Chapter 16
Composition of Web Services: From Qualitative to Quantitative Timed Properties

Nawal Guermouche and Claude Godart

Abstract Dealing with service composition is an important and challenging issue of distributed systems. Existing works investigate mechanisms for analyzing and synthesizing a composition based on qualitative properties which characterize operations and/or messages choreography constraints. Apart from these qualitative properties, quantitative properties such as time related features are a crucial setting to consider. Augmenting service's behavior with timed properties increases the expressiveness and brings new difficult problems. This requires defining rigorous verification and composition primitives for taking into account such properties. In this chapter, we present a formal composition and verification approach which considers quantitative timed properties assigned to qualitative properties. The chapter starts with a general introduction. Then, it introduces the concepts related to timed Web services, timed conversations and protocols. The following section introduces the notion of composition of Web services with emphasis on the temporal dimension, and defines a formal composition approach. This approach relies on the generation of a mediator which aims surpassing timed conflicts. The next section presents validation primitives based on model checking techniques to verify and validate timed compositions. An implementation of the concepts previously introduced is then described. Before concluding with a larger consideration of time implication in Web services definition and composition, and with open issues, we present a study of the state of the art.

N. Guermouche (✉)
CNRS, LAAS, 7 avenue du colonel Roche, Toulouse F-31400, France
e-mail: nawal.guermouche@laas.fr

N. Guermouche
Univ de Toulouse, INSA, LAAS, Toulouse F-31400, France

C. Godart
LORIA-INRIA-UMR 7503, Vandoeuvre-les-Nancy F-54506, France
e-mail: nawal.guermouche@laas.fr

A. Bouguettaya et al. (eds.), *Web Services Foundations*,
DOI: 10.1007/978-1-4614-7518-7_16,

16.1 Introduction

Service Oriented Architecture (SOA) is gaining acceptance as a promising architecture for organizations to integrate their business applications. In SOA, application's business logic can be modularized and outsourced as Web services so that these services can be mutually used. Based on standards, Web services promote the composition of loosely coupled applications to integrate them into complex business systems. In this field, many industrial and academic efforts have been done to provide specifications and techniques to allow verification and composition of heterogeneous Web services [4].

Web service description is one of the important ingredients for Web service composition. In fact, selecting, using, and composing services in efficient and correct manner, requires to provide rich specifications for describing various kind of important service properties. Indeed, in real life scenarios, Web services and more particularly Web service composition depends on several properties, such as those related to messages choreography constraints [4], security [15], and *timed properties* [13, 25].

In this chapter, we focus on the Web service composition synthesis problem where we consider *qualitative properties* associated with *quantitative properties*. Qualitative properties define messages choreography constraints and quantitative properties relate to timed properties which specify the *necessary delays* to exchange messages (e.g., in an e-government application a prefecture must send its final decision to grant an handicapped pension to a requester after 7 days and within 14 days). Thus, we consider that building correct compositions requires managing message choreography constraints augmented with timed properties. Few recent works have shown the importance to deal with such timed properties in the compatibility analysis of synchronous [27], asynchronous Web services [14], in checking requirements satisfaction [18], and in calculating temporal thresholds for process activities [25].

Since services are developed autonomously, mismatches can arise and a composition can fail. Mainly, we distinguish two kind of problems: non-timed and timed mismatches. Non-timed problems concern interfaces and sequence messages conflicts which happen when: (1) awaited messages are not produced by other services, (2) awaited and sent messages are not adequate (i.e., they have different names or different data types), and (3) there is a mutual services blocking (e.g., a service Q_1 waits for a message m that must be sent by another service Q_2, which also waits for a message m' from Q_1 to send the awaited message m). Detecting and preventing such composition timed problems is a difficult and challenging problem [13]. In fact, when composing services, dependencies between timed properties can be created and some dependencies can generate timed conflicts. In the context of service composition, it is important to detect and prevent such timed conflicts to anticipate composition failures. To do so, a possible solution, is to build a third party service, called *mediator*. The notion of a *mediator* has been already used to solve many problems as data integration [28, 33], Web semantic heterogeneities [30], adaptation of services interfaces (namely *adaptators*) [2], for discovering appropriate services to satisfy client's preferences [10], and as an interface between Web services [4].

To summarize, the problem we are interested in can be defined as follows: given a timed description of a given need, called *client service* in the following, and a set of discovered timed services, how to build a composition of discovered services to satisfy this client service. Note that we focus on correct interactions of services and we do not consider exception handling which are out of the scope of this chapter. The main contributions of our framework are as follows:

1. Unlike existing composition synthesis models, we propose a formal model of asynchronous Web services that takes into account *qualitative timed properties* associated to messages, data, and data constraints.
2. As we deal with timed properties, when synthesizing a composition, timed conflicts can arise. We propose a mechanism to discover these conflicts.
3. In addition, we propose the use of a mediator based process to anticipate and prevent, when possible, the problem of timed (and non timed) conflicts.
4. We propose a model checking based verification process which can be used to validate Web service compositions.
5. Finally, the primitives described in this chapter have been implemented in a prototype that we have used to perform preliminary tests.

The reminder of the chapter is organized as follows: in Sect. 16.2 we present a global overview of our framework. Section 16.3 describes the model we propose in order to specify the Web services properties we consider. Section 16.4 describes our composition approach steps. Section 16.5 presents a concrete example of composition to illustrate our approach. In Sect. 16.6, we present a verification process which aims at verifying compositions of Web services and an experimental setup. Related work is introduced in Sect. 16.7, and finally Sect. 16.8 concludes.

16.2 Global Overview

In this section, we present an overview of our timed composition framework which relies on the following elements:

- *A client Service*: the first element of our framework is the timed description of the client service. This service specifies timed properties associated to the data flow the client provides and to the data flow he expects without any reference to the operations of available services.
- *A set of discovered services*: we assume that a set of timed Web services can be discovered to answer the client service request.
- *A mediator*: it can access the data yet exchanged by the different services and use them to generate any missing messages.

Case Study: e-government Application

Let us present a part of an e-government application inspired from [21] to illustrate the related issues of the problem we handle. The goal of the e-government application we consider is to manage handicapped pension requests. Such a request involves three organizations: (1) *a prefecture*, (2) *a health authority*, and (3) *a town hall*. We suppose that theses organizations are managed by, respectively, the prefecture service (PS), the health authority service (HAS), and the town hall service (THS).

A high level choreography model of the process is depicted in Fig. 16.1. A citizen can apply for a pension. Once applied, the prefecture solicits the medical entity to deliver an examination report of the requester, and the town hall to deliver the domiciliation attestation. After studying the received files, the prefecture sends the notification of the final decision to the citizen. The interaction between these partners is constrained by timed requirements:

- Once the health authority service proposes meeting dates to the citizen, this one must confirm the meeting within 24 h.
- The prefecture requires at least 48 h and at most 96 h from receiving the file from the requester to notifying the citizen with the final decision.
- The medical report must be sent to the prefecture after at least 120 h and at most 168 h after receiving the request of the medical report.

