MODELLING WEB SERVICES INTEROPERABILITY

S. Haddad* and T. Melliti* and P. Moreaux$^{*\,+}$ and S. Rampacek$^+$

$(*)$*LAMSADE* — $(+)$*LERI-RESYCOM*

Université Paris Dauphine, France — *Université de Reims Champagne-Ardenne, France*

Email: {haddad,melliti,moreaux}@lamsade.dauphine.fr - sylvain.rampacek@univ-reims.fr

Key words: Web service, Algebra of timed processes, Timed Labelled Transition systems, Timed automata, Synthesis algorithm

Abstract: With the development of the semantic Web, the specification of Web services has evolved from a “remote procedure call” style to a behavioral description including standard constructors of programming languages. Such a transformation introduces new problems since traditional clients will not be able to interact with these sophisticated services. In this work, we develop a generic agent capable to fully control the interaction process with a Web service given its XLANG behavioral description (XLANG being one of these languages). At first, we give an operational semantic to XLANG in terms of timed transition systems. Then we define a relation between two communicating systems which formalizes the concept of a correct interaction and we propose an algorithm which either detects ambiguity of the Web service or generates a timed deterministic automaton which controls the agent behavior during the interaction with the service. Starting from these theoretical developments we have built a platform which ensures to a user the correct handling of any complex Web service dynamically discovered through the Web.

1 INTRODUCTION

Web services are “self contained, self-describing modular applications that can be published, located, and invoked across the Web" (Tidwell, 2000). They are based on a set of independent open platform standards to reach a high level of acceptance. Web services framework is divided into three areas - communication protocol, service description, and service discovery - and specifications are being developed for each one: the "Simple Object Access Protocol" (SOAP)(SOAP, 2000), which enables communication among Web Services, the "Universal Description, Discovery and Integration" (UDDI)(UDDI, 2002), which is a registry of Web Services descriptions and the "Web Services Description Language" (WSDL)(WSDL, 2001), which provides a formal, computer-readable description of Web services. The latter describes such software components by an interface listing the collection of operations that are network accessible through standard XML messaging (CaudWell and al, 2001). This description contains all information that an application needs to invoke such as the message structure, the response structure and some binding information like the transport protocol, the port address, etc.

However simple operation invocation is not sufficient for some kind of composite services. They require in addition a long-running interaction derived by an explicit process model. This kind of services may often be encountered in two cases. At first when a web service is developed as an agent, it is composed by a set of accessible operations and a process model which schedules the invocation to a correct use of the service. Secondly, facing to the capability limits of Web services, composite services may be obtained by aggregating existing Web services in order to create more sophisticated services (and this in a recursive way).

In order to deal with the behavioural aspects of complex services, some industrial and academic specifications languages have been introduced such as WSFL (Bechhofer and al, 2001), XLANG (Thatte, 2001), WSCL (WSCL, 2002) and more recently BPEL4WS and DAML-S (Ankolekar and al, 2001). Each of them is directly based on top of WSDL. They propose different schemas to glue such services operations according to a process model(Staab et al., 2003). Such specification languages were adopted by tools such as eFlow, SCET[1] for supporting the specification of the composition. A composition tool is mostly defined by three components: a composition module which offers an intuitive interface to plan Web services (data and control flows), an execution entity which controls the services invocation according to the specification model and finally an efficiency evaluator(Casati et al., 2000)(Chandrasekaran et al., 2003).

While composite services are recursively considered as Web services, they introduce new problems since traditional clients will not be able to assure interoperability and sophisticated interaction protocols. Facing with the increasing complexity of the behaviour of such services, the need of a tool for user assistance in interaction processes becomes a necessity.

In this work, we develop a platform capable to fully control the interaction process with a Web service given its process model specification. This work is essentially based on a previous paper which focuses on algorithmic aspect(Melliti and Haddad, 2003). Building such platform raises the following two issues: the first one concerns semantics and algorithmic aspects whereas the second one is related to technological aspects.

Since our goal is to produce a client behaviour which correctly interacts with the service, we have to formally define such an interaction. But this requires beforehand to specify a formalism of representing the service and the client behaviour; composite Web service behaviour may be represented in a variety of ways. While digraphs have been used widely to model business processes (Casati et al., 2000), other models like Petri nets (W. Aalst and Houben, 1994), activity/state charts are also being employed. In our system, we represent the behaviour of a Web service process by a timed transitions system. The timed transitions system models a communicating system where the actions are mainly message exchanges and time passing. In order to design a client behaviour synthesis algorithm, we propose an interaction relation between two communicating systems which formalizes the concept of a correct interaction. Let us briefly explain why the two main ones - the language equivalence and the bisimulation equivalence - do not match our needs. The language equivalence is unable to express the different branching capabilities of the systems since it does not require an equivalence relation between the intermediate states of the two systems. The bisimulation equivalence does not take into account the different nature of the events: in an asynchronous communicating system the sending of a message is an action whereas the reception is a reaction. Thus the interaction relation that we introduce can be viewed as a bisimulation relation modified in order to capture the nature of the events.

