# Timed Conversational Protocol Based Approach for Web Services Analysis

Nawal Guermouche and Claude Godart

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

**Abstract.** Choreography is one of the most important features of Web services. It allows to capture collaborative processes involving multiple services. In this paper, we are interested in analyzing the interoperability of Web services that support asynchronous communications which are constrained by data and timed constraints, using a model checking based approach. In particular, we deal with the compatibility problem. To do so, we have developed a set of abstractions and transformations on which we propose a set of primitives characterizing a set of compatibility classes of Web services. This paper is about the specification and implementation of this approach using the UPPAAL model checker.

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

## 1 Introduction

Web services are the main pillar of the Service Oriented Computing (SOC) paradigm. Based on standard interfaces, they facilitate application-to-application interactions thanks to the notion of *choreography* of message exchanges between services. Such a 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.

It is commonly agreed that in general the interaction of Web services and in particular the compatibility of Web services depends not only on the supported sequences of messages but also on crucial quantitative properties such as timed properties [8,10]. We mean by timed properties the required 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).

Some works have dealt with the problem of compatibility of two services. In [4,3,2], authors consider the sequence of messages that can be exchanged between two synchronous Web services to analyze compatibility of two services.

Considering only the 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]. Another important remark is that in [4], authors consider synchronous Web services. Such assumption is very restrictive since two services can succeed a conversation despite that they do not support the same branching structure.

The compatibility framework presented in [10] considers a more expressive timed constraints model. Although powerful, in some cases, the compatibility framework cannot detect some timed conflicts due to non-cancellation constraints. In fact, the authors deal only with synchronous communicating services and discover timed conflicts based on synchronizing the corresponding timed properties over messages. However, in case of asynchronous Web services, this framework cannot be applied to discover timed conflicts.

In [5], the authors handle the timed conformance problem. The timed conformance problem consists in checking if a given timed orchestration satisfies a global timed choreography. In this framework, the authors deal with timed cost (i.e., the delay) of operations. Our aim is to detect deadlocks that can arise when a set of Web services are interacting altogether. While authors of [5] are not interested in analyzing the compatibility of a choreography but only in checking if a given orchestration conforms to a choreography. So, one of the assumption on which the work presented in [5] relies, is that the choreography does not hold timed conflicts.

Regarding our previous work, [6] presents a framework for analyzing the compatibility of Web services. [6], presents an algorithm to analyze the compatibility of Web services based on the clock ordering process. This work is limited to discover only some kind of timed conflicts that do not consider other eventual timed conflicts that can arise when Web services interact together.

In this paper, we propose an analyzing choreography compatibility framework which is based on our previous work [6,7]. This approach is based on the model checker UPPAAL[1]. In this framework we take into account data flow involved when exchanging messages. Furthermore, we consider constraints over data and timed properties that specify delays concerning message exchanges. By studying the possible impacts of timed properties on a choreography, we remarked that when Web services are interacting together, implicit timed dependencies can be derived from the different timed properties of the different services [6,7]. Such dependencies can give rise to implicit timed conflicts. In order to catch the possible timed deadlocks, we propose a set of model checking based primitives.

More precisely, in order to be able to analyze the compatibility of a set of timed and asynchronous Web services, we propose a set of primitives which consists in: (1) extending the model of conversational protocols proposed in [7,6] to consider together messages, data flow, data constraints, and timed constraints, (2) extending the transformation process we proposed in [7] to allow applying model checking to asynchronous Web services composition analysis, particularly, considering timed properties and data constraints when analyzing asynchronous services, (3) finally, we propose new fine grained asynchronous compatibility classes.

[1] http://www.uppaal.com/

The remainder of the paper is organized as follows. The next section presents how we model the timed behaviour of Web services. In order to be able to handle asynchronous services with the UPPAAL model checker, we present a set of abstractions and transformations. Before concluding, we present our formal choreography compatibility investigations.

## 2 Modeling Timed Behaviour of Web Services

The model we consider is based on timed automata. 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 $m(d1, \ldots, dn)$, or $m(\overline{d})$ for short. These automata are equipped with a set of clocks [1]. 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 satisfied.

A timed constraint is a conjunction of atomic formula that compares the value of a clock $x \in X$, to a positive real constant $a \in \mathbb{R}_{\geq 0}$.

The set of Web services are equipped with a bounded queue to store the incoming messages.

