A TVPCCS Model for Testing Web Service Choreography and Generating BPEL Code

Mohammed Lamine Beggar and Lejian Liao

Abstract—**Web services are software applications which can be used through a network (intranet or Internet) via the exchange of messages based on XML standards. They are aimed at wild reusability and are typically designed to interact with other in order to build larger applications. Web service sometimes requires combining more than one to meet our requirements. Service composition is the process of creating new services from a set of services. In this context, one of the important investigations is the choreography compatibility analysis. We mean by the choreography compatibility the capability of a set of Web services of actually interacting by exchanging messages in a proper manner. Whether a set of services are compatible depends not only on their sequences of messages but also on quantitative properties such as timed properties. In this paper, we present a framework for model checking web service choreography based on checking web service (CWB) in which the web services support synchronous or asynchronous timed communications. Particularly, in this paper we present a new modeling of Web services using temporal value passing calculus of communicating systems (TVPCCS).**

Index Terms—**Model checking, web service composition, temporal value passing calculus of communicating systems, the edinburgh concurrency workbench.**

I. Introduction

Web services are increasingly gaining acceptance as a framework for facilitating application-to-application interactions within and across enterprises, they are typically designed to interact with other in order to build larger applications. Service composition is one of the most promising ideas underlying Web services [1]: new functionalities can be defined and implemented by combining and interacting with pre-existing services.

Since the complexity of composition increases, verification and validation of the composite Web service become a sophisticated task that deserves and has received many studies [2]. In this context, one of the important elements is the compatibility analysis. Two web services are compatible depends not only on static properties like the correct typing of their message parameters, but also on their dynamic behavior [3].

A lot of works on the compatibility problem of Web services deal with services that support synchronous communication and consider only the sequences of messages, that is, two Web services are compatible if each input (resp. output) message of a service corresponds to an output (resp. input) message of the other service in the same order. Nevertheless, the nature of Web services could be asynchronous, and then in this paper we tackle the problem of analyzing the compatibility of a choreography in which the Web services support synchronous or asynchronous communications. It is very important to define the nature of supported communications considered in a compatibility framework because two Web services which are incompatible using synchronous communication can be compatible using the asynchronous one. In an asynchronous communication, when a message is sent, it is inserted into a bounded message queue or stored into a buffer, and the receiver consumes (i.e. receives) the message while it is available in the queue or buffer.

The compatibility of Web services depends not only on the supported sequences of messages but there are other crucial quantitative properties such as timed properties [1], [4], [5], [6]. We define the timed properties to be the delays needed to be able to exchange messages (e.g., in an application, a Insurance Company can send its final decision to policyholder after 3 days and within 7 days). Some works using synchronous communication cannot discover or cannot discover all the eventual timed conflicts [3], [4], [6], [7].

In this paper, we first propose a framework for analyzing the choreography compatibility supporting synchronous or asynchronous communicating services and then present a mapping between TVPCCS and BPEL in order to allow an automatic translation between the two languages that permit designing and verifying in TVPCCS and then translating the specification onto BPEL. In our framework we take into account data flow that can be involved when exchanging messages. Furthermore, we consider timed properties that specify the delays to be able to exchange messages. Implicit timed conflicts could be rose during the interaction and the exchange of messages between Web services because implicit timed dependencies could be built between their different timed properties. We tried to use some existing work to discover deadlocks due to implicit timed conflicts but we notice that it is not possible since they do not consider timed properties or they use synchronous communication mode and then can't discover all deadlocks. In order to catch all the possible timed deadlocks, we propose to use the model checker the CWB (the edinburgh concurrency workbench).

To analyze the choreography compatibility we need the Web services description behavior which is the sequences of messages the service supports and the associated timed requirements. We use Temporal Value Passing Calculus of

Manuscript received October 24, 2011; revised December 28, 2011.

This work is funded by the Natural Science Foundation of China (NSFC, Grant No.60873237), Natural Science Foundation of Beijing (Grant No.4092037), and partially supported by Beijing Key Discipline Program.

M. La. Beggar is with School of Computer Science, Beijing Institute of Technology, Beijing, China (e-mail amine@bit.edu.cn).

L. Liao is with School of Computer Science and Technology, Beijing Institute of Technology, China (e-mail liaolj@bit.edu.cn).

Communicating Systems (TVPCCS) that is a timed extension of Calculus of Communicating Systems (CCS) to model the timed behavior, process algebra like Communicating Sequential Processes (CSP) and CCS have been already used in series of papers [8]-[11]. TVPCCS model is suitable to describe Web services timed behaviors. It is simple, fairly easy to understand and at the same time it is expressive enough to model the properties we consider.

In this paper we make the following contributions:

(1) We propose an asynchronous/synchronous model of Web services that consider sequence of messages, data and timed requirements. (2) We propose an abstraction process that aims to handle Web services using CWB. (3) We propose a mapping between TVPCCS and BPEL.

The reminder of the paper is organized as follows.Section II presents the case study that we use to show the related issues of the proposed approach. In Section III we introduce TVPCCS. In section IV, we discuss informally and intuitively the timed compatibility problem of a choreography using the TVPCCS to model web services and their timed behaviors. Section V presents a mapping from TVPCCS to TCCS. In order to be able to handle asynchronous services by CWB, in section VI, we present CWB objects and model the study case using it. Section VII presents the proposed formal choreography compatibility investigations. In Section VIII, we first explain the principle used to store information in TVPCCS action's name, and then present the mapping from TVPCCS to BPEL. In Section IX, we discuss related work. Finally, in section X we conclude.

II. Case Study

The goal of the application we consider is to manage policyholder request. Such a request involves three Web services: (1) requester service (RS), (2) Auto Repair Garage (ARG), and (3) Administration (Ad). The high level choreography model of the process is depicted on Fig. 1. To benefit from insurance, the process can be summarized as follows. (1) Via a requester service, a policyholder deposits a file in the administration, (2) the policyholder asks an estimated cost from the Auto Repair Garage, (3) the Auto Repair Garage negotiates a date of an appointment to examine the car. (4) After the examination, the Auto Repair Garage sends a report (estimated cost) to the administration. (5) After studying the insurance policy of the policyholder and the report of the Auto Repair Garage the administration sends a notification of the final decision to the policyholder.

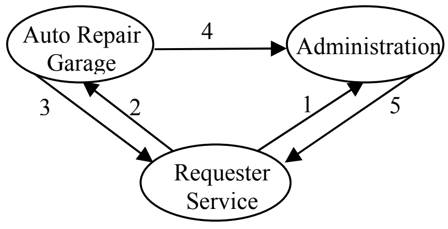


Fig. 1. Global view of the application

The timed requirements are:

1) Once the Auto Repair Garage proposes meeting dates to the policyholder, this latter must send the filled form within 24 hours.
2) The administration requires at least 3 days and at most 7 days from receiving the file from the policyholder to notify him by the final decision.
3) Via the requester service, once the policyholder obtains the car's information form, he must send the filled form within 36 hours.

The Web services we consider could support asynchronous communications. The first issue we deal with is how to model asynchronous communications using TCCS? When the services are interacting together, timed deadlocks could arise because the timed properties of the several services are local and are mutually independent, we must check that the choreography is deadlock free, the second issue is how to consider timed properties when analyzing the compatibility of a choreography?

III. TVPCCS

A. Basic Syntax of TVPCCS

The temporal value passing CCS is a kind of computation model to describe and analyze concurrent systems. In the temporal value passing CCS, every expression denotes an agent, and expresses a concurrent entity which can run freely. The communication between agents is implemented by exchanging information in named channels.

The collection of TVPCCS expressions, ranged over by P, is defined by the following expression where we take $a \in$ Act (set of actions), $X \in$ Var, $t \in$ T (T = {1, 2, 3...} represent divisions in time).

$$P ::= 0 \mid \underline{0} \mid X \mid a.P \mid \underline{a}.P \mid (t).P \mid P+P \mid P{+}{+}P \mid P|P \mid P\backslash a \mid \text{if b then } P \mid A(x_1, \ldots, x_n)$$

0 and $\underline{0}$ represents that the process is inactive and it does not perform any actions (0 represents the completely dead process. It can neither perform any computation, nor can it witness the passage of any time. $\underline{0}$ represents the process which can never perform any actions, but rather idles forever).

a.P represents the process which can perform the action a and evolve into the process P by so doing. The process cannot progress through time before performing the action a, and the action is assumed to occur instantaneously in time. $\underline{a}$.P represents the process which can perform the action a, but will allow any amount of time to pass before doing the action. Prefix action has three kinds of form:

1) τ stands for an internal and unobservable action τ.
2) a(x) represents that variable receives values from input channel a.
3) ā(e) denotes that the value of variable send out along output channel a .

