WWV 2005 Preliminary Version

Verification of Web Services with Timed Automata

Gregorio Diaz [2] Juan-José Pardo [3] María-Emilia Cambronero [4]
Valentín Valero [5] Fernando Cuartero [6]

Departamento de Informática
Universidad de Castilla-La Mancha
Escuela Politécnica Superior de Albacete. 02071 - SPAIN

Abstract

In this paper we show how we can use formal methods for describing and analyzing the behavior of Web Services, and more specifically those including time restrictions. Then, our starting point are Web Services descriptions written in WSCI - WSCDL (XML-based description languages). These descriptions are then translated into timed automata, and then, we use a well known tool that supports this formalism (UPPAAL) to simulate and analyze the system behavior. As illustration we take a particular case study, a travel reservation system.

1 Introduction

Nowadays the society model is changing. Our society is based on the information exchange due to the growth of Internet and Telecommunications. For example in the European Union the annual expenditure on ICT (Information and Communication Technology) amounted to an estimated of more than 500 billion EUR which was approximately 6% of total Gross Domestic Product. And the Internet access has increased for household and enterprises. In 2003, the access level of household to the Internet was 45%. The access of enterpris-

[1] This work has been supported by the CICYT project "Description and Evaluation of Distributed Systems and Application to Multimedia Systems",TIC2003-07848-C02-02 and the UCLM project "Aplicación de Métodos Formales al Desarrollo y Verificación de Web Services"

[2] Email:gregorio@info-ab.uclm.es
[3] Email:jpardo@info-ab.uclm.es
[4] Email:emicp@info-ab.uclm.es
[5] Email:valentin@info-ab.uclm.es
[6] Email:fernando@info-ab.uclm.es

This is a preliminary version. The final version will be published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

ers was higher, reaching in some countries over 90% of all enterprises (source: EUROSTAT [8]).

Due to this change in the society model it becomes necessary to increase the research in the development of systems based in Internet, whose objective is to develop solutions for automating their peer-to-peer collaborations, in an effort to improve productivity and reduce operating costs.

Thus, in the last years some new techniques and languages for developing this kind of distributed systems have appeared, such as the Extensible Markup Language, XML [14], and some new Web Services frameworks [5,9,15] for describing interoperable data and platform neutral business interfaces, enabling more open business transactions to be developed.

Web Services are a key component of the emerging, loosely coupled, Web-based computing architecture. A Web Service is an autonomous, standards-based component whose public interfaces are defined and described using XML [11]. Other systems may interact with a Web Service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.

The Web Services specifications offer a communication bridge between the heterogeneous computational environments used to develop and host applications. The future of E-Business applications requires the ability to perform long-lived, peer-to-peer collaborations between the participating services, within or across the trusted domains of an organization.

The Web Service architecture stack targeted for integrating interacting applications consists of the following components [11]:

- **SOAP[9]:** It defines the basic formatting of a message and the basic delivery options independent of programming language, operating system, or platform. A SOAP compliant Web Service knows how to send and receive SOAP-based messages.
- **WSDL[15]:** It describes the static interface of a Web Service. Then, at this point the message set and the message characteristics of end points are here defined. Data types are defined by XML Schema specifications, which support rich type definitions and allow expressing any kind of XML type requirement for the application data.
- **Registry[5]:** It makes visible an available Web Service and allows the service requesters to discover it by means of relatively sophisticated searching mechanism. It also describes the concrete capabilities of a Web Service.
- **Security layer:** Its goal is to ensure that exchanged informations are not modified or forged in a verifiable manner and that parties can be authenticated.
- **Reliable Messaging layer:** It provides a reliable layer for the exchange of information between parties, guaranteeing the delivery of information with an exactly-once semantics.
- **Context, Coordination and Transaction layer:** It defines interopera-

ble mechanisms for propagating context of long-lived business transactions and enables parties to meet correctness requirements by following a global agreement protocol.

