Automatic generation of *Correct* Web Services Choreographies and Orchestrations with Model Checking Techniques

Gregorio Daz, M. Emilia Cambronero, Juan J. Pardo , Valentín Valero and Fernando Cuartero

***Abstract*—In previous works we have presented the generation of WS-CDL and WS-BPEL documents. In this paper we show the unification of both generations. The aim is to generate correct WS-BPEL skeleton documents from WS-CDL documents by using the Timed Automata as intermediary model in order to check the correctness of the generated Web Services with Model Checking Techniques. The model chequer used is UPPAAL, a well known tool in theoretical and industrial cases that perform the verification an validation of Timed Automata. Note that our interest is focused on Web services where the time constraints play a critical role.**

I. Introduction

Due to the development in new technologies such as telecommunications, nowadays our society is based upon the exchange of information. Clear examples are either the mobile phone nets that allows the mobile users to be available wherever they are or personal computer and PDAs that also increase the information exchange.

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 distributed application have appeared, such as the Extensible Markup Language, XML, and some new Web Services frameworks [7], [13], [19] 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 [16], [2]. 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.

Department of Computer Science, Univ. of Castilla La Mancha
EPSA, Avd. Espaa s/n, Albacete, Spain, 02071.
email: [gregorio,emicp,jpardo,valentin,fernando]@info-ab.uclm.es

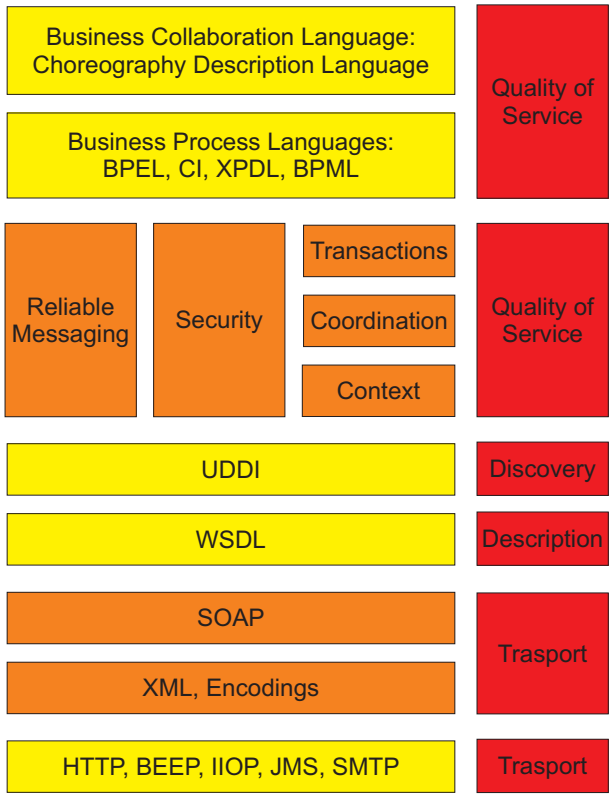

Fig. 1
The Stack Protocol for Web Services

The Web Service architecture stack targeted for integrating interacting applications consists of the components shown in Fig. 1.

The Web Services Choreography and Orchestration specifications are 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.

As in software design in Web Services design an important goal is the correctness. We must proof that our new designs are free of critical errors, because in practice mayor errors always occur during the design phase. And a error in this phase is also more expensive. If you discover it in the maintenance phase then you must start again from the design phase of the software life cycle style view point.

In Web Services the design phase is the most important

phase for Choreography and Orchestration layers. These layers describe the behaviors of each participant in a Web Service. In brief the Choreography describes the general behavior and the Orchestration describes each particular behavior. Thus an important goal is the validation and Verification of these layers in order to proof theirs correctness.

Then the verification process musts play an important role in Web Services development. However it is necessary to notice that due to the wide range of systems that Web Services are applied to, it is mandatory to check the different systems characteristics and a critical characteristic of numerous systems are time constraints. 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 [14]. 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 model checking [6], and there are some well-known tools supporting them, like UPPAAL [9], [10], [17], KHRONOS [3] and SPIN [15].

