Model Checking Techniques applied to the design of Web Services

Gregorio Díaz, M. Emilia Cambronero, Juan J. Pardo ,
Valentín Valero and Fernando Cuartero
Department of Computer Science, Univ. of Castilla La Mancha
EPSA, Avd. España s/n, Albacete, Spain, 02071.
email: [gregorio,emicp,jpardo,valentin,fernando]@info-ab.uclm.es

May 10, 2007

Abstract

In previous work 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 an intermediary model in order to check the correctness of the generated Web Services with Model Checking Techniques. The model checker used is UPPAAL, a well known tool in theoretical and industrial cases that performs the verification and validation of Timed Automata. Note that our interest is focused on Web services where the time constraints play a critical role.

1 Introduction

Web Service technology is a framework, in which the study of design errors can become a master piece, due to the big amount of money and other resources spent on it. Recent studies have been realized about the acceleration of the increasing of the percent of gross domestic product that e-commerce represents for the world economy. Thus, possible errors on the technologies supporting this framework could cause a big cost, not just economically, but also sociologically. For instance, you can think of an e-stock market, where, due to a design error, the system always forces all transactions to use the most expensive seller. This situation can avoid the money saving for users of this platform. Then, it is economically feasible the use of algorithms, methodologies and techniques in order to allow system designers to determine whether designs are free of errors or not.

In Web Services the design phase is the most important phase for Choreography and Orchestration layers, figure 1. 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 prove their correctness.

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 [7, 10]. Furthermore, these formalisms allow us to reason with the

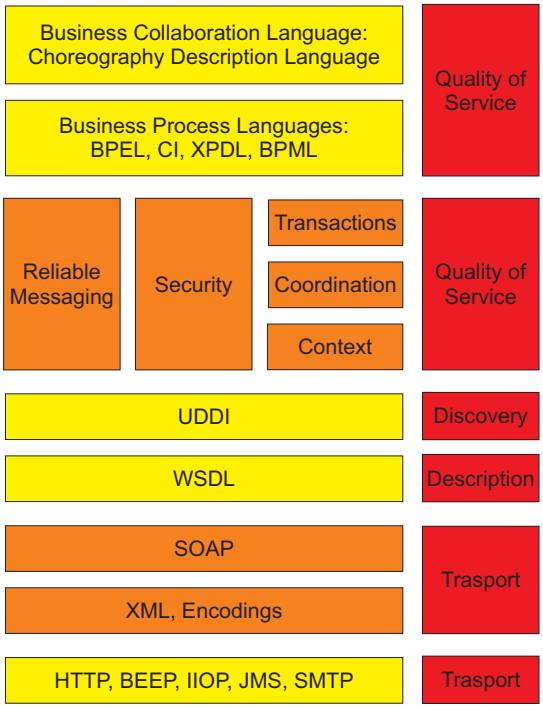

Figure 1: The Stack Protocol for WS.

constructed models, analysing and verifying some properties of interest of the described systems. One of these formalisms are timed automata [1], which are very widely used in model checking [4], and there are some well-known tools supporting them, like UPPAAL [9].

Web services choreography and orchestration layers are the highest layers of the web services framework and they have the proper level of abstraction that 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)* [8] 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.

To illustrate 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 timed automata obtained. The generation of WS-BPEL from timed automata is presented in section 5. Finally, the conclusions and future work are presented in Section 6.

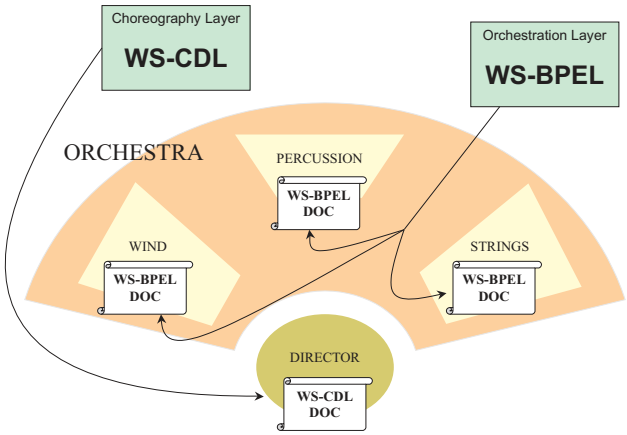

Figure 2: WS-CDL and WS-BPEL relationship.