Notion of Timed Conflicts

Given this set of timed Web services and the client service, our aim is to build a timed composition that satisfies this client service. When building a composition, it

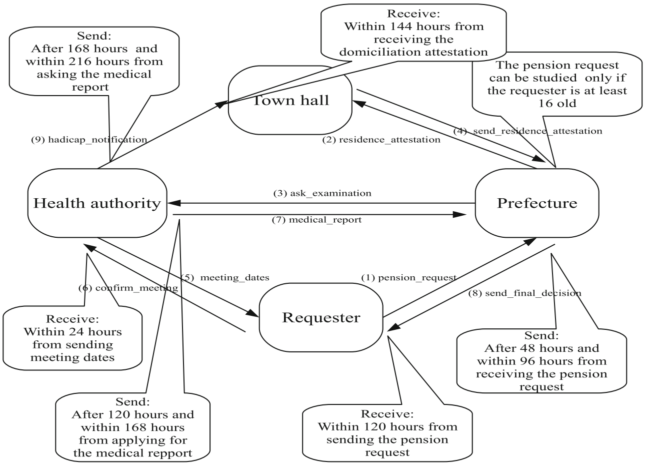

Fig. 16.1 Global view of the e-government application

is mandatory to ensure that data and timed constraints of the involved services are not conflicting. In the context of our work, we do not focus on data type and semantics related analysis problems. We consider simple data which can be simply checked: two data constraints are said to be not conflicting if their solution set is not disjoint. For example, the prefecture studies the pension request only if the requester is at least 16 years old. If we want to create a connection between the requester and the prefecture service to exchange the pension request while the requester is for example at least 18 old, this is possible (i.e., the set of solution of age $\geq 16 \cap$ the set of solution of age$\geq 18 \neq \emptyset$).

While the data constraints we consider can be checked by verifying their set of solutions, timed constraints validation needs more complex investigations. In fact, in a collaboration, timed properties of Web services cannot be checked like simple constraints. In other words, to assert that an interaction is timed deadlock free, it is not sufficient to check timed constraints assigned to sending a message with timed constraints associated to its reception. For example, the prefecture must send its final decision after 48 h and within 96 h from receiving the pension request. On the other side, the requester must receive it within 120 h from sending the request. If we check these two timed constraints as simple constraints, we can conclude that the prefecture and the requester can collaborate together. However, if we examine the progress of the interaction, we can remark that the prefecture can send its final decision only after the medical report has been received. This report must be sent by the medical entity after 120 h and within 168 h from receiving the report request. Since the prefecture must wait for the medical report to send its final decision, i.e., after 120 h, the final decision cannot be sent within 96 h from receiving the pension request. Figure 16.2, illustrates this conflicting interaction. The prefecture sends its decision after 48 h and within 96 h from receiving the pension request. But during this execution, the prefecture must wait for at least 120 h to get the medical report. This presents a simple timed conflict. More complex timed conflicts can arise and can make fail the

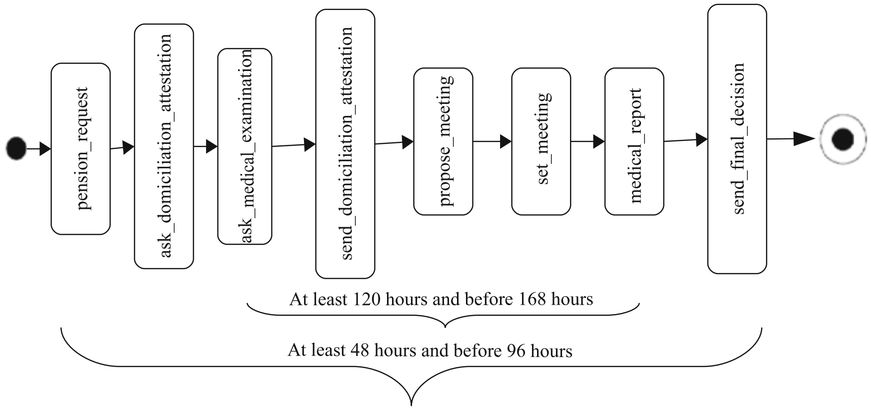

Fig. 16.2 Example of impact of timed properties on web services interaction

composition. As said previously, to succeed the composition, an alternative consists in generating a mediator whose role is to try to prevent these conflicts.

Now, let us check the scenario depicted in Fig. 16.1. We can remark that the town hall has to wait for a medical report of the medical entity before to, for example, deliver an handicapped card. The town hall must receive the report within 144 h, but the medical entity can send its report only after 168 h. So, the town hall cannot receive the report in time and the composition will fail.

But, if we examine the situation in details, we can remark that the medical entity sends its report to the prefecture after 120 h. As a consequence, intuitively, to succeed the collaboration, we can build an indirect connection between the medical entity and the town hall to deliver the medical report within 144 h. This indirect connection can be built by the mediator that generates the message for transmitting the medical report to the town hall in advance. Note that the mediator fails when a required data (i.e., the data involved in the required message) is not available (i.e., the data is not accessible).

To summarize, in this section we have intuitively discussed the impact and the importance to consider timed properties in a composition framework. During a composition, different services with different timed constraints can be involved. Timed properties can give rise to timed conflicts which can make fail the composition. In the following, we propose a formal approach which aims at composing services so that their timed properties are respected.

16.3 Modeling the Timed Behavior of Web Services

As introduced above, one of the important ingredients in a composition framework is the *timed conversational protocol* of Web services which we assume deterministic and able to support synchronous and asynchronous communications. 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 have adopted a deterministic timed automata 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 is denoted by $m(d_1, \ldots, d_n)$, or $m(\bar{d})$ for short. In an *asynchronous* communication, when a message is sent, it is inserted into a bounded message queue, and the receiver can consume (i.e. receives) the message when it is available in the queue. To capture the timed properties when modelling Web services, we use 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 labelled 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 | $x \bowtie c$ | $\psi_1 \wedge \psi_2$, where $\bowtie \in \{\leq, <, =, \neq, >, \geq\}$, $x \in X$, $\psi_1, \psi_2 \in \Psi(X)$, and c is a constant. With that:

Definition 16.1. A *timed conversational protocol* Q is a tuple (S, s_0, F, M, C, X, T) where S is a set of states, s_0 is the initial state, F is a set of final states ($F \subseteq S$), M is a set of messages, C is a set of constraints over data, X is a set of clocks, and T is a set of transitions such that $T \subseteq S \times M \times C \times \Psi(X) \times 2^X \times S$ with an element of the alphabet (exchanged message (M)), a constraint over data (C), a guard over clocks ($\Psi(X)$), and the clocks to be reset (2^X).

The conversational protocols we consider are deterministic. A conversational protocol is said to be deterministic if for each two transitions $(s, \alpha_1, c_1, \psi_1, s'_1)$ and $(s, \alpha_2, c_2, \psi_2, s'_2)$, the following conditions are satisfied:

$\alpha_1 \neq \alpha_2$, or $c_1 \wedge c_2 =$ false, or $\psi_1 \wedge \psi_2 =$ false

Example 16.1 Figure 16.3 illustrates the timed conversational protocol of the PS, THS, HAS services of our use case study, and the client service. In this figure, the initial state of the PS service is p_0, the set of states is $\{p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_9, p_{10}, p_{11}, p_{12}, p_{13}, p_{14}, p_{15}, p_{16}\}$ and the set of final states is $\{p_7, p_9, p_{16}\}$. This service can send and receive messages. For example, it can send the message $examination\_request(sn, handicap)$, denoted $!examination\_request(sn, handicap)$. This message has as parameters the security number (sn), and the handicap ($handicap$) of the requester. Analogously, this service can consume a message, for example, the message $pension\_request(sn, age, handicap)$, denoted $?pension\_request(sn, age, handicap)$. This message has as parameters the security number (sn), the age (age), and the handicap ($handicap$) of the requester. This service achieves correctly its execution if for each interaction it reaches a final state.

To specify that the prefecture must send its final decision within a delay of 48–96 h after receiving the pension request, we associate to the reception of the request of the pension a reset of a clock t_1 ($t_1 := 0$) and we assign the constraint $48 \leq t_1 \leq 96$ to the sending of the final decision.

16.4 Analyzing the Timed Composition Problem

In this section, we present the algorithm that allows to synthesize a composition of timed Web services. Our framework gathers three steps: (1) creating timed P2P connections between the client service and the discovered services (see Sect. 16.4.1), (2) discovering timed conflicts (see Sect. 16.4.2), (3) generating a *mediator* that tries to step in to succeed a connection (see Sect. 16.4.3).

