Timed Specification For Web Services Compatibility Analysis

Nawal Guermouche[1]

LORIA, INRIA Lorraine, Campus scientifique
BP 239, 54506 Vandoeuvre-Lès-Nancy

Olivier Perrin[2]

LORIA, INRIA Lorraine, Campus scientifique
BP 239, 54506 Vandoeuvre-Lès-Nancy

Christophe Ringeissen[3]

LORIA, INRIA Lorraine, Campus scientifique
BP 239, 54506 Vandoeuvre-Lès-Nancy

Abstract

Web services are becoming one of the main technologies for designing and building complex inter-enterprise business applications. Usually, a business application cannot be fulfilled by one Web service but by co-ordinating a set of them. In particular, to perform a coordination, one of the important investigations is the compatibility analysis. Two Web services are said *compatible* if they can interact correctly. In the literature, the proposed frameworks for the services compatibility checking rely on the supported sequences of messages. The interaction of services depends also on other properties, such that the exchanged data flow. Thus, considering only supported sequences of messages seems to be insufficient. Other properties on which the services interaction can rely on, are the temporal constraints. In this paper, we focus our interest on the compatibility analysis of Web services regarding their (1) supported sequences of messages, (2) the exchanged data flow, (3) constraints related to the exchanged data flow and (4) the temporal requirements. Based on these properties, we study three compatibility classes: (i) *absolute compatibility*, (ii) *likely compatibility* and (iii) *absolute incompatibility*.

Keywords: compatibility analysis, Web services, temporal constraints.

1 Introduction

Web services are one of the main technologies adopted by organizations to build their business applications. Generally, a Web service is a set of related functionali-

---

ties that can be programmatically accessed through the Web. These functionalities represent the different operations made available by the Web service and are described in its service description using the WSDL standard language [10]. Autonomy and heterogeneity of services induce several interoperability problems. The interoperability is the key for several issues such as the service composition which involves the synthesis problem (i.e., how to coordinate the component services to fulfill a given goal) [1]. In this context, the *compatibility analysis* plays a crucial role. This analysis consists in checking if a client and a provider can properly interact.

The interaction between a client and a provider depends not only on functionalities the provider can supply, but also on other properties representing requirements of each of them. Hence, service descriptions must be augmented by all the aspects that can represent the client's and the provider's requirements. Such enriched descriptions are useful to keep services informed if they can interact together by considering their requirements. For the best of our knowledge, all the related compatibility checking frameworks consider only the supported messages sequences [4,8]. Other works consider some forms of temporal constraints that cannot express all the service's temporal requirements [3].

Since the exchanged messages can involve data, considering only the supported sequences of messages seems not sufficient. Thus, considering data flow in Web services analysis, and especially in the compatibility analysis is important. Moreover, to receive or send a message, a Web service may express some requirements. Among these requirements, we focus on data and time related conditions. For example, a service that allows us to book a flight ticket (TB service) can supply an electronic ticket if the *credit card number ccn* of the client *is valid*. Moreover, the TB service cannot send the electronical ticket before 60 minutes starting from the payment.

Regarding the temporal aspects, a service can have its own requirements associated to its internal running. Such requirements are called *internal constraints*. For example, if a client that books a flight ticket does not send its credit card number (*ccn*) within 30 minutes, the service cancels. The internal constraints can be, implicitly, mutually dependent. Thus implicit incompatibilities can arise. To tackle this problematical issue, we propose to infer from the internal constraints, the potential implicit constraints. Those inferred constraints can be exposed to the other services, thus we call such constraints *external constraints*. For example, once the client sends its *ccn*, the provider must supply the electronic ticket within 1 hour.

In this paper, we present a timed compatibility analysis framework that deals with the *implicit incompatibilities* that can arise. In our approach, Web services are modeled as finite state machines. This kind of formal representation has been already used in a series of papers [9,3,5,15]. The model we consider in this paper integrates multiple aspects to have a fine-grained abstraction of Web services [13]. Theses aspects are (a) *operations*, which can require input parameters and provide output parameters, (b) *message* passing exchanged via channels able to hold at most one message at a time, (c) *data* that are parameters of the messages and

operations, (d) constraints over these data, (e) temporal constraints, and (f) the behavior of Web services (which may involve multiple operations and message-passing activities) is specified using guarded finite state transition systems. The data used by Web services are managed via information systems. Services can get data by exchanging messages or by reading the information systems. Furthermore, the services can change the values of data in the information systems. This allows us to deal with constraints over counters. For example, if the client provides a wrong *ccn* successively three times, beyond the third attempt the system cancels. To express such requirements, we may consider counters that can be *incremented*, *decremented* or *reset* once a transition is triggered. To express temporal constraints, we rely on clocks as defined in standard timed automata [2].

The paper first recalls the model we rely on in Section 2. Section 3 presents the motivating example used to illustrate our model. The different forms of temporal constraints we define are described in Section 4. In Section 5, we show how we integrate the different forms of temporal constraints by extending to the Web Services Timed Transitions Systems (WSTTS) [15]. In section 6, we present the compatibility analysis process of Web services regarding their different constraints (constraints over data and temporal constraints). Related work is introduced in Section 7, and Section 8 concludes.