- **Business Process Languages layer[2,3]:** It describes the execution logic of Web Services based applications by defining their control flows (such as conditional, sequential, parallel and exceptional execution) and prescribing the rules for consistently managing their non-observable data.
- **Choreography layer[11]:** It describes collaborations of parties by defining from a global viewpoint their common and complementary observable behavior, where information exchanges occur, when the jointly agreed ordering rules are satisfied.

The Web Services Choreography specification is aimed at the composition of interoperable collaborations between any type of party regardless of the supporting platform or programming model used by the implementation of the hosting environment.

Web Services cover a wide range of systems, which in many cases have strong time constraints (for instance, peer-to-peer collaborations may have time limits to be completed). Then, in many Web Services descriptions these time aspects can become very important. Actually, they are currently covered by the top level layers in Web Services architectures with elements such as time-outs and alignments. Time-outs allow to each party to fix the available time for an action to occur, while alignments are synchronizations between two peer-to-peer parties.

Thus, it becomes important for Web Services frameworks to ensure the correctness of systems with time constraints. For instance, we can think in a failure of a bank to receive a large electronic funds transfer on time, which may result in huge financial losses. Then, there is growing consensus that the use of formal methods, development methods based on some formalism, could have significant benefits in developing E-business systems due to the enhanced rigor these methods bring [10]. Furthermore, these formalisms allow us to reason with the constructed models, analysing and verifying some properties of interest of the described systems. One of these formalisms are timed automata [1], which are very used in practice and there are some well-known tools supporting them, like UPPAAL [6,7,12] and KHRONOS [4].

Then, our goal with this paper is to describe how we can verify Web Services with time constraints using model checking techniques. This verification process starts from the top level layers of Web Services architectures (Business Process Language Layer and Choreography layer). The particular Business Process Language layer that we use here is the Web Service Choreography Interface (WS-CI) [2], and the concrete Choreography Layer that we use is the Web Service Choreography Description Language (WS-CDL) [11]. Therefore, the starting point are specification documents written in WS-CDL and WS-CI. However, these description languages are not very useful for the verification process. Thus, these descriptions are translated into timed automata, and the UPPAAL tool is used to simulate and verify the correctness of the system.

As illustration of this methodology of verification we use a particular case study, which is an airline ticket reservation system, whose description contains some time constraints.

The paper is structured as follows. In Section 2 we present the case study that will be used to illustrate the methodology we propose for the verification of Web Services with time restrictions. In Section 3 we describe WSCI - WSCDL and how they are used to describe the case study. In Section 4 we show how we can model the case study and we use the UPPAAL tool to simulate and verify the system behavior. Finally, the conclusions and the future work are presented in Section 5.

2 Case Study: Travel Reservation System

In this section we present the case study that we consider in order to illustrate our methodology of verification. The scenario consists of three participants: a Traveler, a Travel Agent and an Airline Reservation System, whose behavior is as follows:

A Traveler is planning on taking a trip. Once he has decided the concrete trip he wants to make he submits it to a Travel Agent by means of his local Web Service software (*Order Trip*). The Travel Agent selects the best itinerary according to the criteria established by the Traveler. For each leg of this itinerary, the Travel Agent asks the Airline Reservation System to verify the availability of seats (*Verify Seats Availability*). Thus, the Traveler has the choice of accepting or rejecting the proposed itinerary, and he can also decide not to take the trip at all.

- In case he rejects the proposed itinerary, he may submit the modifications (*Change Itinerary*), and wait for a new proposal from the Travel Agent.
- In case he decides not to take the trip, he informs the Travel Agent (*Cancel Itinerary*) and the process ends.
- In case he decides to accept the proposed itinerary (*Reserve Tickets*), he will provide the Travel Agent with his Credit Card information in order to properly book the itinerary.