(t).P represents the process which will do no computation for an amount of time t, but at that point in time will commence behaving as the process P.

P+Q represents a choice between the two processes P and Q. The process behaves as the process P or the process Q, with the choice being made at the time of the first action, or else at the occurrence of a passage of time when only one of the operands may allow the time passage to occur. In this latter case, the second temporally deadlocked process is dropped from the computation. This operator is referred to as weak choice.

P++Q represents a stronger notion of choice between the two processes P and Q. The process behaves as the process P or the process Q, with the choice being made only at the time of the first action. Thus, for instance, any initial passage of time must be allowed by both P and Q. This operator is referred to as strong choice.

A process can be a parallel composition of sub-processes: P|Q, Each of the processes may do any actions independently, or they may synchronize on complementary actions, resulting in *a τ* action. Any passage of time must be allowed and recorded by each of P and Q.

A process may have a restriction, which is denoted by \S. P is a process and S is a set of named channels, imposing that an emission on m $\in$ S by one sub-process of P can occur only if another sub-process does a reception on the same channel.

if *b* then P: represents that P will execute if the Boolean expressions b is 'true'.

A($x_1, \dots, x_n$) represents constant A with arity n. There is a defining equation: A($x_1, \dots, x_n$) = P where the right-hand side P may contain no agent variables, and no free value variable except $x_1, \dots, x_n$.

B. Operational Semantics of TVPCCS

$$\frac{}{a.P \xrightarrow{a} P} \qquad \frac{}{\underline{a}.P \xrightarrow{a} P}$$

$$\frac{P \xrightarrow{a} P'}{P+Q \xrightarrow{a} P'} \qquad \frac{Q \xrightarrow{a} Q'}{P+Q \xrightarrow{a} Q'}$$

$$\frac{P \xrightarrow{a} P'}{P{+}{+}Q \xrightarrow{a} P'} \qquad \frac{Q \xrightarrow{a} Q'}{P{+}{+}Q \xrightarrow{a} Q'}$$

$$\frac{P \xrightarrow{a} P'}{P|Q \xrightarrow{a} P'|Q} \qquad \frac{Q \xrightarrow{a} Q'}{P|Q \xrightarrow{a} P|Q'} \qquad \frac{P \xrightarrow{a(x)} P',\ Q \xrightarrow{\overline{a(e)}} Q'}{P|Q \xrightarrow{\tau} P|Q'\ \{e/x\}}$$

$$\frac{P \xrightarrow{a} P'}{P\backslash b \xrightarrow{a} P'\backslash b}\ (a \neq b)$$

Fig. 2. Action Derivation Rules

$$\frac{}{\underline{0} \overset{t}{\rightsquigarrow} \underline{0}} \qquad \frac{}{\underline{a}.P \overset{t}{\rightsquigarrow} \underline{a}.P}$$

$$\frac{}{(t).P \overset{t}{\rightsquigarrow} P} \qquad \frac{}{(s+t).P \overset{s}{\rightsquigarrow} (t).P} \qquad \frac{P \overset{s}{\rightsquigarrow} P'}{(t).P \overset{s+t}{\rightsquigarrow} P'}$$

$$\frac{P \overset{t}{\rightsquigarrow} P',\ Q \overset{t}{\rightsquigarrow} Q'}{P+Q \overset{t}{\rightsquigarrow} P'+Q'} \qquad \frac{P \overset{t}{\rightsquigarrow} P'}{P+Q \overset{t}{\rightsquigarrow} P'}\ (|Q|_T<t) \qquad \frac{Q \overset{t}{\rightsquigarrow} Q'}{P+Q \overset{t}{\rightsquigarrow} Q'}\ (|P|_T<t)$$

$$\frac{P \overset{t}{\rightsquigarrow} P',\ Q \overset{t}{\rightsquigarrow} Q'}{P{+}{+}Q \overset{t}{\rightsquigarrow} P'{+}{+}Q'} \qquad \frac{P \overset{t}{\rightsquigarrow} P',\ Q \overset{t}{\rightsquigarrow} Q'}{P|Q \overset{t}{\rightsquigarrow} P'|Q'} \qquad \frac{P \overset{t}{\rightsquigarrow} P'}{P\backslash a \overset{t}{\rightsquigarrow} P'\backslash a}$$

Fig. 3. Temporal Derivation Rules

In this section, we define the operational semantics of our language. The semantics is transition based, outlining what actions and time delays a process can witness. In order to define our semantic we must first define a syntactic predicate which will allow us to describe when a process must stop delaying its computation within a particular amount of time. This is done using the following function on TVPCCS terms.

Definition [12] The function $|\,.\,|_T$: TCCS $\rightarrow$ {0, 1 , 2 ,… ,γ} defines the maximum delay which a process may allow before forcing a computation (or deadlock) to occur. Formally, this function is defined as follows:

$$\begin{array}{ll}
|\,0\,|_T = 0 & |\,P{+}{+}Q\,|_T = \max(|\,P\,|_T, |\,Q\,|_T) \\
|\,X\,|_T = 0 & |\,P+Q\,|_T = \min(|\,P\,|_T, |\,Q\,|_T) \\
|\,P\backslash a\,|_T = |\,P\,|_T & |\,P\,|\,Q\,|_T = \min(|\,P\,|_T, |\,Q\,|_T) \\
|\,a.P\,|_T = 0 & |\,(s).P\,|_T = s + |\,P\,|_T
\end{array}$$

This definition is well defined as long as all recursive variables are guarded by action or time prefixes.

In Fig. 1 and Fig. 2, we present the operational rules for our language. They are presented in a natural deduction style, and are to be read as follows: if the transition(s) above the inference line can be inferred, then we can infer the transition below the line. Our transitional semantics over TVPCCS then is given by the least relations

$\rightarrow\ \subseteq$ TVPCCS × Act × TVPCCS and $\rightsquigarrow\ \subseteq$ TVPCCS × T × TVPCCS (written $P \xrightarrow{a} Q$ and $P \overset{t}{\rightsquigarrow} Q$ respectively) satisfying the rules laid out in Fig. 1 and Fig. 2. Notice that these rules respect the informal description of the constructs given in III.A

IV. Synchronous/Asynchronous Communication and Compatibility Problems

In this section, by using examples, we present how to model composition of synchronous or asynchronous Web services using TVPCCS and we discuss informally and intuitively the timed choreography compatibility problem and the related issues

Example 1: Let us first consider the two untimed services Q and Q' depicted on Fig. 4. Using synchronous communication and services with no times property, we are using the model proposed by [9], these two services are incompatible because they don't produce and consume their messages in the same order. The first action of the service Q is to send the message m_0 (d_0, d_1), as services are synchronized over messages, it is blocked because the service Q' is not waiting for the message m_0 (d_0, d_1). The same problem arises for the service Q' whose first action is to send the message m_2(d_3).

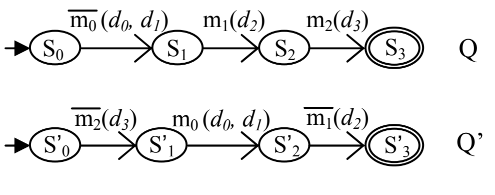


Fig. 4. Untimed Incompatible Synchronous Web services

The representation of Q and Q' is $Q = \overline{m_0}(d_0,d_1).m_1(d_2).m_2(d_3).0$

$Q' = \overline{m_2}(d_3).m_0(d_0,d_1).\overline{m_1}(d_2).0$ and their composition using synchronous communication is

$\text{Composition} = (Q \mid Q') \setminus S$ where S = { $m_0(d_0,d_1)$, $m_1(d_2)$, $m_2(d_3)$ }.