[1]it uses WSFL for dynamic Web services Composition

In addition to the theoretical difficulties we have solved, we have also faced technical difficulties during the development and execution of such composite services. Indeed the existing technologies for Web services have two drastic restrictions on interaction models. The first one refers to the synchronous and asynchronous nature of the interaction and the second one concerns the difference between targeted interaction like RPC and message oriented interaction(B. Benatallah and Rabhi, 2001)(Curbera et al., 2001). Our platform supports all interaction patterns such as centralized and decentralized execution, message oriented and RPC model, synchronous and asynchronous interaction. However in the asynchronous interaction mode, the platform only supports Web services using a Java XML message provider. Let us note that a technical solution bridging the JAXM which other asynchronous interaction providers is possible but it is out of the scope of the present work.

The balance of the paper is the following one. At first, we define a formal semantic for a Web service description language as a timed transitions system. Then we propose a relation between two communicating transition systems which formalizes the concept of a correct interaction between two processes. Based on this relation, we present a client behaviour synthesis algorithm which either produces a timed transitions system for scheduling a correct interaction with the service or detects the ambiguity of the service specification. Afterwards, we focus on the platform specification and implementation. More precisely, we describe the different steps of the process which binds our client with a composite Web service. We also detail the components of our platform. Then we present an illustrative example. Finally we discuss our approach and we give future improvements of this work.

2 A formal semantic for XLANG

XLANG is an XML block-structured specification which offers a set of flow control primitives in order to define the process model of the Web service. The flow control primitives organize the operation execution exactly like the different primitives that we meet in programming languages. An XLANG description is always built on one or more WSDL description which supplies a set of operations. It uses their operations as the basic elements in order to construct the processes. An XLANG process is built by applying control primitives on operations and XLANG subprocesses. Every flow control primitive represents a specific execution order model to the XLANG processes and the WSDL operations according to a specific semantic. In addition to flow control primitive XLANG offers a set of primitives to structure the processes organization by defining an execution context for a set of processes or transactions. We have chosen to deal with the main constructors of this language. The forgotten ones either has an unclear semantic or are of minor interest. For instance we handle the delayFor constructor and skip the delayUntil since it is unusual to specify a fixed date in a service. Rather than following the XML syntax of XLANG, we have chosen to delete the syntactic sugar in order to manage compact expressions. We now present the models that we use for the formalization of the XLANG semantics.

2.1 A quick overview of some formal models

Algebra of timed processes XLANG provides a set of operators describing in a modular way the observable behaviour of a Web service. In fact, this approach is close to the process algebra paradigm illustrated for instance by CCS (Milner, 1989), CSP (C.A.R.Hoare, 1985) and ACP (Bergstra and Klop, 1984). The main objective of the process algebra approach is to cope with the complexity of the conception of parallel systems. In order to achieve this goal, the theoretical developments related to a process algebra generally consists in four steps(Lee et al., 1994). At first one defines a set of operators and syntactic rules for constructing processes (e.g. what we have done in the previous subsection). Then one associates to each operator a set of semantic rules which assign to a process a behavioural interpretation. In order to compare different processes, one introduces some equivalence relations and congruences which express that two processes (or components) have a similar behaviour w.r.t. to different criteria. At last one develops algorithms which decide the equivalence of two processes, working at the syntactical level (e.g. via a set of algebraic laws) or at the semantical level (e.g. with techniques like model-checking).

Since time is an important issue in such systems, the process algebra model has been enlarged by introducing discrete time passing. The discrete time models are usually defined by a special transition representing one unit time passing(Nicollin and Sifakis, 1991). Thus it appears that the syntactic features of XLANG make it a good candidate to be an algebra of timed processes. Beforehand, we give the elements necessary to this semantic.

Labelled Transition System A labelled transition system is an oriented graph where the nodes represent the possible states of the system (with an initial state) and the arcs represent the state transitions. Each arc is labelled by the action whose occurrence has triggered this transition. Depending on the process algebra language, some labels have a special meaning. We will detail our alphabet later.