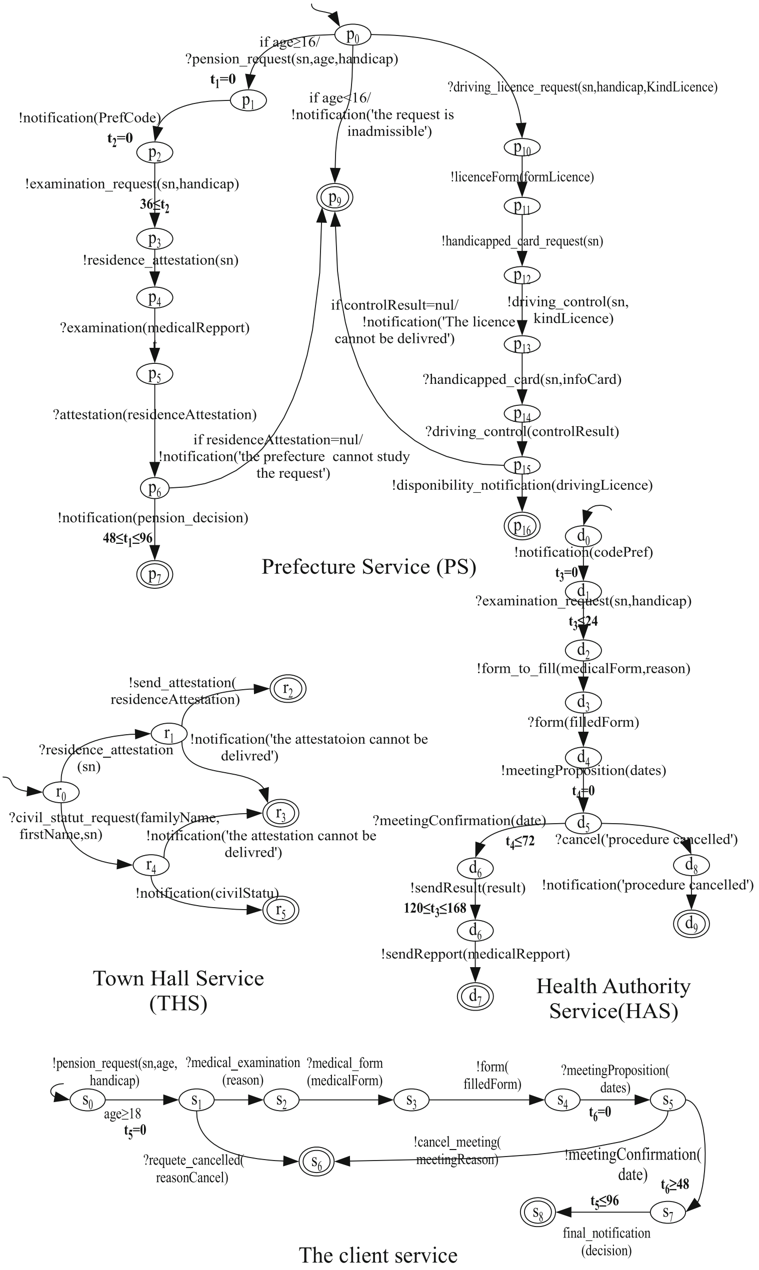

Fig. 16.3 Services of the e-government scenario

16.4.1 Building Timed P2P Connections

Given a set of conversational protocols of the services and a client service, our aim is to build a *timed global automaton* that characterizes the *timed composition schema* (the global automaton is called *Timed Composition Schema Automaton TCSA*).

To build this TCSA, we introduce the concept of *configuration* that represents the states of the TCSA at a given time. A configuration defines the evolution of services states when they are interacting together (i.e., connected via channels). In the initial configuration, all the services are in their initial states. Given a source configuration, the TCSA reaches a new configuration when there exists two services that change their states by exchanging a message so that no timed conflict arises.

Definition 16.2. (*A Timed Composition Schema Automaton*)
A *timed composition schema automaton TCSA* is a tuple (S, Q, M, X, L, T) such that S is a set of configurations, Q is a set of services, M is a set of messages, L is a set of channels, X is a set of clocks, and T is a set of TCSA transitions such that $T \subseteq S \times L \times \Psi(X) \times S$. A transition specifies that, from a source configuration, the TCSA reaches a new configuration when a channel can be created to interconnect two services so that the associated (ordered) timed constraints are satisfied. The set of channels L is defined as a set of $(p_s, p_r, m(\overline{d}))$, with $p_s, p_r \in Q$, and the tuple $(p_s, p_r, m(\overline{d}))$ specifies that the service p_s sends the message $m(\overline{d})$, that involves the set of data types $(\overline{d})$, to the service p_r. In our composition framework, a mediator can be generated, hence the set of considered services is $Q = \{R, A, Med\}$, such that R is the client service, A is the set of the available services, and Med is the generated mediator.

Among the transitions of the different services, we distinguish two kinds of transitions: *passive transitions* and *non-passive transitions*.

- A *passive transition* is a timed (resp. non-timed) transition that has timed constraints of the form $x \leq v$ (resp. $x < v$). In fact, these transitions are considered passive because they do not give rise to timed conflicts.
- A *non-passive transition* is a timed transition that has timed constraints of the form $x \geq v$ (resp. $x > v$). In fact, timed conflicts can arise when these transitions precede transitions that have constraints of the form $x \leq v$ (resp. $x < v$).

The approach of composition is based on the Algorithm 1. This algorithm aims to build connections between the different services to try to satisfy the client service. The steps of this algorithm can be described as follows:

From the set of transitions T, it isolates passive transitions T_p and non-passive transitions T_{np}. Initially, it tries to connect each transition of the client service with the transitions of the different services. Note that this algortihm tries to connect passive transitions before non-passive transitions. In fact, the study we have performed shows that timed conflicts can arise when non-passive transitions precede passive transitions. When the connection fails, this algorithm calls the Algorithm 3 that aims at generating the mediator. When a connection is created, the Algorithm 2 checks if the created connection does not give rise to timed conflict. In the following, we present the process of discovering timed conflicts.

16.4.2 *Making Explicit the Implicit Timed Constraints Dependencies*

As said previously, when creating TCSA transitions, implicit timed dependencies can be created. In that case, timed conflicts can arise. In order to discover timed conflicts when combining services, we need mechanisms for making explicit the implicit timed dependencies. To do so, we propose the *clock ordering* process. The idea behind the *clock ordering* process is to define an order between the different clocks of the services for each new TCSA transition.

To explain why simple checking of timed constraints as simple constraints (called local checking) is not sufficient to detect conflicts, we consider the following example depicted in Fig. 16.4.

Example 16.2 Let us consider the two timed conversational protocols P and P'. We start by building the TCSA of the two conversational protocols by considering the timed constraints as simple constraints, i.e., we check locally the timed constraints of the transitions.

As we can see, the service P sends the message m_0 and resets the clock x. The service P' can receive this message. So we can build the TCSA transition $(s_0s'_0, m_0, x_1 = 0, s_1s'_1)$. Then the service P' sends the message m_1 and resets the clock y. The service P can receive the message m_1. We build the TCSA transition $(s_1s'_1, m_1, y = 0, s_2s'_2)$. Later, the service P sends the message m_2, the service P' can receive it after 20 units of time. Hence, we build the TCSA transition $(s_2s'_2, m_2, y \geq 20, s_3s'_3)$. After that, the service P' sends the message m_3, the service P must receive it within 10 units of time. We build the TCSA transition $(s_3s'_3, m_3, x < 10, s_4s'_4)$. As we can see in Fig. 16.4a, by simply checking timed constraints of transitions, we could build a TCSA.