Example 2: we use the same services Q and Q' depicted on Fig. 4. but this time we consider the case of asynchronous communication. The two services become compatible, the service Q sends the message $m_0(d_0, d_1)$ that is stored in the buffer M0 and start waiting for $m_1(d_2)$, on the other side Q' sends $m_2(d_3)$ that is stored in the associated buffer M2 and then consumes $m_0(d_0, d_1)$, finally sends the message $m_1(d_2)$ that also is stored in a buffer M1 and terminate. At the moment $m_1(d_2)$ becomes available the service Q consumes it and then consumes $m_2(d_3)$ and terminate. By using the existing work, these two services are considered as incompatible although they can succeed an execution. In fact, the proposed frameworks (e.g.,see [3], [4], [6], [7], [13]) deal only with synchronous communicating services.

$$Q = \overline{m_0}(d_0,d_1).m_1'(d_2).m_2'(d_3).0 \qquad M1 = m_1(d_2).\overline{m_1}'(d_2).0$$

$$Q' = \overline{m_2}(d_3).m_0'(d_0,d_1).\overline{m_1}(d_2).0 \qquad M2 = m_2(d_3).\overline{m_2}'(d_3).0$$

$$M0 = m_0(d_0,d_1).\overline{m_0}'(d_d,d_1).0$$

$\text{Composition} = (Q \mid Q' \mid M0 \mid M1 \mid M2) \setminus S$ where S= { $m_0(d_0,d_1)$, $m_1(d_2)$, $m_2(d_3)$, $m_0'(d_0,d_1)$, $m_1'(d_2)$, $m_2'(d_3)$ }.

For each message we add a process acting like a buffer that receives this message, sends it and finally terminate. By adding buffers we transform the synchronous communication to asynchronous one because services Q and Q'are not still synchronized over messages.

When the services interact together, implicit timed conflicts could arise. To illustrate this issue, in the following we present an illustrative example.

Example 3: Let us add timed properties to services Q and Q' depicted on Fig. 5. that use asynchronous communication. Now, the service Q sends the message $m_0(d_0, d_1)$ that is stored in the buffer M0 and start counting time, it must receive the message $m_2(d_3)$ within 10 units of time. On the other hand, Q' can send the message $m_2(d_3)$ that can be stored in the buffer M2. The service Q remains blocked until the message $m_1(d_2)$ will be available but Q' can consume the message $m_0(d_0, d_1)$ which has been already sent by Q. Once consumed, Q' sends the message $m_1(d_2)$ after 20 and within 40 units of time from consuming the message $m_0(d_0, d_1)$ (i.e., $20 \leq x \leq 40$). Consequently, the message $m_1(d_2)$ becomes available in the buffer M1 after 20 units of time from consuming the message $m_0(d_0, d_1)$. In that case, Q will be able to consume the message $m_1(d_2)$ after 20 units of time which means that the message $m_2(d_3)$ can't be consumed within 10 units of time, it is a timed conflict.

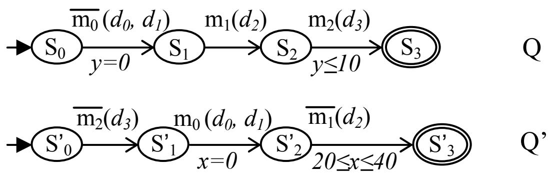


Fig. 5. Incompatible Timed Asynchronous Web services

$$Q = \overline{m_0}(d_0,d_1).\overline{\text{startT1}}.\overline{\text{wfm}_1'}.(\underline{\text{stopT1}}.\underline{0} + +\underline{m_1}'(d_2).\overline{\text{wfm}_2'}.$$
$$(\underline{\text{stopT1}}.\underline{0} + +\underline{m_2}'(d_3).(\underline{0} \mid \underline{\text{stopT1}}.\underline{0})))$$

$$Q' = \overline{m_2}(d_3).\overline{\text{wfm}_0'}.\underline{m_0}'(d_0,d_1).20.\overline{\text{startT2}}.(\underline{\text{stopT2}}.\overline{m_1}(d_2).$$
$$\underline{0} + +\underline{\overline{m_1}}(d_2).(\underline{0} \mid \underline{\text{stopT}}.\underline{0}))$$

$$M0 = \underline{\text{wfm}_0'}.\underline{m_0}(d_0,d_1).\overline{m_0}'(d_0,d_1).\underline{0} + +\underline{m_0}(d_0,d_1).\underline{\text{wfm}_0'}.$$
$$\overline{m_0}'(d_0,d_1).\underline{0}$$

$$M1 = \underline{\text{wfm}_1'}.\underline{m_1}(d_2).\overline{m_1}'(d_2).\underline{0} + +\underline{m_1}(d_2).\underline{\text{wfm}_1'}.\overline{m_1}'(d_2).\underline{0}$$

$$M2 = \underline{\text{wfm}_2'}.\underline{m_2}(d_3).\overline{m_2}'(d_3).\underline{0} + +\underline{m_2}(d_3).\underline{\text{wfm}_2'}.\overline{m_2}'(d_3).\underline{0}$$

$$T1 = \underline{\text{startT1}}.10.\overline{\text{stopT1}}.\underline{0} \qquad T2 = \underline{\text{startT2}}.20.\overline{\text{stopT2}}.\underline{0}$$

$$\text{Composition} = (Q \mid Q' \mid M0 \mid M1 \mid M2 \mid T1 \mid T2) \setminus S$$

S= { $m_0(d_0,d_1)$, $m_1(d_2)$, $m_2(d_3)$, $m_0'(d_0,d_1)$, $m_1'(d_2)$, $m_2'(d_3)$, startT1, stopT1, startT2, stopT2, wfm_0', wfm_1', wfm_2'}.

We add processes acting like Timer, for example we know that the process Q' after consuming the message $m_0(d_0, d_1)$ must send the message $m_1(d_2)$ after 20 units of time and within 40 units, so after consuming the message $m_0(d_0, d_1)$ we wait 20 units of time and then we start the timer T2 which count 20 units of time. The process Q' can send the message $m_1(d_2)$ or wait at most until the timer T2 completes the counting to send it.

When using TVPCCS, if the process $M0 = m_0(d_0,d_1).\overline{m_0}'(d_0,d_1).0$ can't receive the message m_0 at the present instant, then there is a deadlock, because M0 can't allow any passage of time, consequently the definition of buffers change. To guaranty that when a service is waiting for a message m_n', this latter will be consumed as soon as it is available (without any passage of time) new messages are used named wfm_n' (waiting for m_n'). the new definition of buffer is:

$$Mn = \underline{\text{wfm}_n'}.\underline{m_n}(d_x).\overline{m_n}'(d_x).\underline{0} + +\underline{m_n}(d_x).\underline{\text{wfm}_n'}.\overline{m_n}'(d_x).\underline{0}$$

In order to handle the eventual timed conflicts, we propose a Framework for Model Checking Web Service choreography based on CWB. To do so, we first present a mapping from TVPCCS to TCCS and then a mapping from TCCS to CWB.

V. From TVPCCS to TCCS

We assume that all values belong to some fixed value set V. we begin by showing how some basic examples can be reduced from TVPCCS to TCCS.

Let C be a buffer cell.

$$C = in(x).C'(x) \text{ and } C'(x) = \overline{out}(x).C$$

Consider first the parameterized constant C'. It will become a family of constants C'_v, one for each value ∈ V.