Once the Traveler has accepted the proposed itinerary, the Travel Agent connects with the Airline Reservation System in order to reserve the seats (*Reserve Seats*). However, it may occur that at that moment no seat is available for a particular leg of the trip, because some time has elapsed from the moment in which the availability check was made. In that case the Travel Agent is informed by the Airline Reservation System of that situation (*No seats*), and the Travel Agent informs the Traveler that the itinerary is not possible

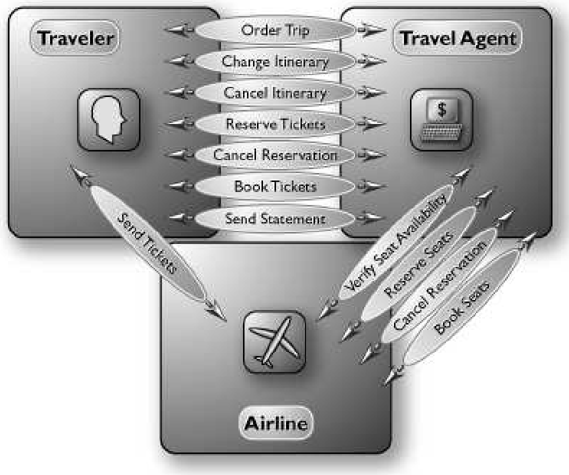

Fig. 1. Flow of the messages exchanged.

(*Notify of Cancellation*). Once made the reservation the Travel Agent informs the Traveler (*Seats Reserved*). However, this reservation is only valid for a period of just one day, which means that if a final confirmation has not been received in that period, the seats are unreserved and the Travel Agent is informed. Thus, the Traveler can now either finalize the reservation or cancel it. If he confirms the reservation (*Book Tickets*), the Travel Agent asks the Airline Reservation System to finally book the seats (*Book Seats*).

According to the previous description, the high level flow of the messages exchanged within the global process (which is called *PlanAndBookTrip*) is that shown in Fig. 1, and a more complete description, including the actions performed by each participant is shown in Fig. 2.

3 The WSCI - WSCDL Description

The Web Services Choreography specification is aimed at being able to precisely describe collaborations between any type of party, regardless of the supporting platform or programming model used by the implementation of the hosting environment. Using the Web Services Choreography specification, a contract containing a "global" definition of the common ordering conditions and constraints under which messages are exchanged is produced that describes, from a global viewpoint, the common and complementary observable behavior of all the parties involved. Each party can then use the global definition to build and test solutions that conform to it. The global specification is in turn realized by combination of the resulting local systems, on the basis of appropriate infrastructure support.

In real-world scenarios, corporate entities are often unwilling to delegate control of their business processes to their integration partners. Choreography offers a means by which the rules of participation within a collaboration can

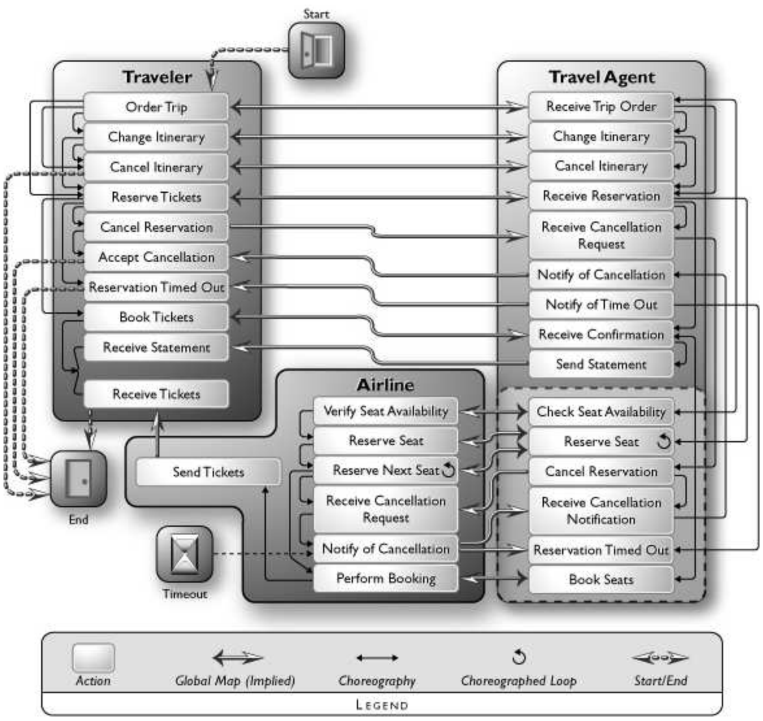

Fig. 2. Overall picture of *PlanAndBookTrip*.