Web services choreography and orchestration layers are the highest layers of the web services framework and they have the proper level of abstraction that it is necessary due to the verification complexity. Thus, our goal with this work is to generate "correct" web services by applying formal techniques. In order to achieve it, we will describe the translation processes from choreography specifications written in *Web Services Choreography Description Language (WS-CDL)* [16] into Timed Automata UPPAAL XML format and from Timed Automata into orchestration specifications written in *web services business process execution language (WS-BPEL)* [2]. However, before the translation between Timed Automata and WS-BPEL, we must verify the time restrictions that the web service must fulfill.

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

The paper is structured as follows. In Section 2 we describe the main features of WS-CDL and WS-BPEL. The translation of WS-CDL documents into Timed Automata is presented in Section 3. In Section 4 we apply the first translation to the case study, and the UPPAAL tool is used to describe, simulate and analyze the obtained timed automata. The generation of WS-BPEL from timed automata is presented in section 5. Finally, the conclusions and the future work are presented in Section 6.

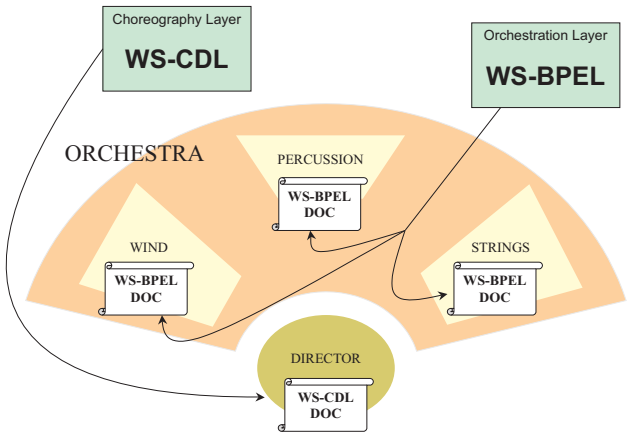

Fig. 2
WS-CDL and WS-BPEL relationship.

II. WS-CDL and WSBPEL Descriptions

Figure 2 illustrates the relationship between WS-CDL, the choreography layer and the orchestration level (WS-BPEL), taking an orchestra as a metaphor of this relation. The key document is the director score, which corresponds to the WS-CDL document, in which each participant is represented as well as the time it enters into action. Furthermore, the wind, percussion and strings scores correspond to the WS-BPEL documents, which show the behaviour of each particular group.

A. 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, 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 information 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 Vari-

able contains or the Token references.

- **Choreographies** define collaborations between interacting parties:
  - **Choreography Life-line** 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** specifies what additional interactions should occur when a Choreography behaves in an abnormal way.
  - **Choreography Finalizer Block** 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** realize a point of collaboration between parties by specifying where and how information is exchanged.
- **Work Units** 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** is the basic building block of a Choreography, 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** allow the creation of descriptions that can record the semantic definitions of every component in the model

In the figure 3 we can see a detailed piece of the WS-CDL document describing our study case. It describes part of the relationship between the Airline and the Travel Agent. The interaction determines the time that the reservation is available, one day.

```
<interaction
 name="reservation&booking" align="true"
 channelVariable="travelAgentAirlineChannel"
 operation="reservation&booking" 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>
 <timeout  time-to-complete="24:00" />
 <record  name="bookingTimeout"
  when="timeout" causeException="true" />
  <source variable=
    "AL:getVariable('tns:resOC', '', '')" />
  <target variable=
    "TA:getVariable('tns:resOC', '', '')" />
   </record>
</interaction>
```

Fig. 3
Part of WS-CDL especification

B. WS-BPEL Description

The Web Services Orchestration specification is aimed at being able to precisely describe the behavior of any type of party an the collaboration among them, regardless of the supporting platform or programming model used by the implementation of the hosting environment. Using the Web Services Orchestration specification, a behavior containing a "specific" definition of the detailed ordering conditions and constraints under which behavior is performed, is produced that describes the specific internal behavior with the exchanged messages with all the parties involved.