Similarly the parameterized negative prefix " $\overline{out}(x)$ " becomes a family of prefixed " $\overline{out_v}$ ", one for each value v. Thus the single defining equation for C' becomes a family of defining equations

$$C'_v = \overline{out_v}.C \quad (v \in V)$$

Now consider the prefix "(x)". To reflect the fact that it can accept any input value, because it binds the variable x, we translate it to "$\sum_{v \in V} in_v$"; in this way the use of a bound variable x is replaced by summation. Thus the defining equation for C becomes.

$$C = \sum_{v \in V} in_v.C'_v$$

Next let's consider this example:
P = in(x).Start(x)
Start(x) = if smallThanTen(x) then F(x)
else if smallThanHundred(x) then G(x)
else H(x)

Since Start takes a parameter, like C', we expect the second equation to be reduced to a family of equations, one for each x ∈ V. But in this case the form of the right-hand side of each equation depends up on the value of x, as determined by the predicates smallThanTen and smallThanHundred the translation is as follows:

$$P = \sum_{x \in V} in_x.Start_x$$

$$Start_x = \begin{cases} F_x & \text{(if smallThanTen(x))} \\ G_x & \text{(if } \neg\text{smallThanTen(x) and smallThanHundred(x))} \\ H_x & \text{(if } \neg\text{smallThanTen(x) and } \neg\text{smallThanHundred(x))} \end{cases}$$

More generally, we assume that P range over the collection of TVPCCS expressions, we also assume value expressions e and Boolean expressions b, built from value variable x,y,… together with value constants v and any operator symbol we wish. Let A be the set of actions names like a, b, in,... then we denote by Ā the set of co-names like $\bar{a}, \bar{b}, \overline{in}$,... Then we set L=A∪Ā; L is the set of labels and we shall use l, l' to range over L. Our translation of TVPCCS expressions into TCCS rests upon the idea that to each label in the TVPCCS corresponds a set {l_v: v∈V} of labels in the TCCS.

For each TVPCCS expression P without free value variables, its translated form $\hat{P}$ is given in Table I.

TABLE I: THE MAPPING FROM TVPCCS TO TCCS

P	$\hat{P}$
X	X
a(x).E	$\sum_{v \in V} a_v \hat{E}\{v/x\}$
$\underline{a}$(x).E	$\sum_{v \in V} \underline{a}_v \hat{E}\{v/x\}$
ā(e).E	$\bar{a}_e.\ \hat{E}$
$\underline{\bar{a}}$(e).E	$\underline{\bar{a}}_e.\ \hat{E}$
τ. E	τ. $\hat{E}$
(t). E	(t). $\hat{E}$
$\sum_{i \in I} E_i$	$\sum_{i \in I} \hat{E}_i$
$E_1 \vert E_2$	$\hat{E}_1 \vert \hat{E}_2$
E\L	$\hat{E} \setminus \{l_v : l \in L, v \in V\}$
if b then E	$\begin{cases} \hat{E} & \text{if } b = \text{true} \\ 0 & \text{otherwise} \end{cases}$
A(e_1, … , e_n)	$A_{e_1,\dots,e_n}$

Here is our modeling of the case study, the meanings of the messages are listed below (abbreviations are defined for each message at the end of each explanation).

FileDeposite: Via a requester service, a policyholder deposits a file in the administration (fd(d_0)).

FormClaim: Via a requester service, the policyholder asks auto repair garage for a form (fc(d_1)).

GettingForm: the policyholder receives the form from auto repair garage (gf(d_2)).

SendFilledForm: after filling the form, the policyholder sends it to auto repair garage (sff(d_3)).

MeetingProposedDates: auto repair garage proposes dates to examine the car (mpd(d_4, d_5, d_6)).

MeetingConfirmedDate: the policyholder chooses a date for meeting (mcd(d_7)).

CarExamination: auto repair garage examines the car (ce(d_8)).

EstimatedCost: the auto repair garage sends the reparation estimated cost to the administration (ec(d_9)).

FinalNotification: the administration sends the decision to the policyholder (fn(d_{10})).

The timed conversational protocols of the three services introduced in Section II are depicted on Fig. 6.

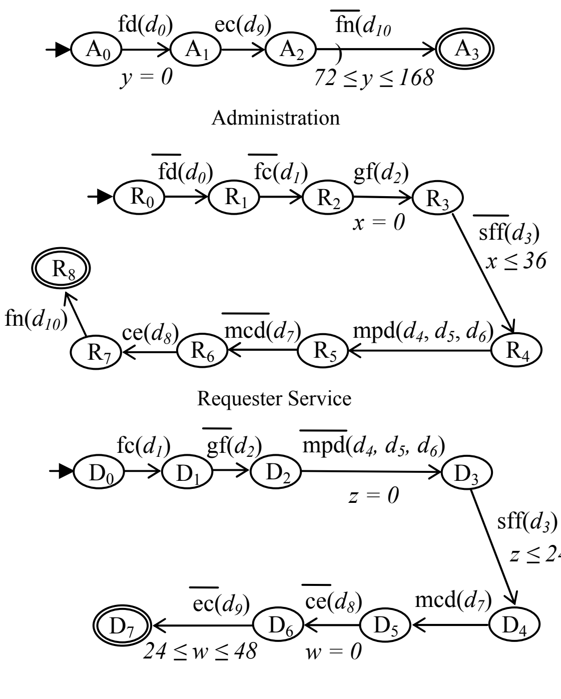


Fig. 6. Web services of the case study

$$AD = \overline{wffd'}.\underline{fd'}(d_0).\overline{startT0}.\overline{startT4}.\overline{wfec'}.(\underline{stopT0}.(\underline{stopT4}.\underline{0} + + \underline{ec'}(d_9).\overline{fn}(d_{10}).(\underline{0}\,|\,\underline{stopT4}.\underline{0})) + + \underline{ec'}(d_9).\underline{stopT0}.\overline{fn}(d_{10}).(\underline{0}\,|\,\underline{stopT4}.\underline{0}))$$

$$ARG = \overline{wffc'}.\underline{fc'}(d_1).\overline{gf}(d_2).\overline{mpd}(d_4,d_5,d_6).\overline{startT2}.\overline{wfsff'}.(\underline{stopT2}.\underline{0} + + \underline{sff'}(d_3).(\overline{wfmcd'}.\underline{mcd'}(d_7).\overline{ce}(d_8).24.\overline{startT3}.(\underline{stopT3}.\overline{ec}(d_9).\underline{0} + + \overline{ec}(d_9).(\underline{0}\,|\,\underline{stopT3}.\underline{0}))\,|\,\underline{stopT2}.\underline{0}))$$

$$\begin{aligned}\mathrm{RS} = {} & \overline{\mathrm{fd}}(d_0).\overline{\mathrm{fc}}(d_1).\overline{\mathrm{wfgf'}}.\underline{\mathrm{gf'}}(d_2).\overline{\mathrm{startT1}}.(\underline{\mathrm{stopT1}}.\overline{\mathrm{sff}}(d_3).\\ & \overline{\mathrm{wfmpd'}}.\underline{\mathrm{mpd'}}(d_4,d_5,d_6).\overline{\mathrm{mcd}}(d_7).\overline{\mathrm{wfce'}}.\underline{\mathrm{ce'}}(d_8).\overline{\mathrm{wffn'}}.\\ & \underline{\mathrm{fn'}}(d_{10}).\underline{0}++\overline{\underline{\mathrm{sff}}}(d_3).(\overline{\mathrm{wfmpd'}}.\underline{\mathrm{mpd'}}(d_4,d_5,d_6).\\ & \overline{\mathrm{mcd}}(d_7).\overline{\mathrm{wfce'}}.\underline{\mathrm{ce'}}(d_8).\overline{\mathrm{wffn'}}.\underline{\mathrm{fn'}}(d_{10}).\underline{0}\,|\,\underline{\mathrm{stopT1}}.\underline{0}))\end{aligned}$$

$$\mathrm{T0} = \underline{\mathrm{startT0}}.72.\overline{\mathrm{stopT0}}.\underline{0}$$

$$\mathrm{T1} = \underline{\mathrm{startT1}}.36.\overline{\mathrm{stopT1}}.\underline{0}$$

$$\mathrm{T2} = \underline{\mathrm{startT2}}.24.\overline{\mathrm{stopT2}}.\underline{0}$$

$$\mathrm{T3} = \underline{\mathrm{startT3}}.24.\overline{\mathrm{stopT3}}.\underline{0}$$

$$\mathrm{T4} = \underline{\mathrm{startT4}}.168.\overline{\mathrm{stopT4}}.\underline{0}$$

$$\mathrm{Bufce} = \underline{\mathrm{wfce'}}.\underline{\mathrm{ce}}(d_8).\overline{\mathrm{ce'}}(d_8).\underline{0}++\underline{\mathrm{ce}}(d_8).\underline{\mathrm{wfce'}}.\overline{\mathrm{ce'}}(d_8).\underline{0}$$