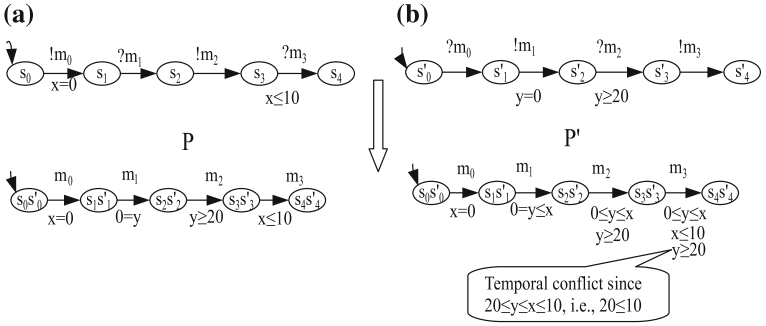

Fig. 16.4 Make explicit the implicit timed constraints dependency. **a** Local checking of the constraints of the transitions. **b** Clock ordering

However, the message m_2 can be exchanged after 20 units of time and m_3 can be exchanged within 10 units of time. As m_3 can be exchanged after exchanging the message m_2, it can be exchanged only after 20 units of time. However, the message m_3 should be exchanged within 10 (i.e., [0, 10]) units of time and after 20 (i.e., [20, ∞)) units of time, which is a contradiction and represents a timed conflict. To cater for such implicit timed properties, we propose to perform a *clock ordering* process. This process allows to define an order between the clocks of the TCSA transitions. Below, we show how we define the clock order.

The two services can exchange the message m_0 via the TCSA transition ($s_0s'_0$, m_0, $x = 0$, $s_1s'_1$). Then when building the TCSA transition ($s_1s'_1$, m_1, $y = 0$, $s_2s'_2$)

Algorithm 1: Composition

```
Input: A client service $Q_g = (S_c, s_{0_c}, F_c, M_c, C_c, X_c, T_c)$, a set of Web services
        $Q_i = (S_i, s_{0_i}, F_i, M_i, C_i, X_i, T_i)$, for $i = \{1, .., n\}$
, the initial configuration of the TCSA $\overline{s} = (s_{0_1}, \ldots, s_{0_n})$, the current state of the client service
$s_c = s_{0_c}$
Output: TCSA $= (S, s_0, F, M, C, X, T)$, and the mediator
        $Med = (S_{med}, s_{0_{med}}, F_{med}, M_{med}, C_{med}, X_{med}, T_{med})$
begin
    succesComposition=true;
    for each transition $t_c = (s_c, \alpha_c, c_c, \psi_c, Y_c, s'_c)$ of each trace of the client service and if
    succesComposition do
        $T_{visited} = null$;
        if ($t_c \in T_p$) and (satisfaction($t_c$,$\overline{s}$)) then
            $s_c = s'_c$;
        else
            if ($t_c \in T_p$) then
                choose one transition $t_p$ of $T_p$;
                $T_{visited} = T_{visited} \cup t_p$;
                while $T_p \not\subseteq T_{visited}$ and $\neg$ satisfaction($t_{sp}, \overline{s}$) do
                    choose another transition $t_{sp}$ of $T_{sp}$;
                    $T_{visited} = T_{visited} \cup t_{sp}$;
                if $\neg$ satisfaction($t_{sp}$,$\overline{s}$) then
                    Choose one transition $t_{snp}$ de $T_{snp}$;
                    $T_{visited} = T_{visited} \cup t_{snp}$;
                    while $T_{snp} \not\subseteq T_{visited}$ and $\neg$ satisfaction($T_{snp}, \overline{s}$) do
                        choose another transition $t_{snp}$ of $T_{snp}$;
                        $T_{visited} = T_{visited} \cup t_{snp}$;
                    if $\neg$ satisfaction($t_{snp}$,$\overline{s}$) then
                        succesComposition=false;
            else
                if $(\psi_c = x > v) or (\psi_c \geq v)$ then
                    choose a transition $t_{sp}$ of $T_{sp}$;
                    $T_{visited} = T_{visited} \cup t_{sp}$;
                    while $T_{sp} \neq \emptyset$ and $\neg satisfaction(T_{sp}, \overline{s})$ do
                        choose another transition $t_{sp}$ of $T_{sp}$;
                        $T_{visited} = T_{visited} \cup t_{sp}$;
                    if $\neg$ satisfaction($t_{sp}$,$\overline{s}$) then
                        if satisfaction($t_c, \overline{s}$) then
                            $s_c := s'_c$
                        else
                            choose a transition $t_{snp}$ of $T_{snp}$;
                            $T_{visited} = T_{visited} \cup t_{snp}$;
                            while $T_{snp} \neq \emptyset$ and $\neg satisfaction(T_{snp}, \overline{s})$ do
                                choose another transition $t_{snp}$ of $T_{snp}$;
                                $T_{visited} = T_{visited} \cup t_{snp}$;
                            if $\neg$ satisfaction($t_{snp}$,$\overline{s}$) then
                                succesComposition=false;
    if succesComposition then
        if $\overline{s} \in F$ then
            Return ASCT and the mediator;
        else
            'The composition fails because the services do not reach their final states';
    else
        'The composition fails because the client service cannot be satisfied';
```

we can define the order $y \leq x$ since y is reset after x. So we associate this order to the TCSA transition as follows $(s_1 s'_1, m_1, 0 \leq y \leq x, s_2 s'_2)$. Then, the service P can send the message m_2 to the service P' which can receive it after 20 units of time. So when the two services exchange the message m_2, $(0 \leq y \leq x) \wedge (y \geq 20)$ must be satisfied. We build the TCSA transition $(s_2 s'_2, m_2, 0 \leq y \leq x, y \geq 20, s_3 s'_3)$. Until now, there is no timed conflict. Note that we propagate the constraint $y \geq 20$ over the successor transitions. When the service P' sends the message m_3, the service P can receive it within 10 units of time, i.e., $20 \leq y \leq x \leq 10$ must be satisfied. However, this latter induces to a timed conflict $(20 \leq 10)$. As we can see in Fig. 16.4b, by defining a clock ordering when combining services, implicit timed conflicts can be discovered.

The Algorithm 2 allows to define an order between the different clocks of services. Based on the computed order, it detects timed conflicts. This algorithm has as input a candidate TCSA transition $t_i = (s_i, m_i(\overline{d}), c_i, \psi_i, Y_i, s'_i)$. To discover timed conflicts, it proceeds as follows.

- It propagates timed constraints, of the form $x > v$ (resp. $x \geq v$), from a predecessor transition t_{i-1} to the transition t_i.
- A clock z which is reset in a predecessor transition t_{i-1}, has a value bigger than a clock y which is reset in the current transition t_i. Hence, it defines the order $y \leq z$.
- In addition, it propagates the order $z_1 \leq \cdots \leq z_n$ of the predecessor transition t_{i-1}.
- If in the transition t_i there exists a constraint of the form $x \leq v$ (resp. $x \geq v$) and at the same time, a clock y is reset, then it defines the order $x - y \leq v$ (resp. $x - y \geq v$). That means, the difference between the two clocks x and y is always less (resp. bigger) than v.
- If among the set of constraints and defined orders, there exists two constraints $x \geq v$ and $x' \geq v'$, and at the same time, there is an order of the form $x - x' \geq v$, it implies the order $x \geq v + v'$. In fact, this order allows to consider the clocks value accumulation.

By applying theses steps when building TCSA transitions, timed conflicts are discovered if at least one of the following conditions is satisfied.