be clearly defined and agreed to, jointly. Each entity may then implement its portion of the Choreography as determined by the common or global view. It is the intent of WS-CDL that the conformance of each implementation to the common view expressed in WS-CDL is easy to determine. Figure 3 shows a possible usage of the Choreography Description Language. In the particular example we are using we take WS-CI as the Business Process Execution Layer (BPEL for short). However, before that we must provide the WS-CDL description.

WS-CDL describes interoperable collaborations between parties. In order to facilitate these collaborations, services commit to mutual responsibilities by establishing relationships. Their collaboration takes place in a jointly agreed set of ordering and constraint rules, whereby information is exchanged between the parties. The WS-CDL model consists of the following entities:

- **Participant Types, Role Types and Relationship Types**: within a Choreography the information is always exchanged between parties within or across trust boundaries. A Role Type enumerates the observable behavior a party exhibits in order to collaborate with other parties. A Relationship Type identifies the mutual commitments that must be made between two parties for them to collaborate successfully. A Participant Type is grouping together those parts of the observable behavior that must be implemented by the same logical entity or organization.
- **Information Types, Variables and Tokens**: Variables contain infor-

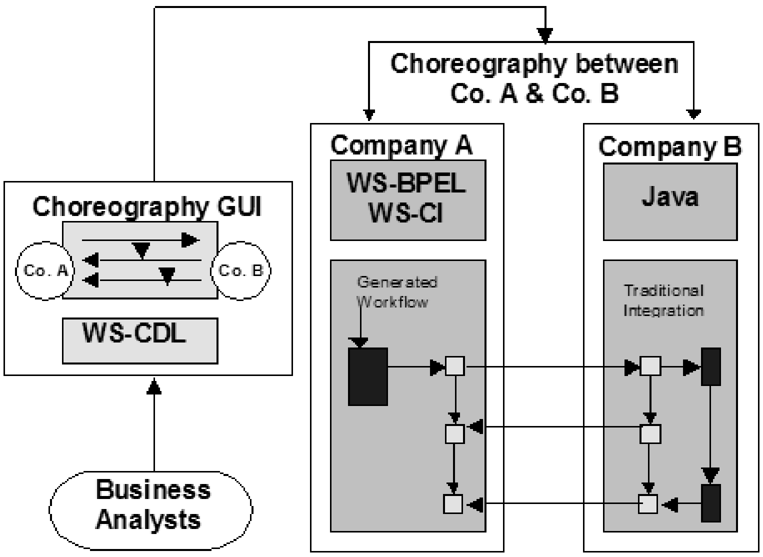

Fig. 3. WS-CDL and WS-CI usage.

mation about commonly observable objects in a collaboration, such as the information exchanged or the observable information of the Roles involved. Tokens are aliases that can be used to reference parts of a Variable. Both Variables and Tokens have Types that define the structure of what the Variable contains or the Token references.