2 Overview

In this paper, we enhance with temporal constraints the model of Web services automatic composition introduced in [13]. In the following section, we introduce the model we proposed.

2.1 The model

In our earlier framework [13], Web services are represented as conversational automata. In these automata, the alphabet consists of actions and exchanges of messages. A message is either sent or received. An output message is denoted by $!m$, whilst an input one is denoted by $?m$. A message containing a list of parameters is denoted by $m(d_1, \ldots, d_n)$, or $m(\bar{d})$ for short. There are many papers dealing with conversational automata to represent web services, including the *Colombo* model [6]. Our model extends the Roman model [7,5] by introducing communication and data capabilities, and can be considered as an extension of the conversational model introduced in [9], where data are exchanged via messages, and can be used to express guards of transitions. An action a with a list of input parameters $d_1, \ldots, d_m$ and list of output parameters $d'_1, \ldots, d'_n$ is denoted by $a(d_1, \ldots, d_m; d'_1, \ldots, d'_m)$. Throughout the paper, we only consider "well-formed" Web services, in which we assume that, when a parameter is used, possibly in a guard, it has been previously initialized. There are three ways to initialize a parameter: either it is the output of a previous action, it is retrieved via a web service exchange of message, or via reading the information system. Constraints used in guarded transitions can be defined

as conjunctions of atomic formulas $d \bowtie val$, where val is a possible value for the parameter d, and $\bowtie$ is binary relation to compare the value of d and val.

Definition 2.1 A *Web service* Q is a tuple (S, s_0, F, M, A, C, 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, A is a set of actions, C is a set of constraints, and the set of transitions $T \subseteq S \times (M^{?!} \cup A) \times C \times S$, such that, from a source state, the service exchanges a message or performs an action in A, with a constraint to reach a target state, where $M^{?!} = \{?m | m \in M\} \cup \{!m | m \in M\}$ is the set of input/output messages. An input (resp. output) transition is a transition of the form $(\_, ?m, \_)$ (resp. $(\_, !m, \_)$).

3 Motivating example

To illustrate our model, we consider a simple application for an online flight ticket booking. This application is carried out by two Web services: the *ticket booking service* (*TB*) and the *payment by credit card* service (*PCC*). The requester specifies the destination (city) and the date of the travel. Then the *TB* service suggests a list of tickets description (price, class, timing). The requester makes a choice, and he/she provides a credit card number *ccn* to complete the booking. Then the *TB* service invokes the (*PCC*) service to check the credit card number (*ccn*) validity in order to guarantee the payment.

   The *TB* service has several requirements such as:

  (i) once *TB* service sends the description list of tickets, if the client does not confirm the travel booking by providing his *ccn* in 1800 seconds, the ticket booking is cancelled.

 (ii) if the client provides a wrong *ccn* successively three times, beyond the third attempt the system cancels.

(iii) if the client books a ticket during the *Christmas holiday*, i.e., in the period [25 *december*, 05 *january*], *TB* service performs a rebate on the price of tickets.

   To establish a conversation between the two services, requirements that are strongly bound to temporal features must be checked. Thus, we believe that Web services description interfaces (for example WSDL interface) must be enhanced by *time-related properties*. The time-related properties can have several forms. In the following sections, we detail the issues related to such properties.

4 Temporal constraints

From our point of view, each service can have *time-related requirements* to express the internal process of the service. Such requirements are called *internal constraints*. In order to establish an interaction between a client service and a provider service, each one of them exposes to the other requirements that must be checked before initializing the interaction. Such exposed requirements are called *external constraints*.
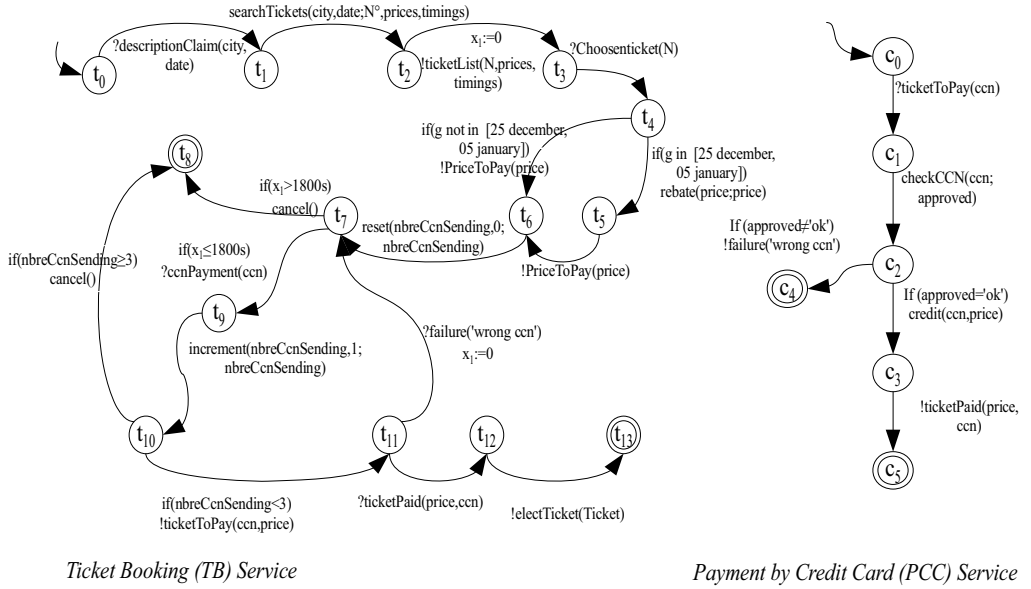
Fig. 1. The online flight booking ticket.