In real-world scenarios, corporate entities are often unwilling to delegate control of their business processes to their integration partners. Orchestration offers a means by which the rules of participation within a collaboration can be clearly defined and agreed to, jointly. Each entity may then implement its processes as determined by specified behavior in the Orchestration document.

Let us introduce the reader in more detail about WS-BPEL. It is an interface description language and describes the observable behaviour of a service by defining business processes consisting of stateful long-running interactions in which each interaction has a beginning, a defined behaviour and an end, all of this being modelled by a flow, which consists of a sequence of activities. The behaviour context of each activity is defined by a scope, which provides fault handlers, event handlers, compensation handlers, a set of data variables and correlation sets.

Let us now see a brief description of these components:

- **Events**, which describe the flow execution in an event driven manner.
- **Variables**, which are defined by using WSDL schemes, for internal or external purposes, and are used in the message flow.
- **Correlations**, which identify processes interacting by means of messages.
- **Fault handling**, defining the behaviour when an exception has been thrown.
- **Event handling**, defining the behaviour when an event occurs.
- **Activities**, which represent the basic unit of behaviour of a Web Service. In essence, WS-BPEL describes the behaviour of a Web Service in terms of choreographed activities.

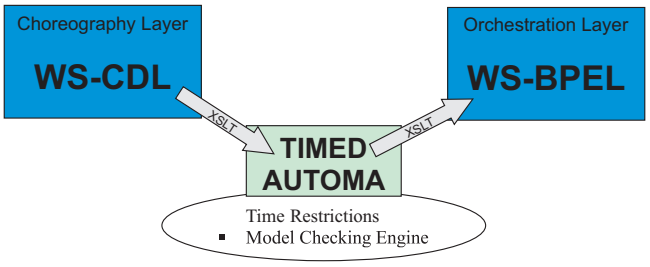

Fig. 4
From the Choreography layer to the Orchestration layer

III. Translation from WS-CDL to Timed Automata

Figure 4 depicts the translation processes that we present. In figure we can also see that WS-CDL documents are translated into timed automata in a first step and in a second step we will translate the timed automata into WS-BPEL documents. Therefore, we now present the automatic translation from WS-CDL documents into timed automata. For this purpose, we must first analyze the WS-CDL documents in order to identify the common shared points between them. The first stage is to obtain the general structure describing the system that we are analyzing. In timed automata, this structure is defined by the so-called *System*, which consists of the individual processes that must be executed in parallel. Each one of these processes is defined by using a template. Templates are used to describe the different behaviors that are available in the system.

Then, for each component of a WS-CDL description we have the following correspondence in timed automata (see Fig. 5 for a schematic presentation of this correspondence):

Role : They are used to describe the behaviour of each class of party that we are using in the choreography. Thus, this definition matches with the definition of a *template* in timed automata terminology.

Relation type : They are used to define the communications between two roles, and the needed channels for these communications. In timed automata we just need to assign a new channel for each one of these channels, which are the parameters of the templates that take part in the communication.

Participant type : They define the different parties that participate in the choreography. In timed automata they are processes participating in the system.

Channel types : A channel is a point of collaboration between parties, together with the specification of how the information is exchanged. As said before, channels of WS-CDL correspond with channels of timed automata.

Variables : They are easily translated, as timed automata in UPPAAL support variables, which are used to represent some information.

Now the problem is to define the behaviour of each template. This behaviour is defined by using the information provided by the flow of choreographies. Choreographies are sets of workunits or sets of activities. Thus, activities and workunits are the basic components of the choreographies, and they capture the behavior of each component. Activities can be obtained as result of a composition of other activities, by using sequential composition, parallelism and choice. In terms of timed automata these operators can be easily translated:

- The sequential composition of activities is translated by concatenating the corresponding timed automata.
- Parallel activities are translated by the cartesian product of the corresponding timed automata.
- Choices are translated by adding a node into the automata which is connected with the initial nodes of the alternatives.

Finally, time restrictions are associated in WS-CDL with workunits and interaction activities. These time restrictions are introduced in timed automata by means of guards and invariants. Therefore, in case a workunit of an activity has a time restriction we associate a guard to the edge that correspond to the initial point of this workunit in the corresponding timed automaton.