- There exists an order of the form $v \leq x_1 \leq \cdots \leq x_n \leq v'$ where $v' \leq v$.
- There exists three constraints $x \geq v'$ and $y \leq v''$ and $x - y \leq v$ with $v' - v'' > v$ (i.e., following the constraints $x \geq v'$ and $y \geq v''$, the difference $x - y \leq v$ is violated).
- There exists three constraints $x \leq v'$, $y \leq v''$ and $x - y \geq v$ with $v' < v$ (i.e., the constraint $x - y \geq v$ is violated),
- There exists three constraints $x \leq v'$, $y \geq v''$, and $x - y \geq v$ with $v' - v'' < v$ (i.e., the constraint $x - y \geq v$ is violated).

Algorithm 2: Clock_Order

```
Input: A transition $(s_i, m_i(\overline{d}), \psi_i, Y_i, s'_i)$
Output: boolean
begin
    if $s_i$ is the initial state then
        return true;
    else
        for each $\varrho_{i-1} \in \psi_{i-1}$, such as $\varrho_{i-1} = x \geq v$ or $\varrho_{i-1} = x > v$ of
        $(s_{i-1}, m_{i-1}(\overline{d}), \psi_{i-1}, Y_{i-1}, s'_{i-1})$ do
            $\psi_i = \psi_i \cup \varrho_{i-1}$;
            for each $y = 0 \in Y_i$ and $z = 0 \in Y_{i-1}$ do
                $\psi_i = \psi_i \cup y \leq z$;
            for each $z_1 \leq z_2 \in \psi_{i-1}$ do
                $\psi_i = \psi_i \cup z_1 \leq z_2$;
            for each $y \in Y_i$ et $\varrho_i \in \psi_i$ do
                if $\varrho_i = x \leq v$ then
                    $\psi' = x - y \leq v$;
                    $\psi_i = \psi_i \cup \psi'$
                else
                    if $\varrho_i = x \geq v$ then
                        $\psi' = x - y \geq v$;
                        $\psi_i = \psi_i \cup \psi'$
            if $\exists \varrho_i = x \geq v$ and $\varrho'_i = x' \geq v'$ and $\varrho''_i = x - x' \geq v$ then
                $\psi' = x \geq v + v'$  $\psi = \psi \cup \psi'$
        if $\exists\ v \leq y_0 \leq \ldots \leq y_n \leq v' \in \psi_i$ such as $v' < v$ then
            return false;
        else
            if $\exists\ x \geq v' \in \psi_i$ and $y \leq v'' \in \psi_i$ and $x - y \leq v \in \psi_i$ such as $v' - v'' > v$ then
                return false;
            else
                if $\exists\ x \leq v' \in \psi_i$ and $y \leq v'' \in \psi_i$ and $x - y \geq v \in \psi_i$ such as $v' \geq v''$ and $v' < v$
                then
                    return false;
                else
                    if $\exists\ x \leq v' \in \psi_i$ and $y \geq v'' \in \psi_i$ and $x - y \geq v \in \psi_i$ such as $v' - v'' < v$ then
                        return false;
                    else
                        return true;
```

The clock ordering process is very important as it allows to predict timed conflicts. A simple technique such as using only a mediator, whose aim is to provide messages without a clock ordering process, will be insufficient and cannot resolve a problem when it arises (i.e., when a timed conflict occurs it means that timed properties are violated). Indeed, our goal is to predict and prevent timed conflicts before they arise. To do so, we use the clock ordering process in association with a mediator.

16.4.3 Generation of a Timed Mediator

As said previously, because of timed (and non-timed) conflicts, a timed P2P connection process can fail. The mediator aims to prevent these conflicts by creating the required messages. In our approach, a required message is created taking the involved data from the history of past exchanged messages, i.e., the current available data (we assume here that data having the same name, have also the same value).

In order to produce the required messages, we check if the involved data are available, i.e, they have been already exchanged. In other terms, the mediator reuses the data historic to produce the required messages.

The mediator is defined using the computed TCSA, by adding input, output and empty messages. As long as the TCSA can be executed, the mediator does nothing. When two services can exchange a message and there are clocks which are reset, the mediator resets the same clocks via an empty transition. In fact, these clocks can be used later by the mediator to consume messages within a defined time window, whilst, when a deadlock can arise, the mediator generates the required message to prevent this deadlock.

Algorithm 3: Generation of a mediator

```
Input: A transition $t_i = (s_i, \alpha(\overline{d}), c, \psi, Y, s'_i)$, the set of exchanged data $D_a$
Output: A transition
mediator($(s_i, \alpha_i, c_i, \psi_i, Y_i, s'_i), D_a, \overline{s}$);
begin
    if $\alpha =!m(\overline{d_m})$ then
        $M_{med} := M_{med} \cup m(\overline{d})$;
        $S_{med} := S_{med} \cup s'_m$;
        $T_{med} := T_{med} \cup (s_m, ?m(\overline{d}), c_i, \psi_i, Y_i, s'_m)$ ;
        return $(s_m, ?m(\overline{d}), c_i, \psi_i, Y_i, s'_m)$;
    else
        if $\alpha =?m(\overline{d_m}))$ then
            if $\overline{d_m} \subseteq D_a$ then
                $M_{med} := M_{med} \cup m(\overline{d_m})$;
                $S_{med} := S_{med} \cup s'_m$;
                $T_{med} := T_{med} \cup (s_m, !m(\overline{d_m}), c_i, Y_i, s'_m)$ ;
                return $(s_m, ?m(\overline{d}), c_i, Y_i, s'_m)$;
            else
                The required data are not available, so the required message cannot be produced;
                return null;
```

16.5 Back to the Case Study

In order to illustrate the approach presented in this chapter, we propose to show a concrete composition example using the PS, HAS, THS services, and the client service introduced in Sect. 16.2. We first try to build a TCSA (Sect. 5.1, Fig. 16.5a without the timed involvement of a mediator. Then we introduce the mediator (Sect. 5.2, Fig 16.5b to resolve timed problems.

16.5.1 Composition Without the Timed Involvement of the Mediator

As in our framework, a mediator can be involved, we generate an empty mediator that has initially only one state m_0. The initial configuration of the *TCSA* is

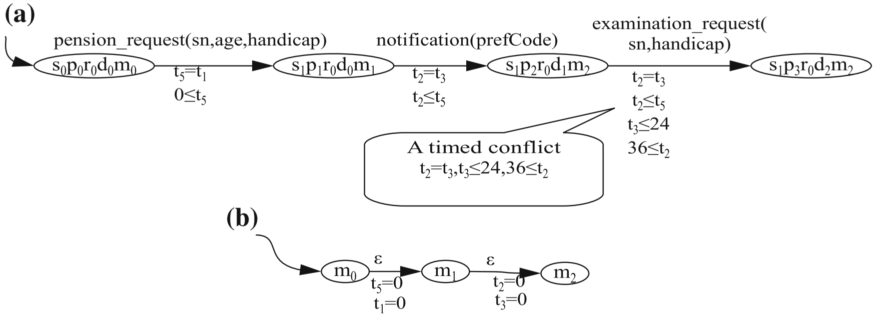

Fig. 16.5 Composition without the timed involvement of the mediator. **a** A conflicted TCSA. **b** The associated mediator