$$\mathrm{Bufec} = \underline{\mathrm{wfec'}}.\underline{\mathrm{ec}}(d_9).\overline{\mathrm{ec'}}(d_9).\underline{0}++\underline{\mathrm{ec}}(d_9).\underline{\mathrm{wfec'}}.\overline{\mathrm{ec'}}(d_9).\underline{0}$$

$$\mathrm{Buffc} = \underline{\mathrm{wffc'}}.\underline{\mathrm{fc}}(d_1).\overline{\mathrm{fc'}}(d_1).\underline{0}++\underline{\mathrm{fc}}(d_1).\underline{\mathrm{wffc'}}.\overline{\mathrm{fc'}}(d_1).\underline{0}$$

$$\mathrm{Buffd} = \underline{\mathrm{wffd'}}.\underline{\mathrm{fd}}(d_0).\overline{\mathrm{fd'}}(d_0).\underline{0}++\underline{\mathrm{fd}}(d_0).\underline{\mathrm{wffd'}}.\overline{\mathrm{fd'}}(d_0).\underline{0}$$

$$\mathrm{Buffn} = \underline{\mathrm{wffn'}}.\underline{\mathrm{fn}}(d_{10}).\overline{\mathrm{fn'}}(d_{10}).\underline{0}++\underline{\mathrm{fn}}(d_{10}).\underline{\mathrm{wffn'}}.\overline{\mathrm{fn'}}(d_{10}).\underline{0}$$

$$\mathrm{Bufgf} = \underline{\mathrm{wfgf'}}.\underline{\mathrm{gf}}(d_2).\overline{\mathrm{gf'}}(d_2).\underline{0}++\underline{\mathrm{gf}}(d_2).\underline{\mathrm{wfgf'}}.\overline{\mathrm{gf'}}(d_2).\underline{0}$$

$$\begin{aligned}\mathrm{Bufmcd} = {} & \underline{\mathrm{wfmcd'}}.\underline{\mathrm{mcd}}(d_7).\overline{\mathrm{mcd'}}(d_7).\underline{0}++\underline{\mathrm{mcd}}(d_7).\underline{\mathrm{wfmcd'}}.\\ & \overline{\mathrm{fn'}}(d_7).\underline{0}\end{aligned}$$

$$\begin{aligned}\mathrm{Bufmpd} = {} & \underline{\mathrm{wfmpd'}}.\underline{\mathrm{mpd}}(d_4,d_5,d_6).\overline{\mathrm{mpd'}}(d_4,d_5,d_6).\underline{0}++\\ & \underline{\mathrm{mpd}}(d_4,d_5,d_6).\underline{\mathrm{wfmpd'}}.\overline{\mathrm{mpd'}}(d_4,d_5,d_6).\underline{0}\end{aligned}$$

$$\mathrm{Bufsff} = \underline{\mathrm{wfsff'}}.\underline{\mathrm{sff}}(d_3).\overline{\mathrm{sff'}}(d_3).\underline{0}++\underline{\mathrm{sff}}(d_3).\underline{\mathrm{wfsff'}}.\overline{\mathrm{sff'}}(d_3).\underline{0}$$

$$\begin{aligned}\mathrm{Composition} = {} & (\mathrm{AD}\,|\,\mathrm{ARG}\,|\,\mathrm{RS}\,|\,\mathrm{Bufce}\,|\,\mathrm{Bufec}\,|\,\mathrm{Buffc}\,|\,\mathrm{Buffd}\,|\\ & \mathrm{Buffn}\,|\,\mathrm{Bufgf}\,|\,\mathrm{Bufmcd}\,|\,\mathrm{Bufmpd}\,|\,\mathrm{Bufsff}\,|\,\mathrm{T0}\,|\\ & \mathrm{T1}\,|\,\mathrm{T2}\,|\,\mathrm{T3}\,|\,\mathrm{T4})\setminus \mathrm{S}\end{aligned}$$

$$\begin{aligned}\mathrm{S} = \{ & \mathrm{ce}(d_8), \mathrm{ce'}(d_8), \mathrm{ec}(d_9), \mathrm{ec'}(d_9), \mathrm{fc}(d_1), \mathrm{fc'}(d_1), \mathrm{fd}(d_0), \mathrm{fd'}(d_0),\\ & \mathrm{fn}(d_{10}), \mathrm{fn'}(d_{10}), \mathrm{gf}(d_2), \mathrm{gf}(d_2)', \mathrm{mcd}(d_7), \mathrm{mcd'}(d_7),\\ & \mathrm{mpd}(d_4,d_5,d_6), \mathrm{mpd'}(d_4,d_5,d_6), \mathrm{sff}(d_3), \mathrm{sff'}(d_3), \mathrm{startT0},\\ & \mathrm{startT1}, \mathrm{startT2}, \mathrm{startT3}, \mathrm{startT4}, \mathrm{stopT0}, \mathrm{stopT1}, \mathrm{stopT2},\\ & \mathrm{stopT3}, \mathrm{stopT4}, \mathrm{wfce'}, \mathrm{wfec'}, \mathrm{wffc'}, \mathrm{wffd'}, \mathrm{wffn'}, \mathrm{wfgf'}, \mathrm{wfmcd'},\\ & \mathrm{wfmpd'}, \mathrm{wfsff'}\}\end{aligned}$$

VI. From TCCS to the CWB (the Edinburgh Concurrency Workbench)

The Edinburgh concurrency workbench (CWB) is an automated tool which caters for the manipulation and analysis of concurrent systems. In particular, the CWB allows for various equivalence, preorder and model checking using a variety of different process semantics.

There are different objects, namely agents, action sets and propositions, using CCS, (for synchronous/asynchronous untimed web services) processes are defined as agents where *m1* stands for receiving the message m_1 and 'm1 stands for sending the message m_1.Using TCCS, *m1.0* stands for $m_1.0$, *\$m1.0* for $\underline{m_1}.0$ and *2.m1.0* for $2.m_1.0$. In CWB the propositions are nexpressed in the propositional modal μ-calculus, an agent A (process A) satisfies <K>P if it has a K-derivative satisfying P, that is, if there is some a ∈ K such that $A \xrightarrow{a} A'$ with A' satisfying P. an agent A (process A) satisfies [K]P if every K-derivative of A satisfies P, that is, if A' satisfies P whenever there is an a ∈ K such that $A \xrightarrow{a} A'$. In natural models the computing tree logic (CTL) operator F can be characterized by a μTL formula: $M \models F^{+}\psi \text{ iff } M \models \mu q X(\psi \vee q)$. so in CWB we define a parameterized proposition EF(P)=min(Z.P|<->Z) and AF(P)=min(Z.P|([-]Z &<->T)). Where min(X.P) is the least fix point temporal formula.

The CWB objects resulting from the modeling are depicted on Fig. 7.

VII. Formal Asynchronous Compatibility Checking

In this section, we present the compatibility checking using our modeling and CWB. Like the existing works [4], [6], [7], [14], we distinguish three compatibility classes: (1) full compatibility, (2) partial compatibility, and (3) full incompatibility.

A. Full compatibility

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

Formally, a set of Web services interact without an eventual blocking is equivalent to check if the services reach their final states. At the same time when the services reach their final states, all the sent messages must be consumed is formally equivalent to check that when the services reach their final states, all buffers must be empty.

Each agent representing a web service or a message buffer send a message terminate *"tr"* just before terminate, and another agent V is defined receiving the messages terminate from all the web services and the message buffers and finally sending the message composable.

Let $P_1 \ldots P_n$ be n agents corresponding to n (asynchronous) services ($P_{1..n}$= …'tr.\$0), $B_1 \ldots B_m$ be m agents corresponding to m messages buffers ($B_{1..m}$= …'tr.\$0) and $T_1 \ldots$ T_k be k agents corresponding to k timers. Composition = (P_1|…| P_n| B_1|…|B_m| T_1|… |T_k| V)\S where S is the set of messages and $V = \underbrace{\$\mathrm{tr}\ldots\$\mathrm{tr}}_{n+m}.'\mathrm{composable}.0$

Web services are said to be fully compatible when for all paths the message composable is emitted. To check the full compatibility we use the command :

checkprop(Composition, AF(<'composable>T)).