IV. The verification of the Case Study

Some examples of the use of WS-CDL and WS-BPEL can be found in [4], [5], [11]. The case study that we are going to use to illustrate how the translation works is inspired from the work [11], where this particular case study was used to illustrate how timed automata can be used for the formal verification of properties for WS-CDL documents.

This system consists of three participants: a Traveller, a Travel Agent and an Airline Reservation System, whose behaviour is as follows:

A Traveller 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

Role = Template
Relation Type = Channel$^+$
Participant Type = Process$^+$
Channel Type = Channel
Variables = Variables
Choreography = Choreography$^+$ | Activity
Activity = Work Unit | Sequence | Paralelism | Choice
Sequence = Activity$^+$
Paralelism = Activity$^+$
Choice = Activity$^+$
Work Unit = State & Guard & Invariant

where the symbols +, | are BNF notation,
em and & *is used to join information*

Fig. 5
Schematic view of the translation

(*Order Trip*). The Travel Agent selects the best itinerary according to the criteria established by the Traveller. 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 Traveller 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 Traveller 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 Traveller that the itinerary is not possible (*Notify of Cancellation*).

Once made the reservation the Travel Agent informs the Traveller (*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 Traveller 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. 6.

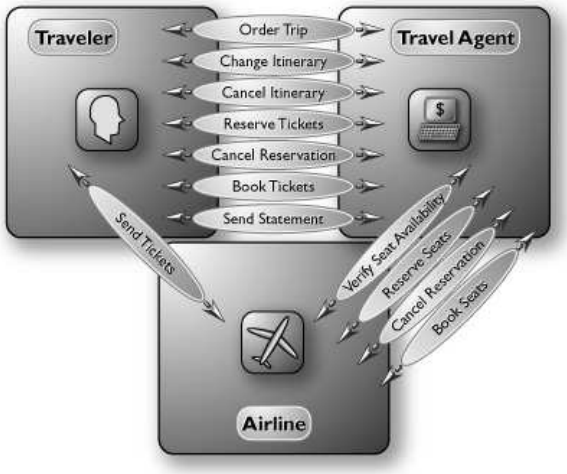

Fig. 6
Flow of the messages exchanged.

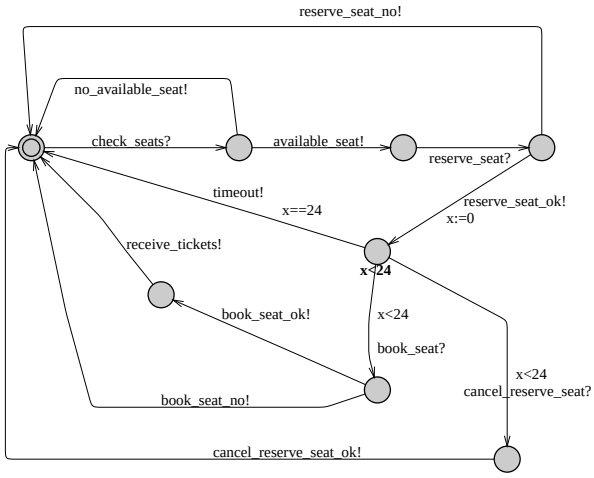

Fig. 7
Timed automata for airline Reservation System.

A. Translation of the Case Study

Figure 3 presents a detailed piece of the WS-CDL document describing our example that we have used to obtain the translation into timed automata. Following the guidelines described above we have obtained in this case three timed automata: the traveler, the travel agent and the airline company. These automata are shown in Figures 7, 8 and 9.

Notice the use of the clock x in the timed automaton corresponding to the airline reservation system, which is used to control when the reservation expires. This clock is initialized when the action *reserved_seat* is done.