- **Choreographies**: They define collaborations between interacting parties:
  - · **Choreography Life-line**, which expresses the progression of a collaboration. Initially, the collaboration is established between parties, then work is performed within it and finally it completes either normally or abnormally.
  - · **Choreography Exception Block**, which specifies the additional interactions should occur when a Choreography behaves in an abnormal way.
  - · **Choreography Finalizer Block**, which describes how to specify additional interactions that should occur to modify the effect of an earlier successfully completed Choreography (for example to confirm or undo the effect).
- **Channels**: They establish a point of collaboration between parties by specifying where and how information is exchanged.
- **Work Units**: They prescribe the constraints that must be fulfilled for making progress and thus performing actual work within a Choreography.
- **Activities and Ordering Structures**: Activities are the lowest level components of the Choreography that perform the actual work. Ordering Structures combine activities with other Ordering Structures in a nested structure to express the ordering conditions in which information within the Choreography is exchanged.
- **Interaction Activity**: It is the basic building block of a Choreography,

```
  <interaction name="reservation&booking"
               channelVariable="travelAgentAirlineChannel"
               operation="reservation&booking"
               align="true"
               initiate="true" >
    <participate  relationshipType="TravelAgentAirline"
               fromRole="TravelAgent" toRole="Airline" />
    <exchange  name="reservation"
               informationType="reservation" action="request" >
      <send    variable="tns:reservationOrderID" causeException="true" />
      <receive variable="tns:reservationAckID" causeException="true" />
    </exchange>
    <exchange  name="booking" informationType="booking" action="respond" >
      <send    variable="tns:bookingRequestID" causeException="true" />
      <receive variable="bookingAckID" causeException="true" />
    </exchange>
    <timeout time-to-complete="24:00" />
    <record  name="bookingTimeout" when="timeout" causeException="true" />
      <source
        variable="AL:getVariable('tns:reservationOrderCancel', '', '')" />
      <target
        variable="TA:getVariable('tns:reservationOrderCancel', '', '')" />
    </record>
 </interaction>
```

Fig. 4. Part of the WS-CDL specification

which results in an exchange of information between parties and possible synchronization of their observable information changes and the actual values of the exchanged information.

- **Semantics**: It allows the creation of descriptions that can record the semantic definitions of every component in the model.

Figure 4 shows a part of the WS-CDL document that describes our case study. This part shows the relationship between the Airline and the Travel Agent. We can see that this interaction description determines that the maximum time a reservation is available is just of one day.

3.1 WSCI

WSCI is an interface description language. It describes the observable behavior of a service and the rules for interacting with the service from outside. It is not an executable language but it is precise and unambiguous enough.

The observable behavior of each party in a message exchange is described independently of the others.

The basic construct of WSCI is the Action, which is bound to some WS-CDL operation.

The main concepts in WSCI language are the following:

Interface: WSCI maps the description of a web service to the notion of interface.

Activities and choreography of activities: WSCI describes the behavior of a Web Service in terms of choreographed activities. A choreography describes temporal and logical dependencies among activities, where atomic activities represent the basic unit of behavior of a Web Service.

Processes and units of reuse: A process is a portion of behavior labeled with a name. It can be reused by referencing its name.

Properties: It allows us to reference a value within the interface definition. They are the equivalent of variables on other languages.

Context: It describes the environment in which a set of activities is executed. Each activity is defined in exactly one context definition.

Message correlation: It describes how conversations are structured and which properties must be exchanged to do the service correctly.

Exceptions: The definiton of exception is part of the context definition. There are three kinds of exceptions and when an exception occurs the current context must terminate after the activities associated with the exception have been performed.

Transactions and compensation activities .- A transaction asserts that a set of activities is executed in an all-or-nothing way. A transaction may declare a set of compensation activities that will be executed if the transaction has completed successfully, but needs to be undone.

Global model: The global model is described by a collection of interfaces of the participating services and a collection of links between the operations of communicating services.

3.2 Example. Travel Reservation System

We now present the modeling details for the case study under consideration.

3.2.1 Travel Agent Interface

The model for the travel agent has the following elements:

- The main activities of the travel agent are represented via nested processes.
- The iterative processes are described by means of *while* activities.
- We use exceptions to capture the withdrawal of the trip request or the reservation request.
- The interface uses two different correlations, which identify the same conversation involving the travel agent with both the traveler and the airline reservation system.