Definition 1 *A labelled transition system LTS is defined by a tuple $LTS = (S, L, \rightarrow, s_0)$ where:*

- *S is a set of states with $s_0 \in S$ the initial state*
- *L is a finite set of labels*
- *$\rightarrow \subseteq S \times L \times S$ is the transition relation*

Transition rules A LTS is the representation of the behaviour of a process. The states of the process are simply the current process after some part of an execution. To each operator op, one associates a set of transition rules which define the possible behaviour of a process whose outer constructor is op. Let us suppose that we want to define a rule $[op_x]$ for a generic process $P = op(P_1, P_2, \ldots)$. At first, we have a boolean expression over some potential transitions of selected components of P: $Bexp(\{P_{o(i)} \xrightarrow{\alpha_i} P'_{o(i)}\})$. This condition is enforced by a second condition on the occurring labels denoted $guard(\{\alpha_i\})$. If the two conditions are fulfilled then a state transition for P is possible where the label $Lexp(\{\alpha_i\})$ is an expression depending on the labels of sub processes transition and the new state is an expression $Nexp(P, \{P'_{o(i)}\})$ depending on the original process and the new sub processes. Below, a generic rule is presented with the usual style.

$$[op_x] : \frac{Bexp(\{P_{o(i)} \xrightarrow{\alpha_i} P'_{o(i)}\})}{P \xrightarrow{Lexp(\{\alpha_i\})} Nexp(P, \{P'_{o(i)}\})}$$
$$\text{where } guard(\{\alpha_i\})$$

2.2 Semantic rules for XLANG

In the sequel, we will complete the XLANG algebra with an operational semantic as the first step for the development of our platform. Here we describe the events of a LTS associated to an XLANG specification:

- The set of types of messages will be denoted M. There are two events associated to a message m: the emission denoted by $!m$ and the reception denoted by $?m$. We also denote $!M = \{!m \mid m \in M\}$ and $?M = \{?m \mid m \in M\}$ and the joker character $*$ may be substituted by ! or ?.
- Since the service may evolve in an unobservable way (e.g. the evaluation of a condition) we introduce τ, the internal action.
- Since XLANG takes into account the time, χ denotes one unit time passing. We have chosen to represent time passing by units because the time constraints of a Web service are generally "soft" and thus the discretization of time is a valid abstraction.

- The exception event set of XLANG is denoted by E.
- In order to control that the client correctly detects the end of the service, we introduce $\surd$ the termination event. This action will also simplify the definition of the operational semantic.

The basic processes The process $?o[m]$ (which corresponds to the input operation of WSDL) consists in receiving a message of type m. The process $!o[m]$ (which corresponds to the notification operation of WSDL) consists in sending a message of type m. We consider only these two types of WSDL operations. The two other types can be built with the sequence constructor (see below). The raise process $r[e]$ simply raises an exception e which must be handled in some way (see below the context process). Since we consider that due to internal or external conditions, any basic action of a process can be delayed, the behaviour of the basic processes is specified by the following rules:

$$*o[m] \xrightarrow{\chi} *o[m] \text{ and } *o[m] \xrightarrow{*m} empty \text{ where } * \in \{!, ?\} \text{ and } r[e] \xrightarrow{e} empty$$

The sequence process and the empty process The process $P;Q$ executes the process P followed by the process Q. Since the operator ";" is associative, we safely restrict the number of operands to two processes. The *sequence* process acts at its first subprocess while this process does not indicate its termination. In the latter case, the *sequence* process becomes the second process in a silent way.

$$\frac{P \xrightarrow{a} P' \wedge \neg P \xrightarrow{\surd} P''}{P;Q \xrightarrow{a} P';Q} \text{ where } a \neq \surd \text{ and } \frac{P \xrightarrow{\surd} P'}{P;Q \xrightarrow{\tau} Q}$$

The empty process *empty* does nothing; it is similar to the *skip* instruction of some languages. It can also be interpreted as the neutral element for the operator ";". It is different from the null process 0. After some units of time it indicates its termination and then becomes 0.

$$empty \xrightarrow{\chi} empty \text{ and } empty \xrightarrow{\surd} 0$$

The switch, the while and the all process The process $switch(\{c_i, P_i\}_{i\in I})$ chooses to behave as one process among the set $\{P_i\}$. Each branch of its execution is guarded by an **internal** condition denoted by a qualified name (c_i). The conditions are evaluated w.r.t. the order of their appearance in the description. However since the client has no mean to predict the choice of the service, this order is irrelevant. The main consequence is that from the point of view of the client, this choice is non deterministic. The *switch* process becomes one of its sub-processes in a silent way. Let us note that we have implicitly supposed that at least one condition is fulfilled. In the other case, it is enough to add the process *empty* as one of the sub-processes.

$$\forall i \in I \; switch(\{c_i, P_i\}_{i\in I}) \xrightarrow{\tau} P_i$$

The process $while(c, P)$ iterates an inner process while an **internal** condition c is satisfied. Like the *switch* process, the *while* process evaluates in a silent way its condition. Thus we have two rules depending on this internal evaluation.

$$while(c,P) \xrightarrow{\tau} P; while(c,P) \text{ and } while(c,P) \xrightarrow{\tau} empty$$