2 WS-CDL and WS-BPEL 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.

2.1 WS-CDL Description

WS-CDL describes interoperable collaborations between parties. In order to facilitate these collaborations, services commit themselves 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. 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** 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 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, namely, one day.

2.2 WS-BPEL Description

The Web Services Orchestration specification is aimed at being able to precisely describe the behavior of any type of party and the collaboration among them, regardless of the supporting platform or programming model used by the implementation of the hosting environment.

WS-BPEL is an interface description language and describes the observable behavior of a service by defining business processes consisting of stateful long-running interactions in which each interaction has a beginning, a defined behavior and an end, all of this being modelled by a flow, which consists of a sequence of activities. The behavior 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**, describe the flow execution in an event driven manner.
- **Variables**, are defined by using WSDL schemes, for internal or external purposes, and are used in the message flow.
- **Correlations**, identify processes interacting by means of messages.

```
<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>
```

Figure 3: Part of WS-CDL especification

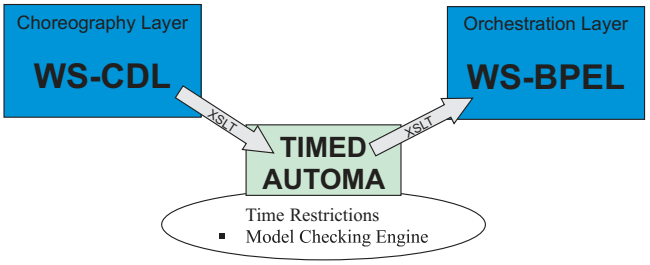

Figure 4: From WS-CDL to WS-BPEL.

- **Fault handling**, defines the behavior when an exception has been thrown.
- **Event handling**, defines the behavior when an event occurs.
- **Activities**, represent the basic unit of behavior of a Web Service. In essence, WS-BPEL describes the behavior of a Web Service in terms of choreographed activities.

3 Translation from WS-CDL to Timed Automata

Figure 4 depicts the translation processes that we present. In this figure, we can also see that WS-CDL documents are translated into timed automata in a first step and in a second step the timed automata latter are translated into WS-BPEL documents. We now present the automatic translation from WS-CDL documents into timed automata that we have presented in [6]. For this purpose, we must first analyze the WS-CDL documents in order to identify the common points shared 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):

Roles : These are used to describe the behavior of each type of party that we are using in the choreography. Thus, this definition matches with definition of a *template* in timed automata terminology.

Relation types : These are used to define the communications between two roles, and the channels needed 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 types : These 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 mentioned above, 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 certain information.

Now the problem is to define the behavior of each template. This behavior 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 the case of a workunit of an activity having a time restriction we associate a guard to the edge that corresponds to the initial point of this workunit in the corresponding timed automaton.

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*

Figure 5: Schematic view of the translation.

4 The verification of the Case Study

Some examples of the use of WS-CDL can be found in [3, 5]. The case study that we are going to use to illustrate how the translation works is closely linked to [5], 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 behavior 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 (*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 the case where he rejects the proposed itinerary, he may submit the modifications (*Change Itinerary*), and wait for a new proposal from the Travel Agent.
- In the case where he decides not to take the trip, he informs the Travel Agent (*Cancel Itinerary*) and the process ends.
- In the case where 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

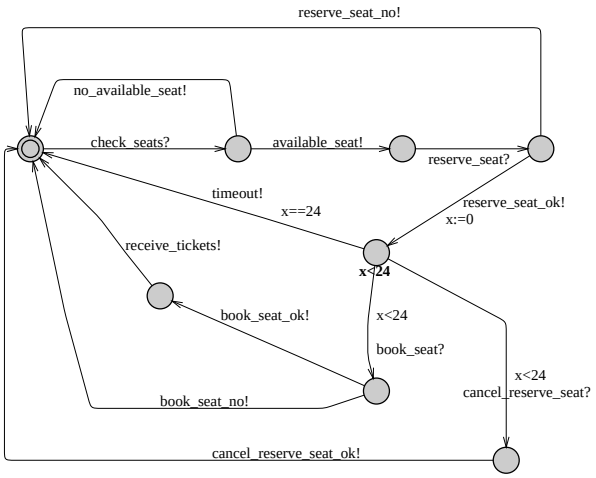

Figure 6: Timed automata for airline Reservation System.