B. Partial compatibility

When the set of n services $P_1 \ldots$ P_n do not constitute a full compatible choreography, we check if the services can fulfill at least one execution so that all the produced messages during this execution are consumed.

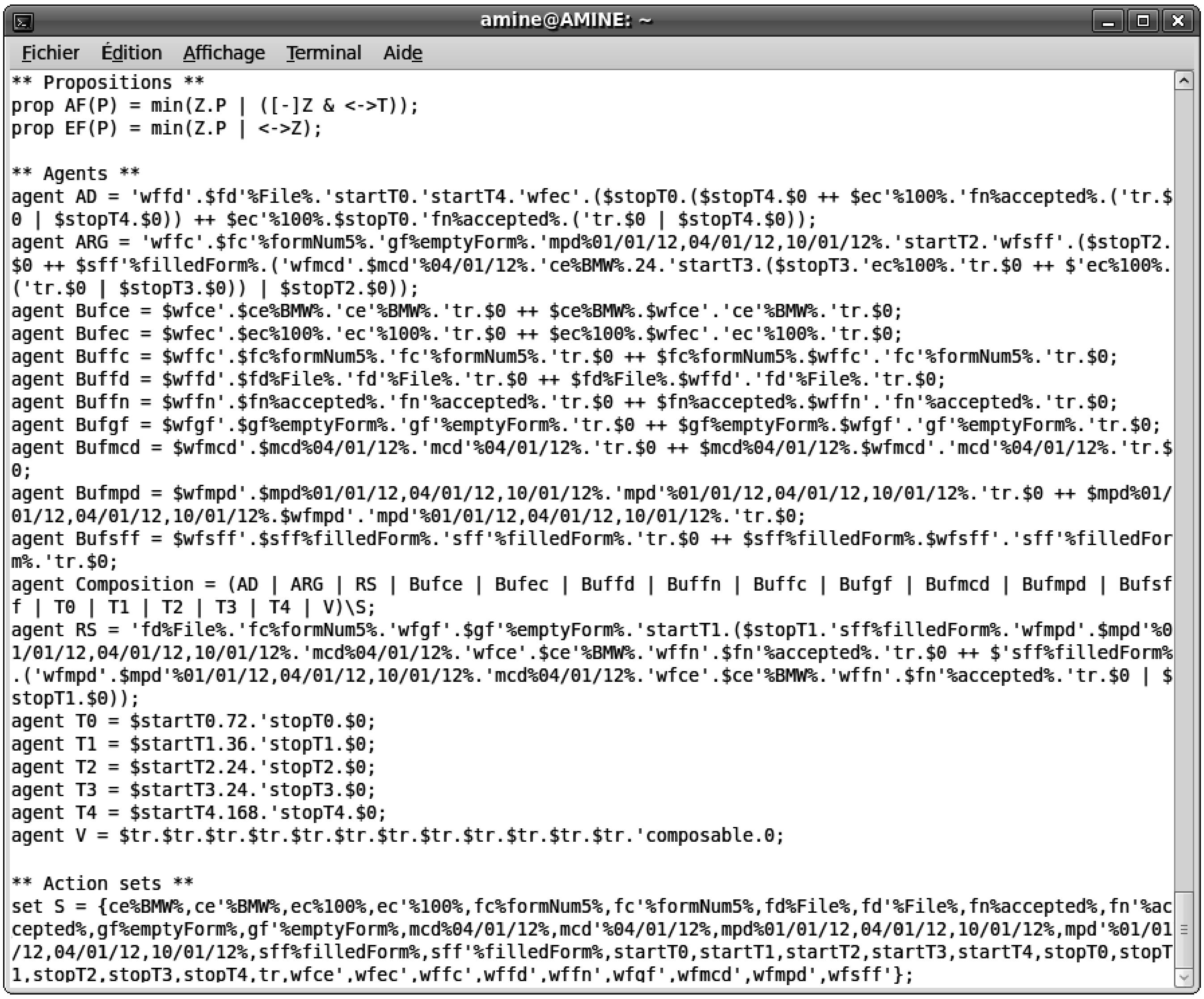
Formally, a set of services can terminate at least one execution is equivalent to check if the state *"final"* can be reached via at least one path and at the same time when the services reach their final states, all buffers must be empty.

Web services are said to be partially compatible when they are not fully compatible but for some paths the message composable is emitted. To check the partial compatibility we use the command :

checkprop(Composition, EF(<'composable>T)).

C. Full incompatibility

When a set of n Web services do not even constitute a partial compatible choreography, we say that the services build a full incompatible choreography. The 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"* or for all the paths, when the services reach their final states, buffers are not empty.



```
** Propositions **
prop AF(P) = min(Z.P | ([-]Z & <->T));
prop EF(P) = min(Z.P | <->Z);

** Agents **
agent AD = 'wffd'.$fd'%File%.'startT0.'startT4.'wfec'.($stopT0.($stopT4.$0 ++ $ec'%100%.'fn%accepted%.('tr.$
0 | $stopT4.$0)) ++ $ec'%100%.$stopT0.'fn%accepted%.('tr.$0 | $stopT4.$0));
agent ARG = 'wffc'.$fc'%formNum5%.'gf%emptyForm%.'mpd%01/01/12,04/01/12,10/01/12%.'startT2.'wfsff'.($stopT2.
$0 ++ $sff'%filledForm%.('wfmcd'.$mcd'%04/01/12%.'ce%BMW%.24.'startT3.($stopT3.'ec%100%.'tr.$0 ++ $'ec%100%.
('tr.$0 | $stopT3.$0)) | $stopT2.$0));
agent Bufce = $wfce'.$ce%BMW%.'ce'%BMW%.'tr.$0 ++ $ce%BMW%.$wfce'.'ce'%BMW%.'tr.$0;
agent Bufec = $wfec'.$ec%100%.'ec'%100%.'tr.$0 ++ $ec%100%.$wfec'.'ec'%100%.'tr.$0;
agent Buffc = $wffc'.$fc%formNum5%.'fc'%formNum5%.'tr.$0 ++ $fc%formNum5%.$wffc'.'fc'%formNum5%.'tr.$0;
agent Buffd = $wffd'.$fd%File%.'fd'%File%.'tr.$0 ++ $fd%File%.$wffd'.'fd'%File%.'tr.$0;
agent Buffn = $wffn'.$fn%accepted%.'fn'%accepted%.'tr.$0 ++ $fn%accepted%.$wffn'.'fn'%accepted%.'tr.$0;
agent Bufgf = $wfgf'.$gf%emptyForm%.'gf'%emptyForm%.'tr.$0 ++ $gf%emptyForm%.$wfgf'.'gf'%emptyForm%.'tr.$0;
agent Bufmcd = $wfmcd'.$mcd%04/01/12%.'mcd'%04/01/12%.'tr.$0 ++ $mcd%04/01/12%.$wfmcd'.'mcd'%04/01/12%.'tr.$
0;
agent Bufmpd = $wfmpd'.$mpd%01/01/12,04/01/12,10/01/12%.'mpd'%01/01/12,04/01/12,10/01/12%.'tr.$0 ++ $mpd%01/
01/12,04/01/12,10/01/12%.$wfmpd'.'mpd'%01/01/12,04/01/12,10/01/12%.'tr.$0;
agent Bufsff = $wfsff'.$sff%filledForm%.'sff'%filledForm%.'tr.$0 ++ $sff%filledForm%.$wfsff'.'sff'%filledFor
m%.'tr.$0;
agent Composition = (AD | ARG | RS | Bufce | Bufec | Buffd | Buffn | Buffc | Bufgf | Bufmcd | Bufmpd | Bufsf
f | T0 | T1 | T2 | T3 | T4 | V)\S;
agent RS = 'fd%File%.'fc%formNum5%.'wfgf'.$gf'%emptyForm%.'startT1.($stopT1.'sff%filledForm%.'wfmpd'.$mpd'%0
1/01/12,04/01/12,10/01/12%.'mcd%04/01/12%.'wfce'.$ce'%BMW%.'wffn'.$fn'%accepted%.'tr.$0 ++ $'sff%filledForm%
.('wfmpd'.$mpd'%01/01/12,04/01/12,10/01/12%.'mcd%04/01/12%.'wfce'.$ce'%BMW%.'wffn'.$fn'%accepted%.'tr.$0 | $
stopT1.$0));
agent T0 = $startT0.72.'stopT0.$0;
agent T1 = $startT1.36.'stopT1.$0;
agent T2 = $startT2.24.'stopT2.$0;
agent T3 = $startT3.24.'stopT3.$0;
agent T4 = $startT4.168.'stopT4.$0;
agent V = $tr.$tr.$tr.$tr.$tr.$tr.$tr.$tr.$tr.$tr.$tr.$tr.'composable.0;

** Action sets **
set S = {ce%BMW%,ce'%BMW%,ec%100%,ec'%100%,fc%formNum5%,fc'%formNum5%,fd%File%,fd'%File%,fn%accepted%,fn'%ac
cepted%,gf%emptyForm%,gf'%emptyForm%,mcd%04/01/12%,mcd'%04/01/12%,mpd%01/01/12,04/01/12,10/01/12%,mpd'%01/01
/12,04/01/12,10/01/12%,sff%filledForm%,sff'%filledForm%,startT0,startT1,startT2,startT3,startT4,stopT0,stopT
1,stopT2,stopT3,stopT4,tr,wfce',wfec',wffc',wffd',wffn',wfgf',wfmcd',wfmpd',wfsff'};
```