4.1   Internal constraints

The internal constraints are specified when the Web service is designed. They relate for example to the temporal requirements needed to exchange messages and fulfill an operation. Especially, an internal constraint allows to express the fact that triggering a transition may depend on other timed transitions. In our work, we distinguish two kinds of internal constraints: (i) *activation constraints* that correspond to the cancellation transitions and (ii) *dependency constraints* used to specify requirements related to ordinary transitions (i.e., non-cancellation ones).

Example 4.1 To illustrate an activation constraint over a sequence of transitions, we consider the requirement (i) of the motivating example which is: *once the TB service sends the description list of tickets, if the client does not confirm the travel booking by providing his ccn in 1800 seconds, the tickets booking will be cancelled.*

To model such feature, we use the standard clocks of timed automata [2]. As seen in Fig. 1, we reset a clock x_1 with the transition that sends the ticket description ($!ticketList(N, price, timing)$), and we guard the transition that cancels the booking with a temporal constraint $x_1 > 1800s$.

Furthermore, an internal constraint can be specified over a period that can be expressed via a global clock providing the absolute time. An example of such constraints is presented by the requirement (iii): *if the client books a ticket during the christmas holiday, i.e., the period* [25 *december*, 05 *january*], *TB service performs a rebate on the price of tickets.* To be able to express such time-related requirements, we propose to use temporal constraints over a *global clock*, which cannot be reset in transitions.

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 $\bowtie \in \{\leq, <, =, \neq, >, \geq\}$, $x \in X$, and c is a constant.

The internal constraints can also have another form, such as depending on the number of times[4] transitions can be triggered. Back to the requirement (ii) of the motivating example, the system must cancel after the third wrong attempt to provide the *ccn*. To consider such a feature in the Web services model, we propose to enhance the Web services specification with constraints over *counters*. To manage such constraints and for the sake of simplicity of the model, we propose to consider counters as data that can be stored in the information system. Those data (counters) can be *incremented*, *decremented* and *reset*.

To summarize, internal constraints are expressed via (1) *local clocks* used in timed automata theory [2] to specify the relative period in which a transition must be triggered, and (2) a *global clock* to specify the absolute date (or period) in which a transition must be triggered, and (3) constraints over data such that constraints over *counters* to specify the number of times in which a transition must be triggered.

The internal constraints related to the cancellation transitions are called *activation constraints*, whilst those related to non-cancellation transitions are called *dependency constraints*.

4.2   External constraints

To initialize a conversation between a client service and a provider service, each one can expose requirements that must be checked. As presented above, each service has its internal constraints. So we propose to infer from those internal constraints the external constraints, that can be exposed to the other services. For example, we can infer from the internal constraints of the *TB* service, depicted in Fig. 1, that *TB* cancels after at least $1800s$.

In Section 6.2, we illustrate how we infer external constraints from internal ones, and we will show the importance of the inference of the external constraints in the compatibility checking.

An external constraint can be defined as: *Once pre-conditions are satisfied, an input/output message must be performed in a given period.*

Definition 4.2 An external constraint e is a tuple (p, ϱ, d), where ϱ denotes an input/output message in $M^{?!}$, p is the pre-condition that must hold to perform ϱ, such that d is the period in which ϱ can be performed. If ϱ is an input (resp. output) message, the external constraint is called an *input* (resp. *output*) external constraint.

In the following, we restrict our attention to two forms of preconditions:

---

[4] carrying out an operation or exchanging a message a specific number of times

- A precondition equals to *true*, which means that ϱ must be performed in a period starting from the invocation of the service (see Section 6.2.2).

- A precondition equals to an *input/output* message, which means that ϱ must be performed in a period starting from the exchange of this message (see Section 6.2.1).

Example 4.3 The client can claim to receive the electronic ticket not later than 60 minutes after sending his *ccn*.

Once the CCN is sent (!ccnPayment(ccn)), the electronic ticket must be received (?electTicket(ticket)) in 60 minutes ([0, 3600s]).

In this example, the related external constraint of the client is (!*ccnPayment*(*ccn*), ?*electTicket*(*ticket*), [0, 3600*s*]).