The process $all(\{P_i\}_{i\in I})$ simultaneously activates a set of processes $\{P_i\}$. XLANG does not include synchronization primitives since it considers synchronization as an internal action unobservable by the client. This parallel execution is similar to a "fork join" in the sense that the combined process finishes its interaction when all the sub-processes have achieved their execution. The subprocesses of a *all* process act independently except for two actions. They simultaneously let pass a unit of time and they simultaneously indicate their termination. In the latter case, the *all* process becomes the null process.

$$\frac{\exists j \in I \; P_j \xrightarrow{a} P'}{all(\{P_i\}_{i\in I}) \xrightarrow{a} all(\{P_i\}_{i\in I\setminus\{j\}} \cup P')} \text{where } a \notin \{\chi, \surd\}$$

$$\frac{\forall i \in I \; P_i \xrightarrow{\chi} P'_i}{all(\{P_i\}_{i\in I}) \xrightarrow{\chi} all(\{P'_i\}_{i\in I})} \text{ and } \frac{\forall i \in I \; P_i \xrightarrow{\surd} P'_i}{all(\{P_i\}_{i\in I}) \xrightarrow{\surd} 0}$$

The pick process and the context process The pick process $pick[\{m_i, P_i\}_{i\in I}, \{d, Q\}, \{e_j, R_j\}_{j\in J}]$ manages a condition race between sub-processes based on timing or triggers. It contains one or more event handler sub-blocks. Each event handler associates a specific service behaviour to an occurrence of the corresponding event. The possible kinds of event are the reception of an expected message (m_i), the triggering of a time-out whose duration is expressed w.r.t. to some time unit by an integer d (delay actions) or the raising of some exception e_j. When some event happens the service behaves as the associated process (P_i, Q or R_j). The "time" event introduces a watchdog for reception of messages. There is at most one such event in the construction. The specification of catching processes is authorized only if the pick process is the exception part of a context process.

The context process $[P, E]$ has different roles but here we only describe the handling of exceptions. Each context contains an (optional) exception process E which is a pick process. The exception process has catching sub-processes which intercept the raised exceptions during the current context execution or during a nested one if the raised event has not been previously catched. The "time" action of the exception block is a watchdog for the context execution delay. Similarly, cancelling or aborting messages can be handled by this construction. Due to space considerations, we only give a semantic to the $context$ process and then we will informally explain the semantic of a $pick$ process when it is not an exception process. In the following rules, E the $exception$ process is an abbreviation for the process $pick(\{m_i, P_i\}_{i\in I}, \{d, Q\}, \{e_j, R_j\}_{j\in J})$.

This rule expresses that the exception block may be triggered by the reception of the expected messages.

$$[P, E] \xrightarrow{?m_i} P_i$$

The next rules specify that the time elapses under the control of the watchdog.

$$\frac{P \xrightarrow{\chi} P' \wedge \neg P \xrightarrow{\surd} P''}{[P, E] \xrightarrow{\chi} [P', pick(\{m_i, P_i\}_{i\in I}, \{d-1, Q\}, \{e_j, R_j\}_{j\in J})]} \text{ where } d > 1$$

$$\frac{P \xrightarrow{\chi} P' \wedge \neg P \xrightarrow{\surd} P''}{[P, E] \xrightarrow{\chi} Q} \qquad \text{where } d = 1$$

Let us note that we can adapt the three previous rules in order to determine the behaviour of a $pick$ process which is not an $exception$ process. The adaptation consists in deleting the conditions related to P. The following two rules handle the case of a raised exception depending on whether the $context$ process has a sub-process to handle this exception. If it is not the case the exception is “transmitted” to the including context block.

$$\frac{P \xrightarrow{e_j} P'}{[P, E] \xrightarrow{\tau} R_j} \text{ and } \frac{P \xrightarrow{e} P'}{[P, E] \xrightarrow{e} empty} \text{where } e \in E\backslash\{e_j\}_{j\in J}$$

The last rules describe the actions of P inside the $context$.

$$\frac{P \xrightarrow{\surd} P'}{[P, E] \xrightarrow{\tau} empty} \text{ and } \frac{P \xrightarrow{a} P'}{[P, E] \xrightarrow{a} [P', E]}$$

3 Generation of an interacting client

Using the previous rules, starting from a XLANG specification we develop the LTS related to its behaviour. Although we will not prove it here, this LTS has a finite number of states.

We want to specify the behaviour of an agent able to correctly interact with the service. Obviously we choose the formalism of the labelled transition systems for the representation of this behaviour. We remark that this LTS must be **deterministic** in order to be implementable.

Now we proceed in two steps. At first, we need to formally define what is a correct interaction between two LTS. Once this relation is defined, we develop an algorithm producing the LTS of the client behaviour if such a behaviour exists or detecting the ambiguity of the Web service.