Fig. 7. CWB objects resulting from modeling the case study

VIII. Mapping TVPCCS Into BPEL

A. General Outline

The translation from BPEL to TVPCCS presented in the first work [15] implicitly preserve the BPEL structure, but the translation from TVPCCS to BPEL does not: TVPCCS is more flexible than BPEL. For example in BPEL there is no message exchanged inside a service, a service can communicate only with other services. In TVPCCS there are no constraints about this. In the TVPCCS design we have to be careful, if we want a simple automatic translation, to write behavior structurally similar to BPEL ones. The TVPCCS programmers have to respect this rule: they have to write the main behavior simply instantiating all the processes representing services, in the usual manner.

At the basis of our mapping there is the correspondence between TVPCCS actions and BPEL interactions. BPEL services and TVPCCS processes instantiated in the main process correspond to each other. From TVPCCS to BPEL, we use the behaviors specified in the process definition to generate the BPEL service description with the names of partner links, port type, operations and variables.

B. Decomposing TVPCCS Expressions

We would like to map TVPCCS process onto a hierarchical decomposition of specific and adequate BPEL constructs. For example the TVPCCS parallel operator can be used to represent BPEL sequence or Flow construct (see Table II), it is important to identify the right construct. We must also recognize complex BPEL constructs; bottom up traversal of the parse tree is used to identify in the TVPCCS process, parts that are suitable to BPEL constructs.

The name of TVPCCS actions stores information that will help us while choosing the “good” construct. Names will carry all information needed to map a TVPCCS process onto “BPEL block”.

C. Actions name

All BPEL basic activities perform interactions between web services. An interaction is characterized by the partner link, the port type, and the operation involved in the two communicating partners. When the process representing a service is defined, an action is simply an emission or a reception. The name of the action must store information (partner link, port type, and operation involved) on the receiver in the emission case, on the sender in the reception case.

TVPCCS emission (resp. reception) action can be used to represent BPEL reply (resp. receive) activity, asynchronous invoke activity or synchronous invoke activity performing two actions (sending a request and receiving a response) corresponds in TVPCCS to en emission followed immediately by a reception (the emission (resp. reception) action can be used for other structured activities, it will be explained later).

We must be able to make difference between TVPCCS emission (resp. reception) action that represents BPEL reply (resp. receive) activity, synchronous invoke activity and …. To do so, a description of the activity is added in the name of the action.

For instance, the following TVPCCS processes:

$P := \overline{rep\_PL\_PT\_x}(z).0$

Will be translated to the BPEL activity:

<reply partnerLink=”PL” portType=”PT” operation=”x” variable=”z”>

$P := \overline{aInv\_PL\_PT\_x}(y).0$

Will be translated to the BPEL activity:

<invoke partnerLink="PL" portType="PT" operation="x" inputVariable="y">

$P := \underline{rec\_PL\_PT\_x}(y)\ .0$

Will be translated to the BPEL activity:

<receive partnerLink=”PL” portType=”PT” operation=”x” variable=”y”>

And:

$P := \overline{sInv\_PL\_PT\_x}(y).\underline{sInv\_PL\_PT\_x}(z).0$

Will be translated to the BPEL activity:

<invoke partnerLink="PL" portType="PT" operation="x" inputVariable="y" outputVariable=”z”>

TVPCCS emission and reception action can also be used in the representation of BPEL assign activity.

$P := \left(\overline{ass\_copy}\ (5).0 \mid ass\_copy\ (x).0\right) \backslash \{ass\_copy\}$

Will be translated to the BPEL activity:

```
<assign … >
    <copy>
        <from expression="5"/>
        <to var="x"/>
    <copy>
</assign>
```

Note that a restriction on the name of the channel is used to ensure that the value “5” will be assigned to the variable “x” and not another variable when using the same channel “ass_copy”, different channel’s names must be used if one don’t want to use the restriction.

D. Translating TVPCCS to BPEL

A parallel processing is identified by processes linked by the parallel operator. If synchronization dependencies between processes are enabled, TVPCCS emission and reception activities will be used to represent BPEL link construct. The name of these activities will be “link” followed by the name of the link (for more details about the translation of the BPEL flow construct and links see [15]).

TABLE II: TVPCCS-BPEL MAPPING

Sample TVPCCS Specification	Sample BPEL Code

$Q_1 \mid Q_2$	<flow>
	  <…activity1…>
	  <…activity2…>
	</flow>

$Q_1 . Q_2$ or $\begin{pmatrix} Q_1 . \overline{seq\_next}(1).0 \\ \mid \\ seq\_next(v).Q_2 \end{pmatrix} \backslash\{seq\_next\}$	<sequence>
	  <…activity1…>
	  <…activity2…>
	</sequence>

A(x) where A(x) := if (x ≥ 0) then Q_1 . A(x)	<while condition= "bpws:getVariableData(x)>=0"> <…activity1…> </while>

$\underline{pick\_onMessage\_PL1\_PT1\_x1}$(m1).$Q_1$
++
$\underline{pick\ \_onMessage\_PL2\_PT2\_x2}$(m2).$Q_2$
++
82.Q3
	<pick ... >
	  <onMessage … variable="m1">
	    <…activity1…>
	  </onMessage>
	  <onMessage ... variable="m2">
	    <…activity2…>
	  </onMessage>
	  <onAlarm>
	    <for>'P3DT10H'</for>
	    <…activity3…>
	  </onAlarm>
	</pick>

if (x ≥ 0) then Q_1
else Q_2
	<if>
	  <condition>
	    $x≥0
	  </condition>
	    <…activity1…>
	  <else>
	    <…activity2…>
	  </else>
	</if>

A sequential composition is identified by two processes linked by the prefixing operator “.” or by parallel operator and one is post fixed by an emission action and the other one is prefixed by a reception action on the same channel. Generally, the prefixing operator is used because it is simplest and more intuitive, however, in some cases only the representation using the parallel operator can be used, as in the following example:

```
<sequence>
   <flow>
      <…activity1…>
      <…activity2…>
   </flow>
   <…activity3…>
</sequence>
```

The flow construct and activity3 are performed sequentially, and the flow construct contains activity1 and activity2 that must be executed concurrently, which means that activity3 starts after the completion of both activity1 and activity2.

The TVPCCS representation is:

$(Q1 \, . \, \overline{seq\_next}\,(1) \, . \, 0 \mid Q2 \, . \, \overline{seq\_next}\,(1) \, . \, 0 \mid seq\_next\,(v1) \, . \, seq\_next\,(v2) \, . \, Q3 \,) \setminus \{seq\_next\}$

Note that B is not well formed TVPCCS process

B:=(Q1 | Q2) . Q3

The BPEL while construct is identified by a process that executes repeatedly contained activities as long as the Boolean condition evaluates to true at the beginning of each iteration.