To summarize, as shown in Fig. 2, we distinguish (1) *internal constraints* and (2) *external constraints*. Internal constraints can be specified by (i) *temporal constraints* and (ii) *constraints over data*. The temporal constraints can be specified over *local clocks* and *a global clock*. The local clocks rely on standard timed automata clocks [2], whereas, the global clock relies on an absolute time. The constraints over data involve *parameters* of the exchanged messages, or *counters*. The internal constraints specify the activation constraints (cancellation) and *dependency* constraints (non-cancellation). The external constraints are constraints exposed by the client and the provider service, that must be checked before initializing the interaction. Those external constraints are inferred from the internal constraints (see Section 6.2).
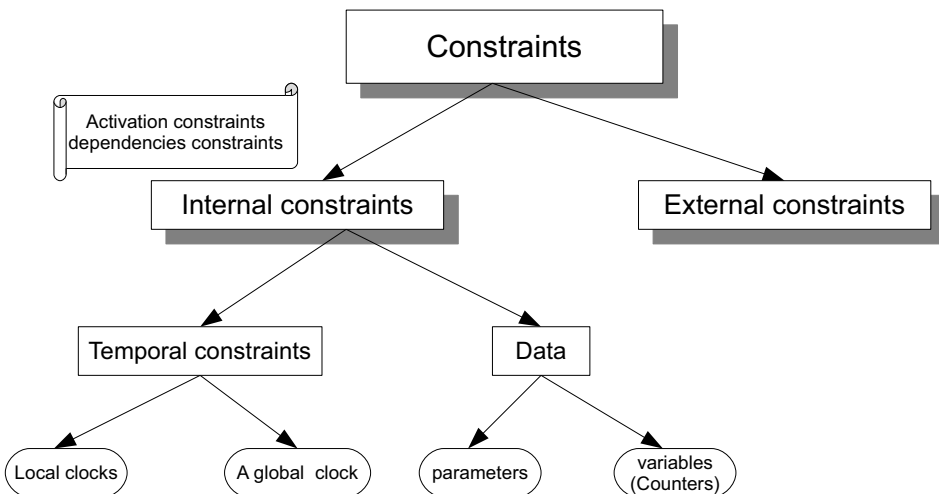

Fig. 2. Constraints specification.

5   Web services timed conversations

In this section, we show how we integrate temporal constraints presented above into web service specifications. To this aim, we propose to extend Web Service Timed Transition Systems (WSTTS) which are timed conversational automata [15]. A WSTTS is a finite state machine equipped with a set of clock variables and transitions guarded by constraints over clock variables. WSTTS use the standard form of constraints used in timed automata [2]. To consider constraints presented previously, WSTTS are replaced by *Extended Web Service Timed Transition Systems* (EWSTTS). In addition, in EWSTTS we include data capabilities used in our previous work [13], that are not considered in WSTTS. For the data management, we use information systems which are relational structures. In the following, we introduce the information systems and EWSTTS.

5.1   *Information system*

An information system is used to manage data. It is characterized by a set of objects defined by a set of attributes that can change their values by performing three atomic operations: *add*, *update*, and *delete*. Information can also be *read*. These atomic operations are effects of actions performed by services. To enforce constraints over counters, we consider a set of data that represent the counters. The value of counters can be *incremented*, *decremented* or *reset*.

5.2   *Extended Web Service Timed Transition Systems (EWSTTS)*

EWSTTS introduces several aspects such that: (1) data flow capabilities, (2) constraints over data, (3) a global clock and (4) constraints over counters. Moreover, in WSTTS there are no final states, whereas in the Extended WSTTS we propose (EWSTTS), we consider final states (component F) as usual in timed automata. An EWSTTS is a finite-state machine in which a transition performs an action having an effect on the information system, send or receive a message. Services can get data from the information systems. By sending and receiving messages, services can also exchange data. An EWSTTS is equipped with a set of clocks and data variables, and transitions are guarded by constraints over *clock* and *data* variables. We consider two kinds of data: (i) parameters of messages/operations, and (ii) counters that allow us to express constraints. Transitions are labeled by *constraints over data*, *timed constraints* and resets of *local clocks*.

Definition 5.1 An EWSTTS is a tuple $P = (S, s_0, F, M, A, C, X, T)$ such that

- S is a set of states, s_0 is the initial state and $F \subseteq S$ is a set of final states.
- A is a set of actions.
- M is a set of messages.
- C is a set of constraints over data (including counters).
- X is the set of clocks (local clocks and a global clock).

- A set of transitions $T \subseteq S \times (M^{?!} \cup A) \times C \times \Psi(X) \times 2^{X \setminus \{g\}} \times S$, with an element of the alphabet (action or an exchanged message), a constraint over data, a guard over clocks, and the clocks, except the global clock (g), to be reset.

We define now the extended semantic of WSTTS [15]. The semantic of an EWSTTS is defined using a transition relation over configurations made of a state, a clock valuation and a data valuation. The clock valuation is a mapping $u : X \to \mathbb{T}$ from a set of clocks to the domain of time values. The data valuation is a mapping $v : D \to \mathbb{V}$ from a set of data D to the domain of data values. The mapping u_0 denotes the (initial) clock valuation, such that $\forall x \in X \setminus \{g\}, u_0(x) = 0$. The mapping v_0 denotes the (initial) data valuation.

A service remains at the same state s without triggering a transition when the time increments, if there is no transition (s, a, c, ψ_X, Y, s') such that the constraints c over data, and the temporal constraints ψ_X are both satisfied, where $\psi_X \subseteq \Psi(X)$. A service moves from state s to state s' by triggering a transition $t = (s, a, c, \psi_X, Y, s')$ if the constraints c and ψ_X are satisfied.