Figure 2 shows the timed conversational protocols of an e-government application we consider. The goal is to manage handicapped pension requests. Such a request involves three Web services: (1) a prefecture service (PS) (2) a health authority service (HAS), and (3) a town hall service (TH). A citizen can apply for a pension. The prefecture solicits the medical entity to examine the requester. On the other side, the prefecture asks the town hall to deliver the domiciliation attestation. After studying the received file, the prefecture sends the notification of the final decision to the citizen after 48 hours and within 96 hours from receiving the pension request. To specify such constraint, we associate a reset of the clock $t_1$ ($t_1 = 0$) to the transition that enables to receive the request of the pension and we associate the constraint $48 \leq t_1 \leq 96$ to the transition that enables to send the final decision.

As the approach we propose is based on the model checker UPPAAL, next, we present the required process we propose to adapt this general model to the UPPAAL one to be able to perform the compatibility cheking.

## 3 From Conversational Protocols to UPPAAL Timed Automata

UPPAAL is a model checker for the verification and simulation of real time systems. An UPPAAL model is a set of timed automata, clocks, channels for systems (automata) synchronization, variables and additional elements [9,7].

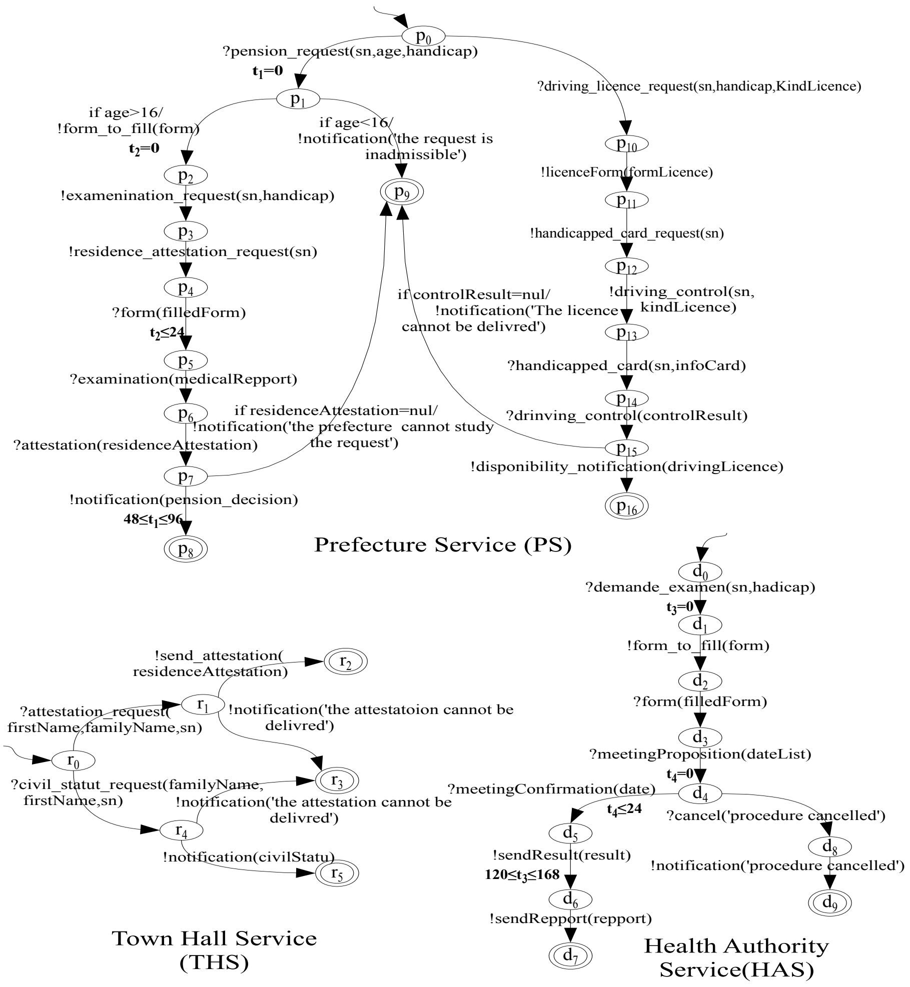
$p_0$
?pension_request(sn,age,handicap)
$t_1$=0
?driving_licence_request(sn,handicap,KindLicence)
$p_1$
if age>16/ !form_to_fill(form)
$t_2$=0
if age<16/ !notification('the request is inadmissible')
$p_{10}$
!licenceForm(formLicence)
$p_2$
!examenination_request(sn,handicap)
$p_9$
$p_{11}$
!handicapped_card_request(sn)
$p_3$
!residence_attestation_request(sn)
$p_{12}$
$p_4$
if controlResult=nul/ !notification('The licence cannot be delivred')
!driving_control(sn, kindLicence)
?form(filledForm)
$t_2 \leq 24$
$p_{13}$
$p_5$
?handicapped_card(sn,infoCard)
?examination(medicalRepport)
$p_{14}$
$p_6$
if residenceAttestation=nul/ !notification('the prefecture cannot study the request')
?drinving_control(controlResult)
?attestation(residenceAttestation)
$p_{15}$
$p_7$
!disponibility_notification(drivingLicence)
!notification(pension_decision)
$48 \leq t_1 \leq 96$
$p_{16}$
$p_8$
Prefecture Service (PS)
$d_0$
?demande_examen(sn,hadicap)
$t_3$=0
$d_1$
!send_attestation( residenceAttestation)
$r_2$
!form_to_fill(form)
$r_1$
$d_2$
?attestation_request( firstName,familyName,sn)
!notification('the attestatoion cannot be delivred')
?form(filledForm)
$d_3$
$r_0$
?meetingProposition(dateList)
?civil_statut_request(familyName, firstName,sn)
$r_3$
$t_4$=0
?meetingConfirmation(date)
$d_4$
!notification('the attestation cannot be delivred')
$t_4 \leq 24$
?cancel('procedure cancelled')
$r_4$
$d_5$
$d_8$
!notification(civilStatu)
$r_5$
!sendResult(result)
!notification('procedure cancelled')
$120 \leq t_3 \leq 168$
$d_6$
$d_9$
!sendRepport(repport)
Town Hall Service (THS)
$d_7$
Health Authority Service(HAS)