$s_0p_0r_0d_0m_0$ (respectively the client service, PS, HAS, THS, and the mediator are in their initial states). From the current state of the client service s_0, the message $!pension\_request(sn, age, handicap)$ can be sent. As we can remark, the PS service waits for this message. Since, the constraints over data ($age \geq 18$ and $age \geq 16$) are not disjoint, we can connect the two transitions (s_0, $!pension\_request(sn, age, handicap), age \geq 18, t_5 = 0, s_1$) and ($p_0$, $?pension\_request\ (sn, age, handicap), age \geq 16, t_1 = 0, p_1$). When the two transitions are fired, the two clocks t_1 and t_5 are reset. So, we generate an empty mediator transition that allows to reset the same clocks. In fact, theses clocks can be used later to specify constraints to produce or consume messages. We build a global TCSA transition that connects the two transitions of the client and PS services with the transition of the mediator $(s_0p_0r_0d_0m_0, pension\_request(sn, age, handicap), t_1 = t_5 = 0, s_1p_1r_0d_0m_1)$. The new configuration becomes $s_1p_1r_0d_0m_1$ and the new current state of the client service becomes s_1. From this new configuration, the current transition of the client service is $(s_1, ?medical\_examination(reason), s_2)$. There is no transition that enables sending the message $medical\_examination(reason)$. So we check if the mediator can produce this message. Since the data $reason$ has not been already exchanged, the mediator cannot generate the message $medical\_examination(reason)$. Among the services transitions, we choose the transition $(p_1, !notification(prefCode), t_2 = 0, p_2)$. Since, the HAS service can consume it, we can connect them. As the clocks t_2 and t_3 are reset, we generate an empty mediator transition that reset the same clocks. We build the TCSA transition $(s_1p_1r_0d_0m_1, notification(prefCode), t_2 \leq t_5, t_2 = t_3, s_1p_2r_0d_1m_2)$. From the new configuration, the HAS service waits for the message $examination\_request(sn, handicap)$ that must be consumed within 24 h from receiving the message $notification(codePref)$. The message $examination\_request(sn, handicap)$ can be sent by the PS after 36 h from sending the message $notification(codePref)$. We build the TCSA transition $(s_1p_2r_0d_1m_2, examination\_request(sn, handicap), t_2 = t_3, t_2 \leq t_5, t_3 \leq 24, t_2 \geq 36, s_1p_3r_0d_2m_2)$. This transition is conflicting, since $t_2 = t_3$, $t_3 \leq 24$ et $t_2 \geq 36$. Thus, we can see that without involving the mediator to handle timed conflicts, the compositions fails.

16.5.2 Involving the Mediator

We show here how the mediator can be involved to handle timed conflicts.

To generate the TCSA (Fig. 16.6a) and the associated timed mediator (Fig. 16.6b), we use the following steps. We apply the same steps described above to reach the configuration $s_1p_2r_0d_1m_2$. From this configuration, the HAS service can fire the passive transition $(d_1, ?examination\_request(sn, handicap), t_3 \leq 24, d_2)$. Since the corresponding transition of the PS service $(p_2, !examination\_request(sn, handicap), t_2 \geq 36, p_3)$ is a non-passive transition, we check if the mediator can generate the message $examination\_request(sn, handicap)$. The data sn, and $handicap$ have been already exchanged. Hence, the mediator can generate the required message $examination\_request(sn, handicap)$ via the transitions $(m_2, !examination\_request(sn, handicap), t_3 \leq 24, m_3)$. When the message is generated, we build the global transition $(s_1p_2r_0d_1m_2, examination\_request(sn, handicap), t_2 = t_3, t_2 \leq t_5, t_3 \leq 24, s_1p_2r_0\,d_2m_3)$. From the new configuration $s_1p_2r_0d_2m_3$, we choose the passive transition $(d_2, !form\_to\_fill(\,medicalForm, reason), d_3)$ of the HAS service. As there is no service that waits for the message $form\_to\_fill\,(medicalForm, reason)$, we generate the mediator transition to consume this message, i.e., $(m_3, ?form\_to\_fill(medicalForm,\ reason), m_4)$, and then we build the global transition $(s_1p_2r_0d_2m_3,\ form\_to\_fill(\,medicalForm, reason), s_1p_2r_0d_3m_4)$. From the new configuration, the current client transition is $(s_1, ?medical\_examination(reason), s_2)$. There is no transition that enables sending the message $medical\_examination(reason)$. The mediator can produce the message $medical\_examination(reason)$, via the transition $(m_4, !medical_examination\,(reason), m_5)$, and then we build the TCSA transition $(s_1p_2r_0d_3m_4,\ medical\_examination(reason), s_2p_2r_0d_3m_5)$. The current transition of the client service is $(s_2, ?medical\_form(medicalForm), s_3)$. The data $medicalForm$ has been already sent by the HAS service. So, the mediator can generate the missing message

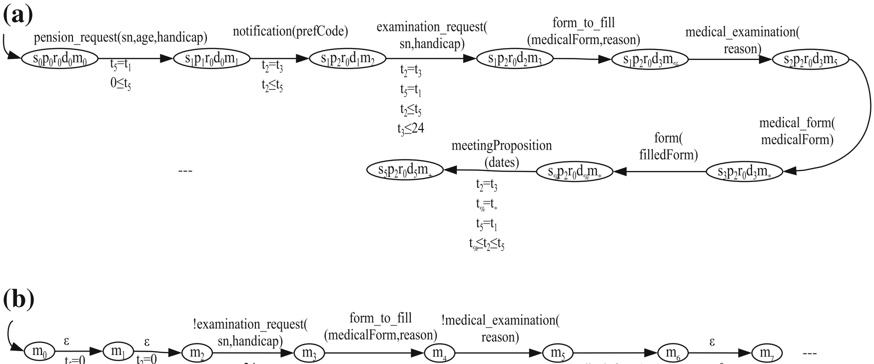

Fig. 16.6 The timed composition schema automaton (TCSA). **a** A part of the generated TCSA. **b** A part of the associated mediator

$medical\_form(medicalForm)$ via the transition $(m_5, !medical\_form(medicalForm), m_6)$. Once the transition of the mediator is generated, we build the global transition $(s_2p_2r_0d_3m_5, medical\_form(medicalForm), s_3p_2r_0d_3m_6)$. From the new configuration, we connect respectively the two transitions of the client and HAS services $(s_3, !form(filledForm), s_4)$ and $(d_3, !form(filledForm), d_4)$ via the TCSA transition $(s_3p_2r_0d_3m_6, form(filledForm), s_4p_2r_0d_4m_6)$.

By applying the same steps, either we build the TCSA, or we detect a conflict that cannot be avoided.

16.6 Formal Verification and Validation of the Built Composition

As presented previously, when the composition succeeds, the algorithm generates a mediator and produces a global timed composition schema TCSA. Such a built TCSA is an *optimized product* built on the fly: indeed, we build progressively the product of timed protocols rather than building the whole product.

The built TCSA is correct if it is deadlock free and it satisfies the client service. Checking that the TCSA is deadlock free can be reduced to checking reachability properties. This problem is PSPACE-complete in general. The problem of client service satisfaction checking can be reduced to the *inclusion problem*, which is decidable [1]. In fact, the formal model of timed conversational protocol that we have defined relies on a deterministic timed automata for which closure and decidability properties have been proved [1].

In the following, we present a formal verification process which aims to validate the built composition. We note that this verification process is generic and can be used to verify atomic and composite services built automatically or manually. This process relies on a model checking approach inspired from [14] and using the UPPAAL model checker.

16.6.1 UPPAAL Overview

UPPAAL is a model checker for the verification and simulation of real time systems [19]. An UPPAAL model is a set of timed automata, clocks, channels for systems (automata) synchronization, variables and additional elements [19].

Each automaton has one initial state. Synchronization between different processes can take place using channels. A channel can be written into (denoted as *channel_name*!), and can be read (denoted as *channel_name*?). A channel can be defined as *urgent* to specify that the corresponding transition must be fired as soon as possible, i.e. immediately and without a delay. Variables and clocks can be associated to processes (automaton). Conditions on these clocks and variables can be associated to transitions and states of the process. The conditions associated to transitions, called *guards*, specify that a transition can be fired if the corresponding guards are satisfiable. The conditions associated to states, called *invariants*, specify that the system can stay in the state while the invariant is satisfiable.