Figure 5 shows a part of travel agent specification, in which an exception to handle the reservation timeout is defined.

```
...
<context>
 <process name ="BookSeats" instantiation="other">
  <action name="bookSeats"
          role="tns:travelAgent"
          operation="tns:TAtoAirline/bookSeats">
  </action>
 </process>
 <exception>
  <onMessage>
   <action name="ReservationTimeOut"
           role="tns:TravelAgent"
           operation="tns:TAtoAirline/AcceptCancellation">
      <correlate
          correlation="defs:reservationCorrelation"/>
   </action>
   <action name="NotifyOfTimeOut"
           role="tns:TravelAgent"
           operation="tns:TAtotraveler/NotifyofCancellation"/>
                     <fault code="tns:reservationTimedOut"/>
  </onMessage>
  ...
  </exception>
  ...
  </context>
```

Fig. 5. Part of the Travel Agent Specification

3.2.2 Traveler Interface

The main top-level process describing the Traveler is declared with *instantiation=other* attribute to describe the fact that the traveler is actually the entity starting the message exchange. Notice that the model captures the possibility of canceling the reservation or the ticket booking, by means of a new context with a new exception.

We use a correlation to ensure that both the travel agent and the airline reservation system know how to fulfill the correlation requirements exhibited by the traveler interface.

3.2.3 Airline Reservation System

The airline reservation system interface is modeled by an interface with two top-level processes, both with the *instantiation=message* attribute.

The seat reservation for each leg is defined as a transaction which defines a compensation activity which probably will withdraw the reservations for all seats.

Figure 6 shows a part of the specification (the timeout control).

```
 ...
<sequence>
 <context>
  <exception>
   <onTimeout property ="tnsd:expireTime"
              type="duration"
              reference="tns:ReserveSeats@end">
      <compensate name="CompensateReservation"
      transaction="seatReservation"/>
   </onTimeout>
  </exception>
 </context>
   ...
</sequence>
```

Fig. 6. Part of the Travel Agent Specification

4 Modeling, Simulation and Verification

The previous descriptions can be translated into timed automata, thus obtaining three automata, which correspond to the traveler, the travel agent and the airline company. These automata are shown in Figures 7, 8 and 9.

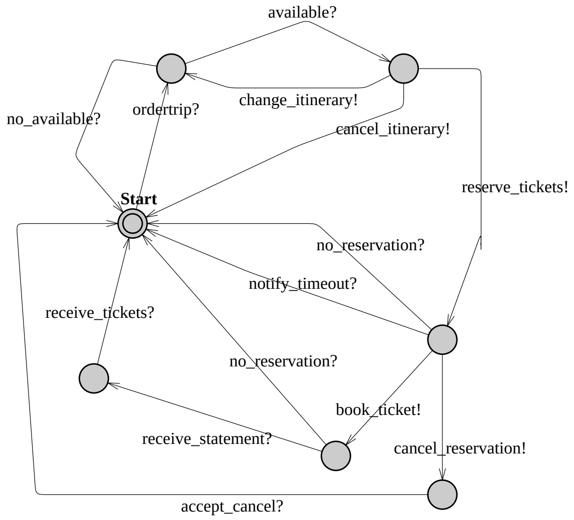

Fig. 7. Timed automaton for Traveler.

Notice the use of clock x in Fig. 8, to control when the reservation expires. This clock is initialized once *reserved_seat* is done.

By means of simulations we can check whether or not the system model holds the expected behavior. These simulations are made by choosing different transitions and delays along the system evolution. At any moment during the simulation, you can see the variable values and the enabled transitions that you can select. Thus, you can choose the transition you want to execute. Nevertheless, you can also select a random execution of transitions, and thus, the

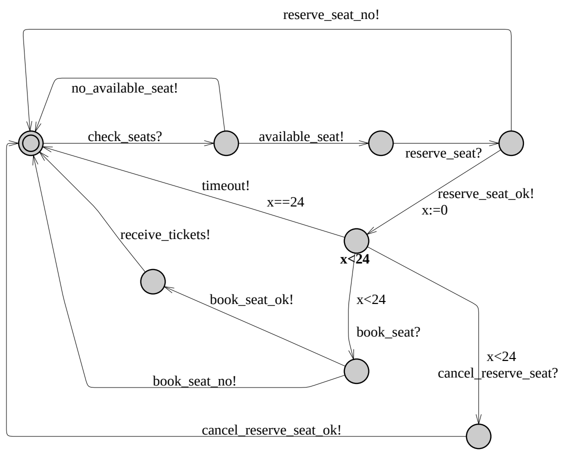

Fig. 8. Timed automaton for Airline Reservation System.