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 the reservation is made the Travel Agent informs the Traveller (*Seats Reserved*). However, this reservation is only valid for a period of 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*).

4.1 Translation of the Case Study

Figure 3 presents a detailed piece of the WS-CDL document describing the 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 6, 7 and 8. 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 carried out.

4.2 Simulation and Verification

In the simulation we can check whether the model keeps the system behavior or not. This can partially be done 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 simulation traces that can later be used to recover an specific execution trace. Actually, the simulation is quite flexible at this point, and you can move back or forward in the sequence.

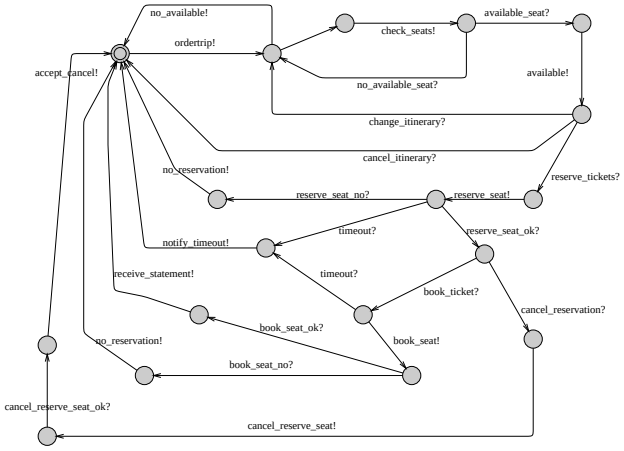

Figure 7: Timed automata for Travel agent web service.

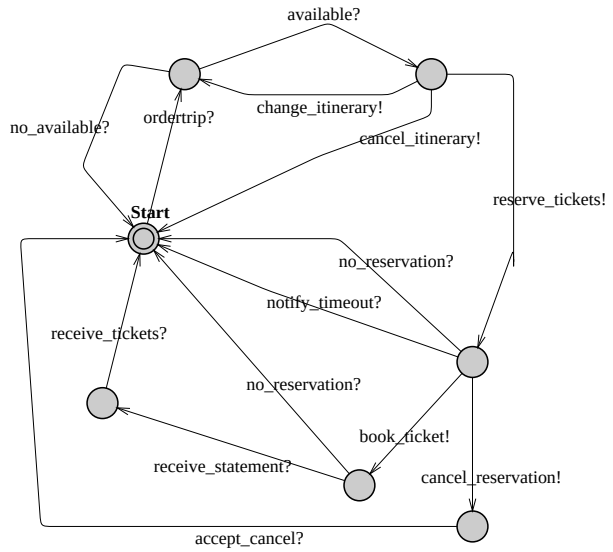

Figure 8: Timed automata for traveler.

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. Having made a number of simulations, 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 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 [9].

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

The main Safety properties are:

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

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

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

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

- The TravelAgent always cancel the reservation on the 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 performing 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 are used 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:

$$Traveler.PlanOrder \longrightarrow TravelAgent.SendItinerary \tag{6}$$

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

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

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

$$\forall\Box\neg Deadlock \tag{8}$$

5 Translation from Timed Automata into WS-BPEL

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

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

Synchronization : These are used to define the communications between two automata templates, and the constraints needed 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 : These are easily translated, as WS-BPEL supports variables, which are used to represent certain 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 the case of a guard in a loop transition having a time restriction we generate a workunit that involves the activities generated from the target of the loop transition to the beginning.

Figure 9 presents a detailed piece of the WS-BPEL document describing our example this is the mirror image of Figure 3 that has 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>
```

Figure 9: Part of WS-BPEL especification

6 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 here. Therefore, the specification and design of Web Services can be made by using some well known formalisms, such 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. In this paper we have seen how to exploit these capabilities by using translations between them, applied to a particular case study. We are currently implementing these translations in a suite tool that uses UPPAAL as the engine for simulation and verification.

Our future work addresses the issue of implementing 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] 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.

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

[5] 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.

[6] 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.

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

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

[9] 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.

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