the delay) of operations. According to our work, our aim is to detect conflicts that can arise when a set of Web services are interacting altogether. Whilst, [6] is not interested in analyzing the compatibility of a choreography but in checking if a given orchestration conforms to a choreography. So, one of the assumption is that the choreography does not hold timed conflicts.

We would like to mention that we are not using the techniques that have been proposed in the context of timed automata such as building region automata since the protocols of the services could be huge, consequently, building the structures such as region automata could be very complex and very huge. Moreover, in the context of our work such structure that gives rich information is not required. Whilst, by using the clock ordering process we are just defining an order between the different clocks in order to make explicit the eventual implicit timed conflicts.

8 Conclusion

In this paper, we presented a formal framework for analyzing the compatibility of a choreography. Unlike the proposed approaches, this framework caters for timed properties of asynchronous Web services. We presented how to model the timed behavior of Web services. To model timed properties, we propose to use the standard clocks of standard timed automata. In a choreography, when the services are interacting together, implicit timed dependencies can arise which could give rise to timed conflicts. We used the clock ordering process to discover such conflicts.

In a compatibility framework, it is important to characterize the executable interaction schema. To do so, we proposed an algorithm that allows to compute the timed choreography interaction schema of a set of Web services that can support asynchronous communications. We presented the *clock ordering* process that aims at discovering implicit timed conflict in a choreography. By using the mechanisms we proposed, we presented classes of timed choreography compatibility.

In our future work, we are interested in analyzing the compatibility of a choreography in which the instances of the involved services is not known in advance. Our aim is to provide primitives for defining dynamically the required instances for a successful choreography. Moreover, we plan to extend the proposed approach to support more complex timed properties when analyzing the compatibility of a set of Web services.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
2. Benatallah, B., Casati, F., Ponge, J., Toumani, F.: On temporal abstractions of web service protocols. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520. Springer, Heidelberg (2005)
3. Benatallah, B., Casati, F., Toumani, F.: Analysis and management of web service protocols. In: 23rd International Conference on Conceptual Modeling (November 2004)
4. Bordeaux, L., Salaün, G., Berardi, D., Mecella, M.: When are two web services compatible? In: Shan, M.-C., Dayal, U., Hsu, M. (eds.) TES 2004. LNCS, vol. 3324, pp. 15–28. Springer, Heidelberg (2005)
5. Diaz, G., Pardo, J.-J., Cambronero, M.-E., Valero, V., Cuartero, F.: Verification of web services with timed automata. In: Proceedings of the International Workshop on Automated Specification and Verification of Web Sites (WWV 2005). ENTCS, vol. 157, pp. 19–34 (2005)
6. Eder, J., Tahamtan, A.: Temporal conformance of federated choreographies. In: Bhowmick, S.S., Küng, J., Wagner, R. (eds.) DEXA 2008. LNCS, vol. 5181, pp. 668–675. Springer, Heidelberg (2008)
7. Guermouche, N., Godart, C.: Timed model checking based approach for compatibility analysis of synchronous web services. Research report (2008)
8. Guermouche, N., Godart, C.: Timed properties-aware asynchronous web service composition. In: Proceedings of the 16th International Conference on Cooperative Information Systems (CoopIS 2008), Monterrey, Mexico, November 9-14, 2008, pp. 44–61 (2008)
9. Guermouche, N., Perrin, O., Ringeissen, C.: Timed specification for web services compatibility analysis. In: International Workshop on Automated Specification and Verification of Web Systems (WWV 2007), San Servolo island, Venice, Italy, December 14, 2007, pp. 155–170 (2007)
10. Kazhamiakin, R., Pandya, P.K., Pistore, M.: Representation, verification, and computation of timed properties in web service compositions. In: Proceedings of the IEEE International Conference on Web Services (ICWS), pp. 497–504 (2006)
11. Kazhamiakin, R., Pandya, P.K., Pistore, M.: Timed modelling and analysis in web service compositions. In: Proceedings of the The First International Conference on Availability, Reliability and Security, ARES, pp. 840–846. IEEE Computer Society Press, Los Alamitos (2006)
12. Ponge, J.: A new model for web services timed business protocols. In: Atelier (Conception des systèmes d'information et services Web) SIWS-Inforsid) (2006)
13. Ponge, J., Benatallah, B., Casati, F., Toumani, F.: Fine-grained compatibility and replaceability analysis of timed web service protocols. In: Parent, C., Schewe, K.-D., Storey, V.C., Thalheim, B. (eds.) ER 2007. LNCS, vol. 4801, pp. 599–614. Springer, Heidelberg (2007)