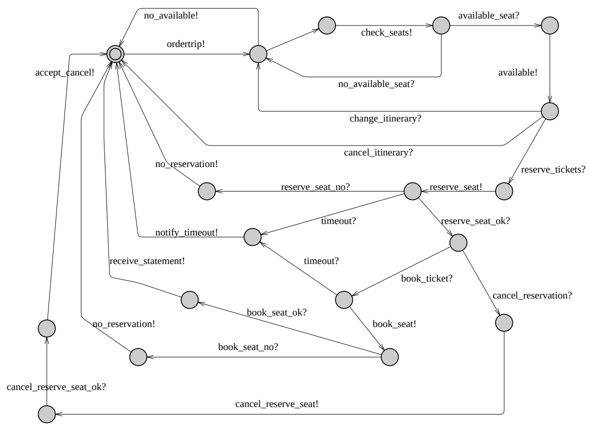

Fig. 9. Timed automaton for Travel Agent.

system evolves by executing transitions and delays in a random way. We have some other options in the Simulator. For instance, you can save simulations traces that can be later used to recover a specific execution trace. Actually, the simulation is quite flexible at this point, and you can back or forward in the sequence.

Then, our main goal in the validation phase of our case study is to check the correctness of the message flow and time-outs, taking into account the protocol definition. We have made a number of simulations, and we have concluded that the system design satisfies the expected behavior in terms of the message flow between the parties.

Before starting the automatic verification, we must establish which are the properties that the model must fulfill. We have divided these properties into three classes: Safety, Liveness and Deadlocks. These properties are specified by means of a *Temporal Logic*, and all of them have been checked by using the UPPAAL tool. The temporal Logic used by UPPAAL is described in [13].

Safety Properties: They allow us to check if our model satisfies some security restrictions. For example, if we have two trains that have to cross the same bridge, a security property is that both trains cannot cross at the same time the bridge:

$$\forall\Box\neg(Train1.crossing \wedge Train2.crossing) \text{ or}$$
$$\neg\exists\Diamond(Train1.crossing \wedge Train2.crossing)$$

The main Safety properties for our case study are the following:

- The TravelAgent always sends the itinerary on traveler's demand:

(1) $$\forall\Box Traveler.Itinerary \Rightarrow TravelAgent.sendItinerary$$

- The TravelAgent always changes the itinerary on traveler's demand:

(2) $$\forall\Box Traveler.ChangeItinerary \Rightarrow TravelAgent.PerformChange$$

- The TravelAgent always cancels the reservation on traveler's demand:

(3) $$\forall\Box Traveler.CancelReservation \rightarrow (TravelAgent.CancelReservtRcv \wedge Airline.PerformCancel \wedge Airline.Clockx < 24)$$

- A reservation is only available 24 hours before performing the booking:

(4) $$\forall\Box(TravelAgent.Booking \wedge Airline.ReceiveBoking \wedge Airline.ClockX <= 24)$$

- A Traveler always receives his tickets and the statement after the payment:

(5) $$\forall\Box Traveler.PaymentPerform \rightarrow (Traveler.Finish \wedge Airline.SnddTckt \wedge TravelAgent.SenddSttment)$$

Liveness Properties: They intend to check that our model can evolve in the right order. Returning to the train example, if a train approaches the bridge, some time later the train will be able to cross it:

$$Train.approach \rightarrow Train.crossed$$

Liveness Properties for our model are simple, for instance, if a Traveler sends a trip demand, some time later the TravelAgent will send the itineraries. Translating it into Temporal Logic we have:

(6) $$Traveler.PlanOrder \longrightarrow TravelAgent.SendItinerary$$

Another liveness property of interest is the following: if a Traveler orders a book within the next 24 hours after the reservation, the Airline performs the booking. Translating it into Temporal Logic we have:

(7) $$(Traveler.BookOdr \wedge Airline.ClockX < 24) \longrightarrow Airline.PerformBook$$

Deadlocks: These are clear restrictions. We could check if our model is deadlock free with the following formula:

(8) $$\forall\Box\neg Deadlock$$

5 Conclusions and Future Work

In this paper we have shown how we can apply formal methods to ensure the correctness of Web Services with time restrictions. We have shown that we can translate the descriptions written in WSCI-WSCDL into timed automata, and thus, we can use the UPPAAL tool to simulate and verify the system behavior.