Definition 5.2 (Semantic of EWSTTS)
Let $P = (S, s_0, F, M, A, C, X, T)$ be a EWSTTS. The semantic is defined as a labeled transition $(\Gamma, \gamma_0, \to)$, where $\Gamma \subseteq S \times V_T \times V_C$ is the set of configurations, such that V_T is a set of temporal valuations, V_C is a set of data valuations, $\gamma_0 = (s_0, u_0, v_0)$ is the initial configuration, and $\to$ is defined as follows:

- *Elapse of time*: $(s, u, v) \overset{\text{tick}}{\to} (s, u + d, v)$
- *Location switch*: $(s, u, v) \overset{a}{\to} (s', u', v')$, if $\exists t = (s, a, c, \psi_X, Y, s')$ such that $v \wedge c$ and $u \wedge \psi_X$ are satisfiable and $\forall y \in Y, u'(y) = 0$, $\forall x \in X \setminus Y, u'(x) = u(x)$, where $Y \subseteq X \setminus \{g\}$. Informally, v' is obtained from v by updating data changed by a.

In our context, we assume that *Location switch* is applied eagerly: when both *Elapse of time* and *Location switch* can be applied, *Location switch* is chosen. Notice that a possible use of *Elapse of time* is to precede a *Location switch* in order to represent the cost in time of an operation.

6   Compatibility analysis

Two Web services are said *compatible* if they can interact without been blocked. In the literature, this may happen when a service is waiting for a message that the other service does not send. The compatibility problem was studied in several works [8,4,3,18]. In all these works, the Web services are modeled only by their messages sequences. However, the compatibility analysis should not only rely on the sequences of messages they can exchange. For example, two Web services can support the same sequences of messages but if they do not involve the same data flow, these services are incompatible.

To provide a compatibility checking framework that considers multiple aspects, we model Web services using their sequences of messages, the parameters of the mes-

sages, the constraints over their parameters, the actions, and the aforementioned temporal constraints. By considering parameters of messages and constraints over these parameters, two Web services (a provider and a client) can be not compatible for many reasons: (1) the client (resp. the provider) waits for a message that the provider (resp. the client) does not send, (2) the data flow of the input messages differs from the output message's data flow, (3) the constraints over data or (4) the temporal constraints corresponding to the output and input messages are inconsistent i.e., they have disjoint sets of solutions.

Regarding the constraints over data, two services are compatible if their respective transitions that allow to send and receive the message are consistent. Thus, we need to check if the transitions are consistent. We call this process *the local consistency checking* of transitions. However, according to the temporal constraints, performing a local consistency checking is not always adequate, since the constraints of some transitions can have an impact on the triggering of other transitions. Thus, we propose to infer external constraints that are implicit according to the internal constraints. In the following, we explain the *local consistency checking* of transitions. Then we show how and why the internal constraints are not sufficient for the compatibility analysis of services.

6.1   Local consistency of transitions

Two Web services Q_1 and Q_2 are said compatible if each message sent by the service Q_1 (resp. Q_2) is received by the service Q_2 (resp. Q_1). So the compatibility checking relies on the consistency of the transitions that correspond to the pairs (*input message, output message*). By considering constraints over data and temporal constraints, we distinguish three classes of consistency of two transitions: (1) *absolute consistency*, (2) *likely consistency* and (3) *absolute inconsistency*.

An output transition t_1 of a service Q_1 is said *absolutely consistent* with an input transition t_2 of a service Q_2, if the solutions of constraints (temporal and over data) of t_1 are solutions of constraints of t_2.

For instance, let us suppose a service Q_1 that can send the message m within the interval $[20, 30s]$, and a service Q_2 that can receive the same message within the interval $[10, 60s]$. Since $[20, 30] \subseteq [10, 60]$, once the message is sent, it will be received. Now, consider that m can be sent in $[10, 40s]$, but received only within $[10, 20s]$. If m is sent at $30s$, it cannot be received, and so for this value, transitions are clearly *inconsistent*. On the other hand, if the message is sent at $10s$, it will be received. Hence, for some solution values the transitions are consistent and for some others, they are not, i.e., they are *likely consistent*.

Definition 6.1 An output transition $t_1 = (s_1, !m, c_1, \psi_{X_1}, s_1')$ is *(locally) absolutely consistent* with an input transition $t_2 = (s_2, ?m, c_2, \psi_{X_2}, s_2')$, denoted by $t_1 \sqsubseteq t_2$ if $Sol(c_1) \subseteq Sol(c_2)$ and $Sol(\psi_{X_1}) \subseteq Sol(\psi_{X_2})$, where $Sol(c_i)$ denotes the set of solutions of the constraint c_i related to the data and $Sol(\psi_{X_i})$ denotes the set of solutions of the temporal constraint ψ_{X_i}.

t_1 is *likely consistent* with t_2, denoted $t_1 \subsetneq t_2$ if $t_1 \not\sqsubseteq t_2$ and $Sol(c_1) \cap Sol(c_2) \neq \emptyset$

and $Sol(\psi_{X_1}) \cap Sol(\psi_{X_2}) \neq \emptyset$.

t_1 is *absolutely inconsistent* with t_2 if $Sol(c_1) \cap Sol(c_2) = \emptyset$ or $Sol(\psi_{X_1}) \cap Sol(\psi_{X_2}) = \emptyset$

6.2 External constraints inference

To analyze the compatibility of two Web services, checking the local consistency of internal constraints related to pairs (*input message, output message*) of services is not always sufficient to detect incompatibilities. In fact, the temporal constraints of some transitions of a service can have an impact on other transitions of the same service. To handle this problem, we suggest to verify if the internal constraints of the service are mutually dependent. To do this, we propose to infer from the internal constraints all the potential constraints, called *external constraints*, since they will be exposed to check if they satisfy the exposed requirements of the client service.