The UPPAAL properties query language is a subset of *Computation Tree Logic* (CTL) [16]. The properties that can be analyzed by UPPAAL are:

- $A[]\psi$: for all the automata' paths, the property ψ is always satisfiable, i.e., for each transition (or a state) of each path, the property ψ is satisfiable.
- $A <> \psi$: for all the automata' paths, the property ψ is eventually satisfiable, i.e., for each path , there is at least one transition (or a state) in which the property ψ is satisfiable.
- $E[]\psi$: there is at least a path in the automata such that the property ψ is always satisfiable, i.e., there is at least one path such that for each transition (or a state), the property ψ is satisfiable.
- $E <> \psi$: there is at least a path in the automata such that the property ψ is eventually satisfiable, i.e., there is at least one transition (or a state) of at least one path in which the property ψ is satisfiable.
- $\psi \rightsquigarrow \phi$: when ψ holds, ϕ must hold.

In the following, we present the formal primitives we propose for composition checking.

16.6.2 Verification of Web service Compositions

In this section, we present the verification process we propose using the model checker UPPAAL. The purpose of this verification process is to check if the built composition holds deadlocks. In this context, we define three composition classes: (1) *fully correct composition* , (2) *partially correct composition*, (3) *incorrect composition*.

16.6.2.1 Fully Correct Composition

We say that a composition is correct if it is (timed and non-timed) deadlock free. This is equivalent to check that its corresponding TCSA does not hold timed and non-timed conflicts. Formally, checking that a composition is fully correct is equivalent to check that all the paths of the TCSA lead to a final state.

Let Q be a TCSA and s_f its final state. Q is said to be fully correct, if the following CTL formula is correct:

$$\boxed{A <> Q.s_f} \tag{16.1}$$

16.6.2.2 Partially Correct Composition

A composition is said to be incorrect if its TCSA is not deadlock free. Formally, a composition is not fully correct if there exists at least a path of the TCSA which does not lead to a final state. This latter can be specified as the following CTL formula:

$$\boxed{E[]\,not\ Q.s_f} \tag{16.2}$$

When the composition is not fully correct, we check if it can achieve at least one correct execution. Formally, a composition can terminate at least one execution if its final state can be reached via at least one path. The former property can be specified as follows.

$$\boxed{E <> \ Q.s_f} \tag{16.3}$$

A composition is said to be partially correct if it is not fully correct (i.e., the property 16.2 is satisfiable) but at the same time it can fulfil at least one execution (i.e., the property 16.3 is satisfiable).

16.6.2.3 Incorrect Composition

When the composition is not even partially correct, we say that the composition is fully incorrect. As specified by the following CTL formula, a composition is said to be fully incorrect if all its TCSA paths do not lead to a final state.

$$\boxed{A[]\,not\ Q.s_f} \tag{16.4}$$

In order to experiment the proposed approach, a prototype has been implemented [12]. Its underlying architecture is depicted in Fig. 16.7. The tool inputs the description of services and the client service as XML documents. The *P2P coordi-*

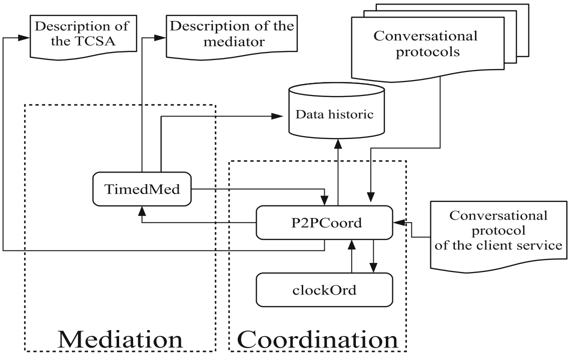

Fig. 16.7 Underlying architecture of the prototype.

nation component tries to build P2P connections (channels) among the services and updates the TCSA description thanks to the Algorithm 1. A third component, the *timed mediator* component, steps in to consume extras messages or to produce, if possible, the required messages using the Algorithm 3. We note that the data historic repository is a database in which we store the involved exchanged data.

16.7 Related Work and Discussion

The research field about how to synthesize automatically a composition is very active. Several research works have been published on automatic service composition, using techniques based on situation calculus [24, 29], transition based systems [9, 23, 11], or symbolic model-checking applied to planning [26]. Unlike the proposed approaches, in our framework we cater for timed properties when composing services.

In [31, 32] the authors consider services as views over data sources. They build on the idea that heterogeneity of data sources may be overcome by exploiting services as wrappers of different information sources, thus providing uniform access to them, exploiting standard protocols such as SOAP and XML. Each data source, i.e., service, is described in terms of input and output parameters (the latter provided by the source), binding patterns and additional constraints on the source. The latter allow to characterize the output data. Analogously, these works consider only atomic services. However, the control flow between data is a crucial aspect. Furthermore, the authors do not consider timed properties.

Like the above works in [20, 22], the considered Web services are atomic. The behavioral aspect is not considered and the timed aspects are not taken into account.

In [6], Web services are described by their BPEL specification. The authors proposed to translate the BPEL specifications into a finite state machine (FSM) specification. As this composition approach is not oriented by the client need (there is no client need notion), the composition consists in performing the product of the whole FSM specification. The composition problem consists then to find paths in the computed cartesian product that satisfies reachability properties. According to our work, this work does not cater for timed properties when building a composition. Moreover, in [6], the authors do not deal with the problem of missing messages, since they do not consider data and communications capabilities as in our framework. In addition, our composition approach is oriented by the client need, defined upon the required data flow, that allows to optimize the cartesian product: we compose only the relevant parts of the services and not the whole services.

In [9], Web services exchange asynchronous messages and they are modelled as Mealy machines. The authors investigate an approach dealing with the unexpected interactions between the local and global behavior of composite Web services. However, only messages without parameters are considered. Moreover, the authors are not concerned with how composing services but they are interested in analyzing the local and global behavior of Web services in a composition. Furthermore, the authors do not deal with timed properties.

An other remark is that, works that consider the control flow, address the composition problem at process level, i.e., they consider the operations the services perform [4, 5, 8, 17]. For example, in [4], one of the important assumptions is that the client need (called *goal service*) is specified upon the operations of the services. The precise specification of the goal service allows for precise matching with available, more elementary services. Nevertheless, in real life scenarios, it is not always possible for a client to precisely specify his need according to the operations of the services. A simple client does not have any preliminary knowledge about the service operations.

Whilst, in our framework, the client need (client service) is specified by the (input and output) data the client expects. Moreover, in [4], the authors do not deal with timed properties when composing services.

The few frameworks that deal with timed properties in Web services specification, focus on compatibility and replaceability analysis [27] and timed model checking a given composition [18]. In both works, the authors consider synchronous Web services. While, in our work, we deal with asynchronous Web services. Furthermore, these works do not deal with the composition synthesis problem of asynchronous timed services. For instance, in [18], the authors assume that the composition is already built.

In [7], the authors focus on the interoperability problem of networked systems where they consider non-functional properties such as the response time (e.g., a consumer who asks for photos must get a list of photos in less than x ms). This work is part of the Connect Integrated Project which aims at enabling continuous composition of networked systems [3]. The non functional properties the author consider are simple and are associated to atomic systems (analogously simple services) which must be connected (analogously composed). Moreover, the approach proposed in [7] aims at monitoring the connected system to check that the non-functional properties such as response time are respected. In our work, we consider timed properties associated to complex services and we handle the problem of building compositions so that timed properties of the involved services are analysed to detect and prevent timed conflicts.