3.1 An interaction relation

As usual in the LTS formalism, we define an observable transition relation between states given by $s \overset{a}{\Rightarrow} s'$ iff $s \xrightarrow{\tau^* a \tau^*} s'$ and $s \overset{\epsilon}{\Rightarrow} s'$ iff $s \xrightarrow{\tau^*} s'$. Moreover we suppose that the exception events are not observable in the LTS of the service. If it is the case, it means that the service does not catch an exception and then must be modified.

We now derive the interaction relation from general considerations. Let us focus to some instant of the execution. If one LTS is able to send a message (action $!m$), the other one must be able to receive this message (action $?m$). If one LTS is able to let the time pass (action χ), the other one must also be able to let the time pass (action χ). At last, if one LTS is terminating (action $\surd$), the other one must also be able to terminate (action $\surd$).

The subtle point is about the reception of a message. Suppose that one LTS expects the reception of $?m$, does it mean that the other one is able to send this message? The answer is not necessary since the latter LTS may evolve in an indistinguishable way from one state to two states, one where it is able to send m and the other one where it is not. However we require that in the other state, it is able to send a message in order to avoid an infinite waiting of the first LTS.

We introduce the following notation $?m^c = !m$, $!m^c = ?m$ and $\forall a \notin \{!m\}_{m\in M} \cup \{?m\}_{m\in M}$ $a^c = a$.

Definition 2 *Let $LTS_1 = (S_1, L_1, \rightarrow_1, s_{01})$ and $LTS_2 = (S_2, L_2, \rightarrow_2, s_{02})$ be two labelled transition systems. Then S_1 and S_2 correctly interact iff $\exists \sim \subseteq S_1 \times S_2$ such that:*

- $s_{01} \sim s_{02}$
- $\forall s_1, s_2$ *such that* $s_1 \sim s_2$
  - *Let $a \notin \{?m\}_{m\in M}$, if $\exists s_1 \overset{a}{\Longrightarrow}_1 s_1'$ then $\exists s_2 \overset{a^c}{\Longrightarrow}_2 s_2'$ with $s_1' \sim s_2'$ and if $\exists s_2 \overset{a}{\Longrightarrow}_2 s_2'$ then $\exists s_1 \overset{a^c}{\Longrightarrow}_1 s_1'$ with $s_1' \sim s_2'$*
  - *Let $m \in M$, if $s_1 \overset{?m}{\Longrightarrow}_1 s_1'$ then*
    - $\exists s_2^- \overset{w}{\Longrightarrow}_2 s_2$, $\exists s_2^- \overset{w}{\Longrightarrow}_2 s_2^+$, $\exists s_2^+ \overset{!m}{\Longrightarrow}_2 s_2'$ *with* $s_1 \sim s_2^+$ *and* $s_1' \sim s_2'$ *where w is a word*
    - $\exists s_2 \overset{!m'}{\Longrightarrow}_2 s_2'$
  - *Let $m \in M$, if $s_2 \overset{?m}{\Longrightarrow}_2 s_2'$ then*
    - $\exists s_1^- \overset{w}{\Longrightarrow}_1 s_1$, $\exists s_1^- \overset{w}{\Longrightarrow}_1 s_1^+$, $\exists s_1^+ \overset{!m}{\Longrightarrow}_1 s_1'$ *with* $s_1^+ \sim s_2$ *and* $s_1' \sim s_2'$ *where w is a word*
    - $\exists s_1 \overset{!m'}{\Longrightarrow}_1 s_1'$

3.2 The synthesis algorithm

The algorithm builds the deterministic LTS (LTS_1) following a kind of determinization of the LTS

(LTS_2) of the service. Each state of the potential client is associated to a subset of states of the service. There is a stack of couples (s_1, S'_2) to be processed where s_1 is a new state of the client and where S'_2 is a subset of states of the service which are related to s_1 via the interaction relation. Let us describe one step of the algorithm.

- At first, one completes S'_2 with $\overset{\epsilon}{\Longrightarrow}_2$ transitions.
- If a state of the client s'_1 is already associated to S'_2 then one redirects all the input edges of s_1 to s'_1 and one deletes s_1.
- Otherwise for each $s_2 \overset{a}{\Longrightarrow}_2 s'_2$ with a and $s_2 \in S'_2$, one builds a new vertex s'_1 and a new edge $s_1 \overset{a^c}{\Longrightarrow}_1 s'_1$ and one stacks (s'_1, S^*_2) where $S^*_2 = \{s'_2 | \exists s_2 \in S'_2, \exists s_2 \overset{a}{\Longrightarrow}_2 s'_2\}$
- Let $a \notin \{?m\}_{m \in M}$ such that $s_1 \overset{a}{\Longrightarrow}_1 s'_1$, if there is a $s_2 \in S'_2$ with $\nexists s_2 \overset{a^c}{\Longrightarrow}_2 s'_2$ then stop and return "service ambiguous".
- Let $s_1 \overset{?m}{\Longrightarrow}_1 s'_1$, if there is a $s_2 \in S'_2$ with $\nexists s_2 \overset{!m'}{\Longrightarrow}_2 s'_2$ then stop and return "service ambiguous".