To illustrate such a situation, let us use the fragments of two services Q_1 and Q_2 depicted in Fig. 3. According to Definition 6.1, the two transitions $(s_1, ?m_1, x < 10, s_2)$ and $(q_3, !m_1, q_4)$ are *likely consistent*, since solutions of $x < 10$ are non-disjoint with the message sending ones. However, these transitions are problematic since Q_1 must receive the message m_1 before $10s$, whilst Q_2 cannot reach the state q_3 before $20s$, i.e., it cannot send m_1 before $20s$. As a consequence, the service Q_1 cannot receive it, i.e., Q_1 and Q_2 are *incompatible*.

So, using constraints inference, we can deduce the two external constraints:

(i) service Q_1: from the receipt of the message m_0 until receiving m_1, there is at most $10s$,

(ii) service Q_2: between the sending of m_0 and the sending m_1, there is at least $20s$.

Formally, the two external constraints are:

$(?m_0, ?m_1, [0, 10])$

$(!m_0, !m_1, [20, 20 + t))$ where t is the time related to the run-time process.

Once we get the two external constraints, we check if their periods are consistent. In the example $[20, 20 + t') \cap [0, 10] = \emptyset$, the two transitions are *absolutely inconsistent*, hence the two services are *incompatible*.
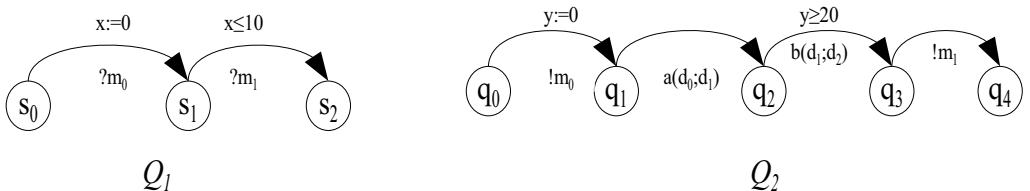

Fig. 3. Absolutely incompatible services.

Two inferred constraints (external constraints) are absolutely consistent if the

period to send the message is included in the period to receive it. However, when the sending period is not included in the receiving period but the two periods have some common values, we say that constraints are *likely consistent*. When the two periods are disjoint, we say that the two constraints are *absolutely inconsistent*.

Definition 6.2 Let p_1, p_2 two pre-conditions such that $(p_1 = p_2 = true)$, $(p_1 = ?m', p_2 =!m')$ or $(p_1 =!m', p_2 =?m')$. An output external constraint $e_1 = (p_1, !m, d_1)$ is *absolutely consistent* with an input external constraint $e_2 = (p_2, ?m, d_2)$, denoted $e_1 \subseteq e_2$ if $d_1 \subseteq d_2$. e_1 is *likely consistent* with e_2, denoted $e_1 \subsetneq e_2$ if $e_1 \nsubseteq e_2$ and $d_1 \cap d_2 \neq \emptyset$. e_1 is *absolutely inconsistent* with e_2 if $d_1 \cap d_2 = \emptyset$

Inference of external constraints can be done via: (i) synchronization over messages or (ii) reference to a common clock. In the following, we explain each of them.

6.2.1   Synchronization over messages

The inference based on the synchronization over a message can be explained using the example of the two Web services depicted in Figure 3. In the service Q_1, the clock x is reset when the message m_0 is received, i.e., in the transition $(s_0, ?m_0, x, s_1)$. On the other hand, the clock y of service Q_2 is reset when the message m_0 is sent, i.e., in the transition $(q_0, !m_0, y, q_1)$. Such transitions are called *rendez-vous synchronizing transitions*, in which we can say that $x \equiv y$. Having a rendez-vous synchronizing transition, we could detect that, once the message m_0 is sent, the service Q_2 can reach the state q_3 at least $20s$, i.e., it can send the message m_1 after $20s$. Since the service Q_1 can receive the message m_1 at most in $10s$, thus the two services are *not compatible* because their conversation will fail.

6.2.2   Global duration

When there is no rendez-vous synchronizing transition, let us assume that both services are started simultaneously. In this particular case, the idea is to infer the required period to send and receive a message. As illustrated in Fig. 4, once the service Q_1 receives the message m_0, it must receive the message m_1 within $20s$. The service Q_2 has no constraint on the transition $(q_2, !m_1, q_3)$ that enables to send the message m_1. Once the service Q_2 performs the operation $a(d_0; d_1)$, it must perform the operation $c(d_2; d_3)$ within the next $10s$. The message m_1 is sent before performing the operation $c(d_1; d_3)$, hence we can infer that the message is sent before $10s$. Then, the external constraints we can infer respectively for the service Q_1 and the service Q_2 are $(true, ?m_1, [0, 20s])$ and $(true, !m_1, [0, 10s])$

As it can be seen, $[0, 10s] \subseteq [0, 20s]$, hence we can deduce that the two services are *compatible*.