In the particular case study we have used to illustrate how this methodology works (the airline ticket reservation system) this translation has been made manually, but our intention is to study if this translation can be made automatically, and in that case to implement a tool supporting this translation.

References

[1] R. Alur and D. Dill. *Automata for modeling real–time systems*. In Proceedings of the 17th International Colloquium on Automata, Languages and Programming, volume 443, Editors. Springer–Verlag, 1990.

[2] Assaf Arkin et al. *Web Service Choreography Interface (WSCI) 1.0*. In http:/www.w3.org/TR/wsci/.

[3] Assaf Arkin, Sid Askary, Ben Bloch, et. al., *Web Services Business Process Execution Language Version 2.0*, Editors. OASIS Open, December 2004. In http://www.oasis-open.org/committees/download.php/ 10347/wsbpel-specification-draft-120204.htm.

[4] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis and S. Yovine. *Kronos: A model-checking tool for real-time systems*. In Proc. 1998 Computer-Aided Verification, CAV'98, Vancouver, Canada, June 1998. Lecture Notes in Computer Science 1427, Springer-Verlag.

[5] Luc Clement, Andrew Hately, Claus von Riegen and Tony Rogers. *UDDI Version 3.0.2*, Editors. OASIS Open, 19 October 2004. In http://uddi.org/pubs/uddi_v3.htm.

[6] G. Diaz, F. Cuartero, V. Valero and F. Pelayo. *Automatic Verification of the TLS Handshake Protocol*. In proceedings of the 2004 ACM Symposium on Applied Computing.

[7] G. Diaz, K.G. Larsen, J. Pardo, F. Cuartero and V. Valero. *An approach to handle Real Time and Probabilistic behaviors in e-commerce: Validating the SET Protocol*. In proceedings of the 2005 ACM Symposium on Applied Computing.

[8] Eurostat yearbook 2004. *The statistical guide to Europe. Data 1992-2002*. European Commission: EUROSTAT, Office for Official Publications of the European Communities, 2004

[9] Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, et. al. *SOAP Version 1.2 Part 1: Messaging Framework*, Editors. World Wide Web Consortium, 24 June 2003. In http://www.w3.org/TR/soap12-part1.

[10] Constance Heitmeyer and Dino Mandrioli. *Formal Methods for Real-Time Computing*. John Wiley & Sons. 1996.

[11] Nickolas Kavantzas et al. *Web Service Choreography Description Language (WSCDL) 1.0*. In http://www.w3.org/TR/ws-cdl-10/.

[12] K. Larsen and P. Pettersson and Wang Yi. UPPAAL *in a Nutshell*. Int. Journal on Software Tools for Technology Transfer, Editors. Springer–Verlag vol.1, 1997.

[13] Kim G. Larsen, Paul Petterson, Wang Yi. *Diagnostic Model-Checking for Real-Time Systems*. Proc. of Workshop on Verification and Control of Hybrid Systems III, Lecture Notes in Computer Science, vol. 1066, pp. 575-586, 1995.

[14] Jean Paoli, Eve Maler, Tim Bray, et. al.*Extensible Markup Language (XML) 1.0 (Third Edition)*, Editors. World Wide Web Consortium, 04 February 2004. In http://www.w3.org/TR/2004/REC-xml-20040204.

[15] Sanjiva Weerawarana, Roberto Chinnici, Martin Gudgin, et. al. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, Editors. World Wide Web Consortium, 03 August 2004. In http://www.w3.org/TR/2004/WD-wsdl20.

[16] Simon Woodman, et al. *Specification and Verification of Composite Web Services*. In proocedings of The 8th Enterprise Distributed Object Computing Conference 2004.