16.8 Conclusion and Perspectives

In this chapter, we present a formal approach to handle timed properties in asynchronous Web services composition. Our framework is oriented by the client data flow. To reach this goal, we first propose a timed automata based formal model of timed conversational protocols. This model provides an operational semantic to consider timed properties of asynchronous communicating Web services. This model gathers: (1) *supported messages*, (2) *data*, (3) *constraints over data*, (4) *timed constraints*, and (5) the *asynchronous conversational aspect* of Web services. Based on this model, we provide an algorithm which aims at building a composition so that no timed conflict arises. In this context, we use *the clock ordering process* that allows to discover implicit timed conflicts that can arise when composing services.

Unfortunately, due to the heterogeneous nature of Web services, timed P2P connections can fail, and the composition too. To tackle this problem, we propose to generate a third party service, called *mediator*. The role of this latter is to avoid conflicts. Obviously, the mediator has a crucial role when composing services, since it contributes to connect the required services by producing the expected messages.

The proposed approach has been implemented in a prototype, which has been used to perform preliminary experiments. Currently, we are trying to carry out fine grained experimentations on a set of richer services.

The framework we presented in this chapter focuses on the composition of timed asynchronous services and considers correct interactions of services. Our ongoing work studies the problem of exceptions handling within the timed composition framework. Moreover, we plan to extend our approach with semantic capabilities in order to support more complex timed properties. This will allow us to construct a composition not only by considering timed properties associated to message exchanges, but also more global constraints.

Another interesting research direction consists in studying dynamic substitution in order to resolve timed conflicts which can be complementary with a mediator based approach. In addition, we plan to extend our approach to support dynamic instantiation when composing timed Web services. In this chapter, we assume that only one instance of each service is required. However, in real scenarios, we can need one or several instances of each service. So, it is interesting to extend the proposed approach to handle such features.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994)
2. Benatallah, B., Casati, F., Grigori, D., Nezhad, H.R.M., Toumani, F.: Developing adapters for web services integration. In: CAiSE, pp. 415–429 (2005)
3. Bennaceur, A., Cavallaro, L., Inverardi, P., Issarny, V., Romina Spalazzese Daniel Sykes, M.T.: Dynamic connector synthesis: revised prototype implementation. In: Deliverable D3.3 ICT FET IP Project (2012)
4. Berardi, D., Calvanese, D., Giacomo, G.D., Hull, R., Mecella, M.: Automatic composition of transition-based semantic web services with messaging. In: Proceedings of the 31st International Conference on Very Large Data Bases, pp. 613–624. ACM, 30 August 2005–2 September 2005
5. Berardi, D., Calvanese, D., Giacomo, G.D., Mecella, M.: Composition of services with nondeterministic observable behavior. In: Service-Oriented Computing—ICSOC 2005, Third International Conference (ICSOC), pp. 520–526 (2005)
6. Bertoli, P., Pistore, M., Traverso, P.: Automated web service composition by on-the-fly belief space search. In: Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, ICAPS 2006, pp. 358–361 (2006)

7. Bertolino, A., Inverardi, P., Issarny, V., Sabetta, A., Spalazzese, R.: On-the-fly interoperability through automated mediator synthesis and monitoring? In: 4^{th} International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA'10) (2010)
8. Brogi, A., Popescu, R.: Towards semi-automated workflow-based aggregation of web services. In: Service-Oriented Computing—ICSOC 2005, Third International Conference (ICSOC), pp. 214–227 (2005)
9. Bultan, T., Fu, X., Hull, R., Su, J.: Conversation specification: a new approach to design and analysis of e-service composition. In: Proceedings of the international conference on World Wide Web, WWW 2003, pp. 403–410 (2003)
10. Charif, Y., Sabouret, N.: An overview of semantic web services composition approaches. Electr. Notes Theor. Comput. Sci. **146**(1), 33–41 (2006)
11. Díaz, G., Cambronero, M.E., Pardo, J.J., Valero, V., Cuartero, F.: Automatic generation of correct web services choreographies and orchestrations with model checking techniques. In: Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW'06), p. 186, 19–25 February 2006
12. Guermouche, N.: Timed interation-aware web service composition (wrtiten in french: Etude des interactions temporises dans la composition de services web). Ph.D. thesis, Nancy university, France (2010)
13. Guermouche, N., Godart, C.: Timed properties-aware asynchronous web service composition. In: Proceedings of the 16^{th} International Conference on Cooperative, Information Systems (CoopIS'08) pp. 44–61, 9–14 November 2008
14. Guermouche, N., Godart, C.: Timed model checking based approach for web services analysis. In: IEEE International Conference on Web Services (ICWS'09) (2009)
15. Guermouche, N., Benbernou, S., Coquery, E., Hacid, M.S.: Privacy-aware web service protocol replaceability. In: IEEE International Conference on Web Services (ICWS'07), pp. 1048–1055, 9–13 July 2007
16. Henzinger, T.A., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic model checking for real-time systems. Inf. Comput. **111**(2), 193–244 (1994)
17. Hull, R., Benedikt, M., Christophides, V., Su, J.: E-services: a look behind the curtain. In: Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), pp. 1–14 (2003)
18. 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)
19. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. Int. J. Softw. Tools Technol. Transf. **1**, 134–152 (1997)
20. McIlraith, S.A., Son, T.C.: Adapting golog for composition of semantic web services. In: Proceedings of the 8th International Conference on Principles and Knowledge Representation and Reasoning (KR'02), pp. 482–496, 22–25 April 2002
21. Mecella, M., Batini, C.: Enabling italian e-government through a cooperative architecture. IEEE Comput. **34**(2), 40–45 (2001)
22. Medjahed, B., Bouguettaya, A., Elmagarmid, A.K.: Composing web services on the semantic web. VLDB J. **12**, 333–351 (2003)
23. Muscholl, A., Walukiewicz, I.: A lower bound on web services composition. In: Proceedings of Foundations of Software Science and Computation Structures (FOSSACS), vol. 4423, pp. 274–287. LNCS (2007)
24. Narayanan, S., McIlraith, S.A.: Simulation, verification and automated composition of web services. In: Proceedings of the International Conference on World Wide Web, WWW 2002, pp. 77–88 (2002)
25. Pichler, H., Wenger, M., Eder, J.: Composing time-aware web service orchestrations. In: Proceedings of the 21st International Conference on Advanced Information, Systems Engineering (CAiSE'09), pp. 349–363, 8–12 June 2009
26. Pistore, M., Marconi, A., Bertoli, P., Traverso, P.: Automated composition of web services by planning at the knowledge level. In: IJCAI, pp. 1252–1259 (2005)

27. Ponge, J., Benatallah, B., Casati, F., Toumani, F.: Fine-grained compatibility and replaceability analysis of timed web service protocols. In: The 26th International Conference on Conceptual Modeling (ER) (2007)
28. Sheth, A.P., Larson, J.A.: Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Comput. Surv. **22**(3), 183–236 (1990)
29. Sohrabi, S., Prokoshyna, N., McIlraith, S.A.: Web service composition via generic procedures and customizing user preferences. In: International Semantic Web Conference, pp. 597–611 (2006)
30. Stollberg, M., Cimpian, E., Mocan, A., Fensel, D.: A semantic web mediation architecture. In: Canadian, Semantic Web, pp. 3–22 (2006)
31. Thakkar, S., Ambite, J.L., Knoblock, C.A.: A view integration approach to dynamic composition of web services. In: Proceeding of 2003 ICAPS Workshop on Planning for Web Services (2003)
32. Thakkar, S., Ambite, J.L., Knoblock, C.A.: A data integration approach to automatically composing and optimizing web services. In: Proceedings of the 2nd ICAPS International Workshop on Planning and Scheduling for Web and Grid Services (2004)
33. Wiederhold, G.: Mediators in the architecture of future information systems. IEEE Comput. **25**(3), 38–49 (1992)