6.3   Web services compatibility classes

As seen in Fig. 5, the compatibility of services can be checked using three steps. The local consistency allows to detect incompatibilities of Web services regarding their constraints over data (step 1). Since the local consistency checking is insufficient
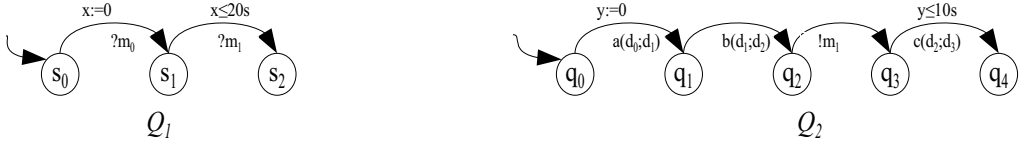
Fig. 4. An absolute compatibility.

to detect temporal incompatibilities, the external constraints must be inferred from internal constraints (step 2). By considering all the inferred constraints, we check if services are compatible regarding their temporal constraints (step 3).
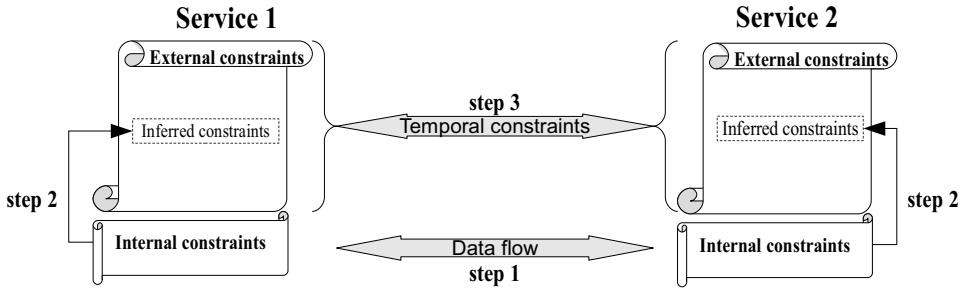

Fig. 5. Compatibility checking process.

According to our notions of consistency for external constraints and transitions, we consider three classes of compatibility for Web services: (i) *absolute compatibility*, (ii) *likely compatibility*, and (iii) *absolute incompatibility*.

Definition 6.3 Let Q_1 and Q_2 be two web services having T_1 and T_2 as respective sets of input/output transitions, and let E_1 and E_2 be their respective sets of external constraints.

Q_1 and Q_2 are *absolutely compatible* if the following holds (for $i, j \in \{1, 2\}, i \neq j$):

- $\forall t_i \in T_i, \exists t_j \in T_j$ such that $t_i \subseteq t_j$,
- $\forall e_i \in E_i, \exists e_j \in E_j$ such that $e_i \subseteq e_j$,

Q_1 and Q_2 are *likely compatible* if Q_1 and Q_2 are not absolutely compatible and the following holds (for $i, j \in \{1, 2\}, i \neq j$):

- $\forall t_i \in T_i, \exists t_j \in T_j$ such that $t_i \subseteq t_j$ or $t_i \subsetneq t_j$,
- $\forall e_i \in E_i, \exists e_j \in E_j$ such that $e_i \subseteq e_j$ or $e_i \subsetneq e_j$,

Q_1 and Q_2 are *absolutely incompatible* if one of the following holds (for $i, j \in \{1, 2\}, i \neq j$):

- $\exists t_i \in T_i, \nexists t_j \in T_j$ such that $t_i \subseteq t_j$ or $t_i \subsetneq t_j$,
- $\exists e_i \in E_i, \nexists e_j \in E_j$ such that $e_i \subseteq e_j$ or $e_i \subsetneq e_j$,

One can notice that the different classes of compatibility are disjoint and cover all the possible cases.

Our approach consists in analyzing the internal behavior of web services to infer external constraints used for checking the compatibility of services. We are studying

methods for the inference of external constraints, and this will allow us to implement the compatibility checking presented in this section.

In the future, we want to apply the compatibility analysis framework for the composition problem. A compatibility checking algorithm will allow us to synthesize a composition by verifying the compatibility of services according to their constraints.

7   Related work

The research field on the compatibility analysis for interoperability applications is very active. A lot of works have been published on automatic service mechanisms, using automata as a formal presentation [8,4,3,18,15,16]. The fundamental issue addressed by all these works is the same: given two services, are they able to interact?

In [8], the authors consider the sequence of messages that can be exchanged between two Web services. According to our approach, we consider also the data flow and constraints over these data. Furthermore, we consider temporal constraints.

Similarly to [8], the approach presented in [4] also considers only sequences of messages. The work presented in [4] has been extended with temporal constraints [3,18].

In [3], the temporal constraints enable to trigger transitions after an amount of time. The lack of the model presented in [3] is that the activation constraints can only be expressed over one transition. To go beyond such requirements, in this paper we proposed to consider the standard clocks used in timed automata [2], to express activation constraints that cannot be expressed in [3], such that having an activation constraint over a sequence of transitions. Contrary to our work, there is no data flow considerations, external constraints and constraints over counters. In the same area in [18] several important aspects such that data flow and temporal requirements as the external constraints, constraints over the global clock and counters are not considered.