**Fig. 1.** Web services of the e-government scenario

In order to perform a model checking using UPPAAL, we use a set of transformation steps, which are: (1) Messages abstraction, (2) data constraints abstraction, (3) final states specification. The steps of messages abstraction and final states specification are presented in [7].

### 3.1 Abstraction of Data Constraints

As said previously, our model considers constraints over data. To analyze these constraints with UPPAAL, the values of the data must be known. However, as the compatibility analysis we propose is done at design time, the values of the data cannot be known in advance. Consequently, the constraints over data cannot be correctly considered. To consider constraints over data, we propose to abstract the variables of messages resulting from the process of message abstraction described in [7] with information about data constraints. The idea is to compute the set of transitions that hold the same variable. If the set of solutions of the data constraints associated to these transitions is disjoined, then we abstract differently the variables. Whilst, if the set of solution is not disjoined, then we remove only the data constraints without changing the variables. To explain this issue, let us present the following example.

*Example 1.* Via this example, we are going to show how we apply the data constraints abstraction process. As we can see, the two services, illustrated in Figure 2, have the following two common transitions (i.e., transitions that hold the same variable):

– $(s_0, m_0++, d_0 < 100, s_1)$ and $(p_1, m_0++, d_0 > 120, p_2)$
– $(s_2, m_{21}--, m_{21} > 0, d_1 < 50, s_3)$ and $(p_0, m_{21}++, d_1 < 80, p_1)$

Let us start with the first pair of transitions $(s_0, m_0++, d_0 < 100, s_1)$ and $(p_1, m_0++, d_0 > 120, p_2)$. We can remark that the set of solutions of the constraints $d_0 < 100$ and $d_0 > 120$ is disjoint, i.e., $Sol(d_0 < 100) \cap Sol(d_0 > 120) = \emptyset$. Hence, by applying the data constraints abstraction process, we substitute $m_0++$ of the transition $(p_1, m_0++, d_0 > 120, p_2)$ by another variable $m_0'$. So the transition becomes $(p_1, m_0'++, p_2)$

Now, we check the second pair of transitions $(s_2, m_{21}--, m_{21} > 0, d_1 < 50, s_3)$ and $(p_0, m_{21}++, d_1 < 80, p_1)$. We can see that the set of solutions of data constraints $d_1 < 50$ and $d_1 < 80$ is not disjoint. The two constraints have a common set of solutions, i.e., $Sol(d1 < 50) \cap Sol(d1 < 80) \neq \emptyset$ . Consequently, when abstracting data constraints, we do not substitute the variable $m_{21}$.