The algorithm starts with the couple of initial states (s_{01}, s_{02}) in the stack and stops either when the stack is empty (i.e. the client has been built) or when it has detected the ambiguity of the service. Again due to space considerations, we do not include the correctness proof of the algorithm.

4 Platform architecture and implementation

Here we describe the architecture of our platform for a dynamic management of clients interacting with Web services. The figure 1 presents the main modules of our system: the service's behaviour generator, the client's behaviour generator, the interaction controller and the communication module.

4.1 The service's and the client's behaviour generators

The construction starts from the description of the service available through its WSDL file. It is made of three steps: the transformation of the WSDL (and XLANG) file describing the service into a DOM tree, the extraction of relevant information from this DOM tree and the computation of the LTS corresponding to the observable behaviour. Let us note that as specified in the previous sections, we do not care of data nor communication support and all activities except message exchange and time passing are modelled as internal (silent) actions.

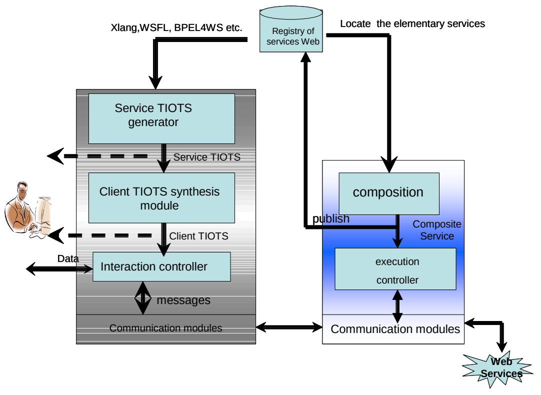

Figure 1: The platform architecture

We first transform the text file of the service description into a DOM tree using an XML parser. Then we do a first analysis of the DOM tree in order to extract information on the links between WSDL operations and XLANG actions and on input and output messages. After this analysis, we generate the LTS guided by a recursive traversal of the DOM tree since the structure of this tree is close to the modular construction of an XLANG specification. However the two way message operations in the description language must be considered as a sequence of one way messages.

The generator of the client applies the algorithm described in the subsection 3.2 with the LTS of the service as input. Once the LTS of the client is built, it applies a kind of factorization of states. Indeed since the handling of time passing leads to numerous states different only w.r.t. by the timing constraints, the factorization tries to incorporate explicit timers inside the LTS in order to aggregate states. The result is in fact a timed automaton. Due to space considerations, we do not detail the factorization algorithm. We illustrate on the following toy example the generation process:

$$S =_{def} [P, E]$$
$$P =_{def} ?op_1; r[e_1]$$
$$E =_{def} pick(\{e_1, !op_2\}, \{2, !timeout\})$$

Figures 2 and 3 respectively present the service and client LTS generated by our platform.

4.2 Interaction controller

This component takes as input the client automaton and a set of binding information from the XLANG description of the service (i.e. WSDL files on which is based the composite service). For each interaction session (initialized by the user), the interaction controller schedulers the user actions during the interaction. It generates a client application instance for each

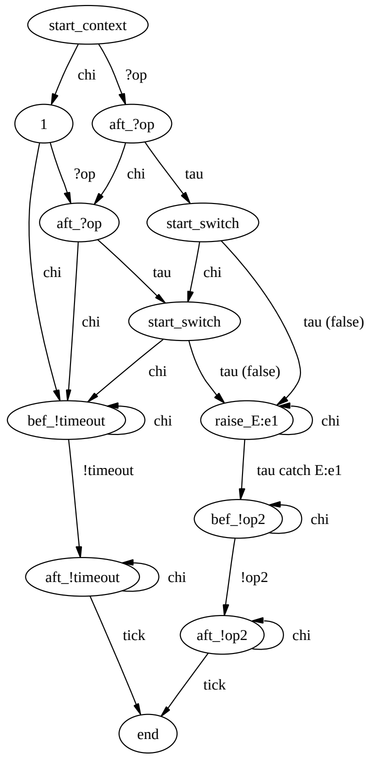

Figure 2: An example of a service LTS

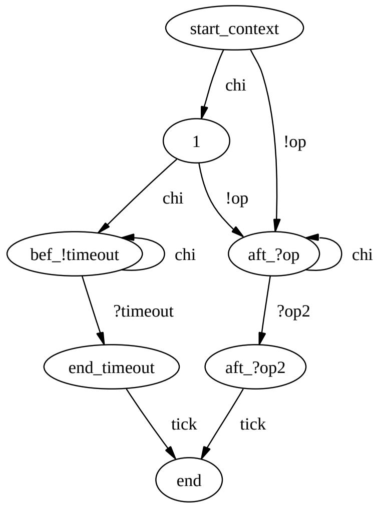

Figure 3: The corresponding client LTS

service instantiation. The application instance is composed by:

- a copy of the client behaviour automaton,
- a set of contextual variables,
- a JAVA RMI[2] client for each WSDL service.