B. Simulation and Verification

In the simulation we can check whether the model holds the system behavior or not. This can partially be made by means of simulations. These 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. Thus, you can choose the transition that you want to execute. Nevertheless, you can also select the random execution of transitions, and thus, the system evolves by executing transitions and delays which are selected randomly. We have some other options in the Simulator. For example, you can save simulations traces that can later be used to recover an specific execution trace. Actually, the simulation is quite flexible at this point, and you can back or forward in the sequence.

Then, with respect to our study case, our main goal in the validation phase 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 ex-

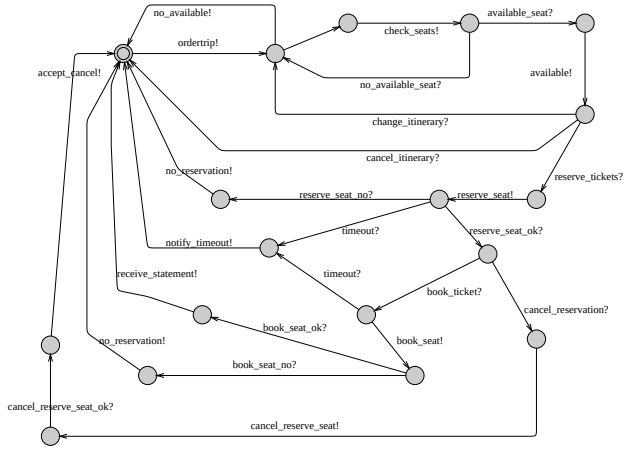

Fig. 8
Timed automata for Travel agent web service.

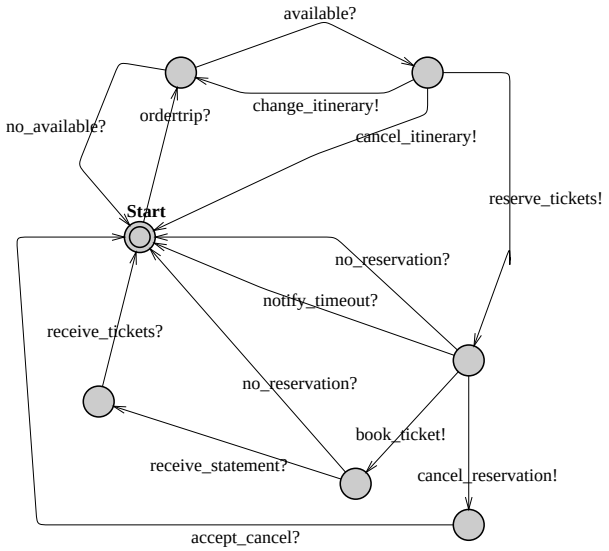

Fig. 9
Timed automata for traveler.

pected 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*. The temporal Logic used by UPPAAL is described in [17].

Safety Properties 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 can not cross at the same time the bridge: $\forall\Box\neg(Train1.crossing \wedge Train2.crossing)$ or $\neg\exists\Diamond(Train1.crossing \wedge Train2.crossing)$

The main Safety properties are:

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

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

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

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

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

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

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

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

- A Traveler always receive theirs ticket and statement after perform the payment:

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

Liveness Properties intend is 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 could cross it. $Train.approach \rightarrow Train.crossed$

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

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

Or for example, if a Traveler makes a book within 24 hours after the reservation, the Airline perform the booking. Translating it into Temporal Logic we have:

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

Deadlocks are clear restrictions. We could check if our model is deadlocks free:

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

V. Translation from Timed Automata into WS-BPEL

Then, for each component of timed automata we have the following correspondence in WS-BPEL specification:

Templates and Processes : They are used to describe the behaviour of each class of party that we are using in the choreography. Thus, this definition matches with the definition of a *process* in WS-BPEL terminology.

Synchronization : They are used to define the communications between two automata template, and the needed constraints for these communications. In WS-BPEL we just need to assign a correlation set for each one of these synchronizations.

Channel : The channel is a point of collaboration between parties, together with the specification of how the information is exchanged. Channels of Timed Automata correspond with ports of WS-BPEL.

Variables : They are easily translated, as WS-BPEL support variables, which are used to represent some information.