The result of the transformation steps we described above is a set of abstract UPPAAL timed automata. These automata preserve the semantic we consider in timed conversational protocol of asynchronous services.

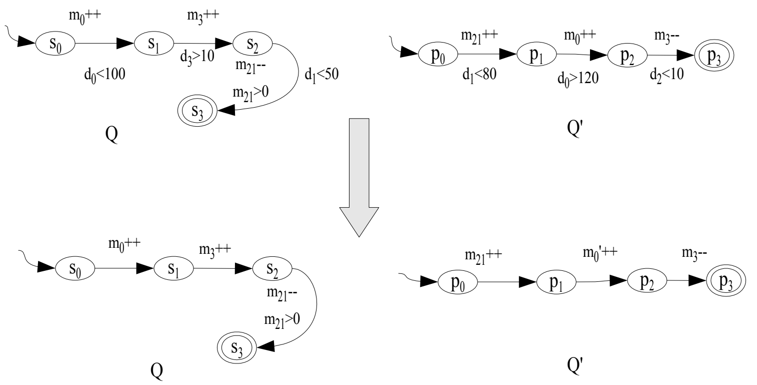

**Fig. 2.** Abstraction of data constraints

Next, we present the formal primitives we propose to characterize the compatibility class of a set of timed asynchronous Web services.

## 4 Formal Asynchronous Compatibility Checking

Based on the previous transformations, we present in this section a compatibility checking using UPPAAL. We define the following fine grained timed asynchronous compatibility classes: (1) *full and perfect compatibility*, (2) *full but non-perfect compatibility*, (3) *perfect partial compatibility*, (4) *partial but non-perfect compatibility*, and (5) *full incompatibility.*

### 4.1 Perfect Full Compatibility

In general, a set of Web services constitute a full compatible choreography if they can interact without an eventual blocking. As we deal with asynchronous services, the output messages are sent without synchronization with the corresponding input. Thus, it is not sufficient to check only if there is no deadlock when services interact together, but, it is important too to check if all the sent messages are consumed. So, a set of services constitute a full and perfect compatible choreography if: (1) they collaborate together without any eventual blocking and (2) at the same time, all the generated messages are consumed.

Formally, checking if a set of Web services interact without an eventual blocking is equivalent to checking if the services reach their final states in all interactions. At the same time, when the services reach their final states, the fact that all the sent messages must be consumed is formally equivalent to checking that, when the services reach their final states, all the variable values are equal to zero.

Let $P_1, \ldots, P_n$ be $n$ (asynchronous) services and $R_1, \ldots, R_n$ be the corresponding set of variables. The set of fully and perfect compatible Web services is specified by the following CTL formulas:

$$\boxed{\begin{array}{l} AF\ P_1.final \wedge \ \ldots \ \wedge P_n.final \wedge AG\ (P_1.final \wedge \ \ldots \ \wedge P_n.final \Rightarrow AF \\ r_1 == 0 \text{ and } \ldots \wedge r_m == 0) \text{ where } r_i \in \{R_1, \ldots R_i, \ldots, R_n\} \end{array}} \quad (1)$$

### 4.2 Non-perfect Full Compatibility

When a set of Web services can collaborate together without an eventual blocking but at the same time, during their interaction, there are messages that cannot be consumed, i.e., extra messages, we say that the services are fully but non-perfectly compatible.

Formally, a set of Web services are said to be fully and non-perfectly compatible if via all the paths, the services reach their final state and at the same time, and in this state, there exists at least one variable whose value is bigger than zero. This latter can be specified by the following CTL formulas:

$$\boxed{\begin{array}{l} AF(P_1.final \wedge \ldots \wedge P_n.final) \wedge EF(P_1.final \wedge \ldots \wedge P_n.final \Rightarrow r_1 > 0 \vee \ldots \vee r_n > 0) \\ \text{where } r_i \in \{R_1, \ldots R_i, \ldots, R_n\} \end{array}} \quad (2)$$

## 4.3 Partial But Non-perfect Compatibility

As services are heterogeneous they can fulfil incorrect conversations. A conversation during which a service remains blocked is called incorrect. A set of Web services are not fully compatible when the set of possible conversations of the services hold at least one incorrect conversation.