According to the service binding information the interaction controller links each client application to a communication modules. The link consists in associating each sending action to the appropriate communication service and routing the message listener to the correct receiving action in the automaton instance. The components of the communication module vary according to the service execution and interaction models. For synchronous services, centralized or decentralized, the controller generates a proxy for every WSDL service involved in the XLANG description. In the case of an asynchronous service the controller launches the client application instance over a messaging provider in order to assure a peer to peer interaction between the client and the service. At the present time, the Web service technologies support more synchronous interaction than asynchronous ones. Actually, in our platform we support only asynchronous Web services implemented using JAVA API for XML Messaging as the messaging provider. Note that a possibility to bridge the JAXM to other communication platforms exist but is very complex.

4.3 Communication module

The ***interaction controller*** requests the communication module services in order to send and receive messages. Depending on the interaction implementation type (synchronous or asynchronous and centralized or decentralized) the interaction controller uses the binding information included in the services description file to instantiate the communication tools. The communication model is instantiated at runtime. We have developed two APIs which either generate client proxies for WSDL RPC services or instantiate a JAVA XML messaging provider (JAXM) for message oriented services.

The JAXM (WST, 2003) enables to write business applications that support messaging standards based on the SOAP1.1 and SOAP with Attachments specifications. It may additionally support one or more SOAP message Profiles, we use for our platform the SOAP Rooting Protocol profile. The JAXM offers a message provider which is a Web application enables sending and receiving synchronous or asynchronous SOAP messages. Both the client and the services must support message provider in order to realize an asynchronous communication.

[2]Remote Method Invocation

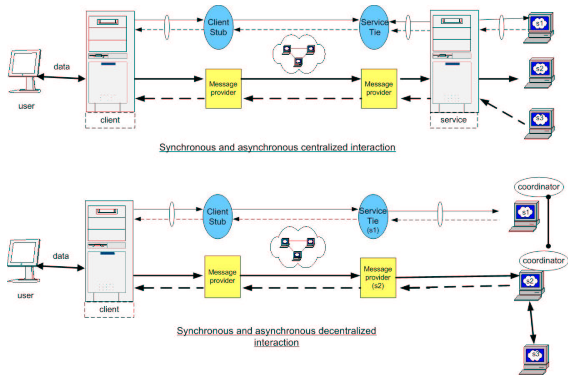

Figure 4: Interaction models supported by the platform

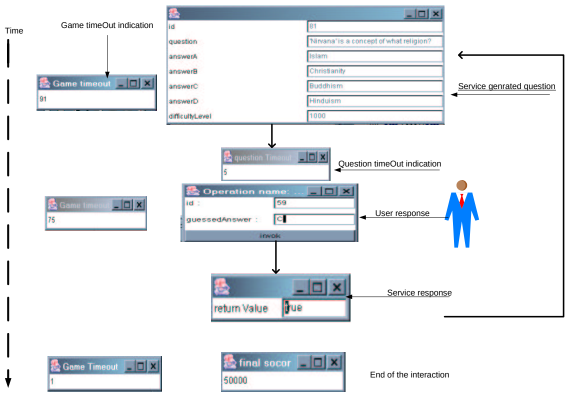

Figure 5: A sample of the service execution sequence

To create an application client that interact within an asynchronous composite Web services the interaction Controller launches the client automaton and its context variable (the application instance thread) in a JAXMServlet and implements the oneWaylistener which root the message to application instance based on the reference correlation. The thread application must obtain the service message provider endpoint and its provider message in order to send and receive SOAP messages. In the figure 4 we present the different communication models supported by the tool.

5 Scenario

In order to test our platform, we have developed a composite Web service. The service offers a quiz game. We orchestrated the operations of an existent Web service and we introduced some kind of time restrictions. The Web service offers three operations:

- ***"getRandomQuestion"*** returns a multiple choice question.
- ***"getRandomQuestionBydifficulty"*** offers to the user the possibility to choose the difficulty of the question.
- ***"checkQuestionResponse"*** takes as parameter the user response with the question reference and returns true or false.

Each user begins the game by sending his identifier. In a limited amount of time, the user answers to a maximum number of questions. In every cycle, the user chooses between selecting a random question or specifying the difficulty level. The time out triggers a new question cycle. The service computes a user score based on the difficulty level and the user response. At the end of the game, the user receives his score and his rank in the hole classification. The composite service is developed as message oriented service while the existing service is an RPC message. We used the JAXM and SOAPRP profile to support asynchronous interaction. In the figure 5 we present the sequence of windows generated by the interaction controller for user assistance. Both the formal semantics and corresponding algorithms are transparent to the user.