The BPEL pick construct is identified by processes linked by weak choice operator, with the condition that these processes begin by :

1) A reception action which name's begin by "pick_onMessage", representing the BPEL <onMessage> construct
2) Allowing some passage of time, representing the BPEL <onAlarm> construct.

IX. Related Works

In this paper, we first propose a framework for analyzing the choreography compatibility supporting synchronous or asynchronous communicating services and then present a mapping between TVPCCS and BPEL in order to allow an automatic translation between the two languages that permit designing and verifying in TVPCCS and then translating the specification onto BPEL, consequently we divide related works in two groups:

The first group is interested in the compatibility analysis of a choreography. The compatibility problem is not limited to analyzing message exchange sequences (conversations). In practice, other metrics affect the Web services compatibility, such as the quantitative properties like timed constraints which plays a crucial role in Web services interaction.

In [3], [16], the authors consider the sequence of messages that can be exchanged between two synchronous Web services. But, considering only message exchange sequences is not sufficient. To succeed a conversation, other metrics can have an impact such as timed properties which are not considered in [3], [16]. Another important remark is that in [3], [16], the authors consider synchronous Web services. Such assumption is very restrictive since the nature of Web services can be asynchronous. To overcome this limitation, we propose a compatibility checking approach for timed asynchronous services.

The compatibility framework presented in [6], [14], that is an extension of the framework presented in [4], considers a more expressive timed constraints model. In fact, the authors deal only with synchronous communicating services. Thus, to discover timed conflicts, the authors are based on synchronizing the corresponding timed properties over messages. Therefore, this framework cannot be applied to discover the eventual timed conflicts in case of asynchronous Web services.

The second group The second group is interested in translations into BPEL.

In [17] authors present an approach to automatically map a workflow net onto BPEL using an iterative approach. To support this approach, they implemented a tool to automatically translate Colored Petri Nets into BPEL code.

In [18] the authors extend the common mapping theme between the algebra and BPEL by providing rules for a two-way mapping. They also confirm however, that due to the expressive and flexible structure of process algebra, the mapping from process algebra to BPEL clearly does not preserve the structure of a process.

In [19] the author supports the idea that building the pi-calculus model, check it and only then map it into WS-BPEL seems to be a more effective way to tackle the problem of verification for WS-BPEL systems.

X. Conclusion

In this paper, we presented a formal framework for analyzing the compatibility of choreography. Unlike the proposed approaches, this framework caters for timed properties of synchronous and asynchronous Web services. We presented how to model the timed behavior of Web services using TVPCCS. In choreography, when the services are interacting together, timed conflicts could arise. In order to handle the eventual timed deadlocks, we proposed an approach which is based on the model checker CWB.

In order to handle asynchronous services, we proposed to add a process acting like a buffer that receives this message, sends it and finally terminate.

We also present a method to generate BPEL specifications from TVPCCS processes. We argue that the automatic generation of BPEL code is a promising way, it allows modeling in a formal language, checking and then generating the BPEL specification. It also saves time and effort, we have presented a translation of BPEL into TVPCCS [15], then we can translate a BPEL specification, check it, correct it, apply hierarchical refinement design method, and finally the BPEL corrected code is automatically generated. This approach is useful also for reverse engineering issues, and when one wants to verify BPEL services developed by others.

References

[1] R. Kazhamiakin, P. K. Pandya, and M. Pistore, "Timed modeling and analysis in web service compositions," In *Proc. 1st International Conf. Availability, Reliability and Security, ARES, IEEE Computer Society*, 2006, pp. 840-846.

[2] H. Huang, W.T. Tsai, R. Paul, and Y. Chen, "Automated model checking and testing for composite web services," In *Proc. 8th IEEE International Symposium. Object-oriented Real-time distributed Computing (ISORC),* Seattle, May 2005, pp. 300-307.

[3] L. Bordeaux, G. Salaün, D. Berardi, and M. Mecella. "When are two web services compatible? ," In *Proc. 5th International Workshop on*

Technologies for E-Services (TES'04), Toronto, Canada, 2004, pp. 15-28.

[4] B. Benatallah, F. Casati, J. Ponge, and F. Toumani. "On temporal abstractions of web service protocols," In *Proc. 17th Conf. Advanced Information Systems Engineering (CAiSE '05)*, Porto, Portugal, 2005.

[5] R. Kazhamiakin, P. K. Pandya, and M. Pistore. "Representation, verification, and computation of timed properties in web service compositions," In *Proc. The IEEE International Conf. Web Services (ICWS)*, 2006, pp. 497-504.

[6] J. Ponge, B. Benatallah, F. Casati, and F. Toumani. "Fine grained compatibility and replaceability analysis of timed web service protocols," In *Proc. 26th International Conf. Conceptual Modeling (ER)*, Auckland, New Zealand, 2007, pp. 599-614

[7] N. Guermouche, O. Perrin, and C. Ringeissen. "Timed specification for web services compatibility analysis," In *Proc. International Workshop on Automated Specification and Verification of Web Systems (WWV'07)*, San Servolo island, Venice, Italy. December 14, 2007, pp. 155-170.

[8] F. Liu, L. Zhang, Y. Shi, L. Lin, and B. Shi. "Formal analysis of compatibility of web services via CCS," In *Proc. 1st International Conf. Next GenerationWeb Services Practices, IEEE Computer Society*, 2005. pp. 143-148.

[9] L. Bao, W.S. Zhang and X.G. Zhang, "Describing and verifying web service using CCS," In *Proc. 7th International conf. Parallel and Distributed Computing, Applications and Technologies (PDCAT '06)*, 2006, pp. 421-426.

[10] J. Li, J. He, H. Zhu, and G. Pu. "Modeling and verifying web services choreography using process algebra," In *Proc. 31st IEEE Software Engineering Workshop*, *IEEE Computer Society*, Washington, DC, USA, 2007, pp. 256-268.

[11] Yeung, W. L., Wang, J., and Dong, W. "Verifying choreographic descriptions of web services based on CSP," In *Proc. IEEE Services Computing Workshops (SCW'06), IEEE Computer Society.* Washington, DC, USA, 2006, pp. 97-104.

[12] F. Moller and C. Tofts. "A temporal calculus of communicating systems," In *Proc. CONCUR 90, Amsterdam, volume 458 of Lecture Notes in Computer Science.* 1990, pp. 401-415.

[13] N. Guermouche and C. Godart. "Uppaal based approach for compatibility analysis of synchronous web services," INRIA, Research report, 2008.

[14] J. Ponge. "A new model for web services timed business protocols,". In *Proc. Atelier (Conception des systèmes d'information et services Web) (SIWS-Inforsid)*, Hermès, Hammamet, Tunisie, Mai 2006, pp. 15-28.

[15] M. L. Beggar, L. Liao, and S. Zhang, "Transforming BPEL into TVPCCS for web services testing," In *Proc. IEEE International Conf. on Computational Intelligence and Software Engineering (CiSE 2011)*, Wuhan, china, 2011.

[16] B. Benatallah, F. Casati, and F. Toumani. "Analysis and management of web service protocols," In *Proc. 23rd International Conf. Conceptual Modeling,* Shanghai, China, November 2004.

[17] W. M. P. van der Aalst and K.B. Lassen, "Translating workflow nets to bpel4ws," BETA Working Paper Series, Eindhoven University of Technology, Eindhoven, 2005.

[18] A. Ferrara, "Web services: a process algebra approach," In *Proc. 2nd international conf. Service oriented computing*, New York, NY, USA, 2004, pp. 242-251.

[19] F. Abouzaid, "A mapping from pi-calculus into BPEL," In *Proc. the conf. Leading the Web in Concurrent Engineering.* 2006, pp. 235-242.

Mohammed Lamine Beggar was born in 1983, is currently a PhD student at school of Computer Science and Technology, Beijing Institute of Technology. His research interests include Analysis of Web Services Composition, and model checking.

Liao Lejian is currently a professor at School of Computer Science and Techno-logy, Beijing Institute of Technology. He is an expert semantic web. His research interests include security protocol analysis and design web service, semantic web, model checking and logic. E-mail: liaolj@bit.edu.cn