In [12], the authors deal with analyzing and verifying properties of composite Web services specified as multiple BPEL processes. The properties are expressed via the temporal logic LTL [17]. The services are specified using an automaton-based formalism, where services are specified by the messages they can exchange asynchronously and the data flow. So, [12] investigates an approach that analyzes some given properties in a given composition, whilst we are interested in the compatibility analysis needed to build a composition.

Another work presented in [15] deals with modeling temporal requirements in a given composition. This work allows to express the same kind of constraints as in [18]. However, this model does not consider data flow, external constraints and constraints over counters and more generally constraints over data. Moreover, this work does not consider the compatibility analysis.

In the same vein, the authors of [11] propose to use timed automata to check the timed properties of a given composition. Thus they translate the descriptions

written in WSCI-WSCDL [14] into timed automata. In this paper, we are interested in checking the compatibility of services, which is a key feature to synthesize a composition, whilst [11] deals with the problem of verifying a given composition.

8 Conclusion and Perspectives

In this paper, we have presented an approach to deal with the automatic compatibility checking of Web services by considering their operations, messages, data associated to messages, together with conditions on these data and temporal constraints. We have defined two forms of constraints: (i) *internal constraints* used to model the service and (ii) *external constraints* inferred from internal ones. The inference of *external constraints* allows to detect some implicit constraints that can be used to show the *incompatibility of services*. The internal constraints can be local or global. Our notion of local clock is identical to the one used in timed automata [2]. We use a global clock to specify constraints that relies on absolute dates. As the global clock relies on the absolute time, thus it is never reset. The internal constraints can express *activation* and *dependency* conditions. Moreover, we consider data capabilities that allow us to specify guards. Then, in order to analyze the compatibility of services by considering these properties, we have proposed to extend the notion of WSTTS [15].

Our future work consists in studying how to infer automatically external constraints, and then to integrate compatibility checking mechanisms into a framework for the composition of Web services modeled as conversational automata [13].

References

[1] G. Alonso and F. Casati. Web services and service-oriented architectures. In *Proceedings of the 21st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 2005.

[2] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[3] B. Benatallah, F. Casati, J. Ponge, and F. Toumani. On temporal abstractions of web service protocols. In *The 17th Conference on Advanced Information Systems Engineering (CAiSE '05). Short Paper Proceedings*, 2005.

[4] B. Benatallah, F. Casati, and F. Toumani. Analysis and management of web service protocols. In *Conceptual Modeling - ER 2004, 23rd International Conference on Conceptual Modeling*, volume 3288 of *LNCS*, pages 524–541. Springer, 2004.

[5] D. Berardi. *Automatic Service Composition. Models, techniques and tools*. PhD thesis, La Sapienza University, Roma, 2005.

[6] D. Berardi, D. Calvanese, G. D. Giacomo, R. Hull, and M. Mecella. Automatic composition of transition-based semantic web services with messaging. In *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, pages 613–624. ACM, 2005.

[7] D. Berardi, D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of e-services that export their behavior. In *Service-Oriented Computing - ICSOC 2003, First International Conference, Trento, Italy, December 15-18, 2003, Proceedings*, volume 2910 of *LNCS*, pages 43–58. Springer, 2003.

[8] L. Bordeaux, G. Salaün, D. Berardi, and M. Mecella. When are two web services compatible? In *Technologies for E-Services, 5th International Workshop, TES 2004, Revised Selected Papers*, volume 3324 of *LNCS*, pages 15–28. Springer, 2005.

[9] T. Bultan, X. Fu, R. Hull, and J. Su. 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*, pages 403–410, 2003.

[10] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (wsdl) 1.1. *http://www.w3.org/TR/2001/NOTE-wsdl-20010315*, 2001.

[11] G. Diaz, J.-J. Pardo, M.-E. Cambronero, V. Valero, and F. Cuartero. Verification of web services with timed automata. In *Proceedings of the International Workshop on Automated Specification and Verification of Web Sites (WWV 2005)*, volume 157 of *ENTCS*, pages 19–34, 2005.

[12] X. Fu, T. Bultan, and J. Su. Analysis of interacting bpel web services. In *Proceedings of the 13th international conference on World Wide Web, WWW 2004, New York, NY, USA, May 17-20, 2004*, pages 621–630, 2004.

[13] N. Guermouche, O. Perrin, and C. Ringeissen. A mediator based approach for services composition. *INRIA-LORIA Research Report*, 2007.

[14] N. Kavanzas and al. Web service choreography description language (wscdl) 1.0. *http://www.w3.org/TR/ws-cdl-10/*.

[15] R. Kazhamiakin, P. K. Pandya, and M. Pistore. Timed modelling and analysis in web service compositions. In *Proceedings of the The First International Conference on Availability, Reliability and Security, ARES*, pages 840–846. IEEE Computer Society, 2006.

[16] A. Muscholl and I. Walukiewicz. A lower bound on web services composition. In *Proceedings of Foundations of Software Science and Computation Structures (FOSSACS)*, volume 4423 of *LNCS*, pages 274–287. Springer, 2007.

[17] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE, 1977.

[18] J. Ponge. A new model for web services timed business protocols. In *Atelier (Conception des systèmes d'information et services Web), SIWS-Inforsid*, 2006.