Formally, a set of Web services are not fully compatible if there exists at least a path of their automata that cannot reach the final state. This later can be specified as the following formula:

$$\boxed{EG\ \neg P_1.final\ \vee\ \ldots\ \vee EG\ \neg P_n.final} \quad (3)$$

When a set of Web services can achieve correctly a set of conversations and at the same time they fail other conversations, we say that the services are partially compatible. In this section, we define particularly the partial but non-perfect compatibility class. This class is assigned to a set of Web services that are partially compatible and at the same time, there is at least one correct conversation during which there is at least one extra message. This is formally specified by the following CTL formulas:

$$\boxed{\begin{array}{l} EF\ P_1.final\ \wedge\ \ldots\ \wedge\ P_n.final\ \wedge \\ (EF\ P_1.final\ \wedge\ \ldots\ \wedge\ P_n.final \Rightarrow r_1 > 0\ \wedge\ \ldots\ \wedge\ r_m > 0) \text{ where } r_i \in \\ \{R_1, \ldots R_i, \ldots, R_n\} \end{array}} \quad (4)$$

Formally, a set of Web services is said to be partially but non-perfectly compatible if the formulas (3) and (4) are satisfied.

## 4.4 Partial and Perfect Compatibility

The partial and perfect compatibility class characterizes the fact that services are not fully compatible but at the same time, they can fulfil correctly conversations during which all the produced messages are consumed.

Formally, a set of Web services can achieve correctly at least one conversation so that all produced messages are consumed is equivalent to checking that there exists at least one path so that all the services reach their final state and at the same time, when the final state is reached, the value of all variables is equal to zero. This is specified by the following CTL formula:

$$\boxed{\begin{array}{l} EF\ P_1.final\ \wedge\ \ldots\ \wedge\ P_n.final\ \wedge \\ (EF\ P_1.final\ \wedge\ \ldots\ \wedge\ P_n.final \Rightarrow r_1 == 0\ \wedge\ \ldots\ \wedge\ r_m == 0) \text{ where } r_i \in \\ \{R_1, \ldots R_i, \ldots, R_n\} \end{array}} \quad (5)$$

A set of Web services whose conversational protocols do not verify the formula 4 and at the same time, verify the formulas 3 and 5 is said to be partially and perfectly compatible.

### 4.5 Full Incompatibility

Full incompatibility characterizes the fact that the set of services cannot, absolutely, collaborate together. Formally, a set of services are fully incompatible, if for all the paths, all the services cannot reach the state 'final'. This property is specified as the following CTL formulas:

$$\boxed{AG \neg P_1.final \ \wedge \ \dots \ \wedge \ AG \neg P_n.final} \tag{6}$$

## 5 Conclusion

In this paper, we presented a formal framework for analyzing the timed compatibility of a choreography. Unlike the existing approaches, this framework caters for timed properties and data constraints of asynchronous Web services. In order to handle timed and data constraints deadlocks, we proposed an approach based on the model checker UPPAAL. The model checker UPPAAL does not take into account constraints over data semantics. In order to handle asynchronous services augmented with data flow and data constraints, we proposed to extend our previous work [7] by the data constraints abstraction process. By using the result of the abstractions, we presented a set of CTL formulas that characterize the different choreography compatibility classes we have defined.

In our ongoing 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., Toumani, F.: Analysis and management of web service protocols. In: Atzeni, P., Chu, W., Lu, H., Zhou, S., Ling, T.-W. (eds.) ER 2004. LNCS, vol. 3288, pp. 524–541. Springer, Heidelberg (2004)
3. Benatallah, B., Casati, F., Toumani, F.: Web service conversation modeling: A cornerstone for e-business automation. IEEE Internet Computing 8(1), 46–54 (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. Eder, J., Tahamtan, A.: Temporal conformance of federated choreographies. In: Bhowmick, S.S., Küng, J., Wagner, R. (eds.) DEXA 2008. LNCS, vol. 5181, Springer, Heidelberg (2008)
6. Guermouche, N., Godart, C.: Asynchronous timed web service-aware choreography analysis. In: van Eck, P., Gordijn, J., Wieringa, R. (eds.) CAiSE 2009. LNCS, vol. 5565, pp. 364–378. Springer, Heidelberg (2009)
7. Guermouche, N., Godart, C.: Timed model checking based approach for web services analysis. In: IEEE International Conference on Web Services (ICWS 2009), Los Angeles, CA, USA, July 6-10, pp. 213–221 (2009)
8. 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)
9. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. International Journal on Software Tools for Technology Transfer (1997)
10. 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)