Now the problem is to define the behavior of each template. This behavior is defined by using the information provided by the flow and synchronization between the different automata. Timed Automata are sets of processes. Thus, processes are the basic components of the Timed Automata, and they capture the behavior of each component. Process can be obtained as a result of a composition of process activities, by using sequential, choice and parallel operators. In terms of timed automata these operators can be easily translated:

- The sequence of transitions is translated by composing a sequence of activities.
- Parallel proceses are translated by the parallel operator with the involved processes.
- Choices are translated by adding activities to a choice operator.

Finally, time restrictions are associated in WS-BPEL with activities and correlations. These time restrictions are introduced in timed automata by means of guards and invariants. Therefore, in case a guard in a loop transition has a time restriction we generate a workunit that involves the generated activities from the target of the loop transition to the beginning of it.

Figure 10 presents a detailed piece of the WS-BPEL document describing our example that is the mirror image of figure 3that have been obtained by applying the translations.

```
...
<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/AcceptCancel">
      <correlate
          correlation="defs:reservationCorrel"/>
   </action>
   <action name="NotifyOfTimeOut"
       role="tns:TravelAgent"
       operation=
           "tns:TAtoTraveller/NotifyofCancel"/>
           <fault code="tns:reservatTimedOut"/>
  </onMessage>
  ...
  </exception>
  ...
  </context>
```

Fig. 10
Part of WS-BPEL especification

VI. Conclusions and Future Work

Nowadays Web Services are becoming a powerful tool for the implementation of distributed applications over Internet. In many cases these services have associated time restrictions, as we have seen in the case study that we have presented. Therefore, the specification and design of Web Services can be made by using some well known formalisms, as timed automata, and tools supporting them (UPPAAL) in order to verify and validate the system behavior. Consequently, it becomes of interest to obtain *correct* web services specifications written in WS-CDL and WS-BPEL languages by using Timed Automata and Model Checking in order to exploit the verification capabilities that they can provide us. Thus, in this paper we have seen how exploit these capabilities by using translations between them, and they have been applied to a particular case study. We are currently implementing these translations in a suite tool that uses UPPAAL as the engine for the simulation and verification.

Our future work is addressed to implement a complete methodology on the generation of *correct* web services. We will use UML graphical diagrams and requirements engineering in the design and analysis of real-time web services respectively, WS-CDL and WS-BPEL as implementation languages, and Timed Automata and Model Checking as formal techniques.

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, 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.

[3] 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.

[4] Mario Bravetti, Roberto Lucchi, Gianluigi Zavattaro and Roberto Gorrieri , *Web Services for E-commerce: guaranteeing security access and quality of service*, In Proc. of the 19th ACM Symposium on Applied Computing (SAC'04), special track on E-Commerce Technologies , ACM Press, 2004.

[5] Mario Bravetti, Claudio Guidi, Roberto Lucchi and Gianluigi Zavattaro , *Supporting E-commerce system formalization with Choreography Languages*, In Proc. of the 20th ACM Symposium on Applied Computing (SAC'05), special track on E-Commerce Technologies , ACM Press, 2005.

[6] Edmund M. Clarke and Jr. and Orna Grumberg and Doron A. Peled, *Model Checking*, MIT Press, 1999.

[7] 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.

[8] Francisco Curbera et al. *Business Process Execution Language for Web Services, Version 1.0*. In http://xml.coverpages.org/WS-BPELv10.pdf.

[9] 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.

[10] 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.

[11] G. Diaz, J. J. Pardo, M. E. Cambronero, V. Valero and F. Cuartero, *Verification of Web Services with Timed Autoamata*, In proceedings of First International Workshop on Automated Specification and Verification of Web Sites, Valencia, March 2005.

[12] G. Diaz, J. J. Pardo, M. E. Cambronero, V. Valero and F. Cuartero, *Automatic Translation of WS-CDL Choreographies to Timed Automata*, In proceedings of WS-FM, 2005.

[13] 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.

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

[15] Gerard J. Holzmann . *The SPIN Model Checker: primer and reference manual*. Pearson Education Inc., 2003.

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

[17] 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.

[18] 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.

[19] 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/2002/ws/desc/.

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