6 CONCLUSION

In this work we have coped with the problem of handling a complex Web service given its XLANG behavioural description. We have given an operational semantic to XLANG in terms of timed transition system and formalized the concept of a correct interaction between two communicating systems. Our key algorithm either detects ambiguity of the Web service or generates a timed deterministic automaton which controls the agent behaviour during the interaction with the service. Starting from these theoretical developments we have built a platform which ensures to a user the correct handling of any complex Web service dynamically discovered through the Web.

We are now pursuing this work into two complementary directions. On the one hand, an implicit hypothesis of our synthesis algorithm is that the medium is a perfect one. We are looking for an algorithm which takes into account an expected behaviour of the medium. On the other hand, given some task requested by the user, we want to generate a client simultaneously interacting with different services in order to improve the quality of the results. These parallel interactions could be considered as a dynamical composition of services.

REFERENCES

Ankolekar, A. and al (2001). Daml-s : Semantic markup for web services. In *Proceedings of the International Semantic Web Working Symposium (SWWS)*, Standford, USA.

B. Benatallah, M. Dumas, M. F. and Rabhi, F. (2001). Towards patterns of web services composition. Technical report, Technical Report UNSW-CSE-TR-0111, The University of New South Wales Sydney, Australia. http://citeseer.nj.nec.com/correct/472153.

Bechhofer, S. and al (2001). Web services flow language (wsfl 1.0). Technical report, IBM Corporation. http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.

Bergstra, J. and Klop, J. (1984). Process algebra for synchronous communication. *Information and Control, 60(1-3)*, pages 109–137.

C.A.R.Hoare (1985). *Communicating sequential processes*. Prentice Hall.

Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V., and Shan, M. (2000). Adaptive and dynamic service composition in eflow. In *Proceedings of the International Conference on Advanced Information Systems Engineering*, pages 13–31, Stockholm, Sweden.

CaudWell, P. and al (2001). *Service Web XML Professionnel*. wrax, Paris.

Chandrasekaran, S., Miller, J. A., Silver, G. S., Arpinar, B., and Sheth, A. P. (2003). Composition, performance analysis and simulation of web services. *Electronic Markets: The International Journal of Electronic Commerce and Business Media (EM)Web Services EM Vol.13,No. 2*.

Curbera, F., Nagy, W. A., and Weerawarana, S. (2001). Web services : Why and how? *OOPSLA 2001 Workshop on Object-Oriented Web Services*.

Lee, I., Bremond-Gregoire, P., and Gerber, R. (1994). A process algebraic approach to the specification and analysis of resource-bound real-time systems. *Proceedings of the IEEE*, pages 158–171. citeseer.nj.nec.com/lee94process.html.

Melliti, T. and Haddad, S. (2003). Synthesis of agents for web services interaction. *International Conference Electronic Commerce, workshop*.

Milner, R. (1989). *Communication and Concurrency*. Prentice Hall.

Nicollin, X. and Sifakis, J. (1991). The algebra of timed process, atp: Theory and application. Technical report, Technical Report RT-C26, Institut National Polytechnique De Grenoble.

SOAP (2000). Simple object access protocol (soap) 1.1. Technical report, World Wide Web Consortium. http://www.w3.org/TR/SOAP/.

Staab, S., van der Aalst, W., Benjamins, V. R., Sheth, A., Miller, J. A., Bussler, C., Maedche, A., Fensel, D., and Gannon, D. (2003). Web services: Been there, done that? *IEEE Intelligent Systems: volume 18*, pages 72–85. http://www.tm.tue.nl/it/research/patterns/ieeewebflow.pdf.

Thatte, S. (2001). Xlang: Web services for business process design. World Wide Web page. http://www.gotdotnet.com/team/xml/wsspecs/xlang-c/default.htm.

Tidwell, D. (2000). Web services - the web's next revolution. *IBM developerWorks*.

UDDI (2002). Universal description, discovery and integration. Technical report, OASIS UDDI Specification Technical Committee. http://www.oasis-open.org/cover/uddi.html.

W. Aalst, V. H. and Houben, G. (1994). Modelling workflow management systems with high-level Petri nets. In *Proceedings of the second Workshop on Computer Supported Cooperative Work, Petri nets and related formalisms*, pages 31–50.

WSCL (2002). Web services conversation language (wscl) 1.0. Technical report, World Wide Web Consortium. http://www.w3.org/TR/wscl10/.

WSDL (2001). Web services description language (wsdl) 1.1. Technical report, World Wide Web Consortium. http://www.w3.org/TR/wsdl.

WST (2003). The java web services tutorial. Technical report, Sun Microsystems, Inc. http://www.alphaworks.ibm.com/tech/ettk/.