Automatic Translation of WS-CDL Choreographies to Timed Automata [⋆]

Gregorio Diaz, Juan-José Pardo, María-Emilia Cambronero,
Valentín Valero, and Fernando Cuartero

Departamento de Informática
Universidad de Castilla-La Mancha
Escuela Politécnica Superior de Albacete. 02071 - SPAIN
[gregorio,jpardo,emicp,valentin,fernando]@info-ab.uclm.es

Abstract. In this paper we show how we can translate Web Services described by WS-CDL into a timed automata orchestration, and more specifically we are interested in Web services with time restrictions. Our starting point are Web Services descriptions written in WSBPEL - WSCDL (XML-based description languages). These descriptions are then automatically translated into timed automata, and then, we use a well known tool that supports this formalism (UPPAAL) to simulate and analyse the system behaviour. As illustration we take a particular case study, an airline ticket reservation system.

1 Introduction

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 [7,13,18] 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 [15]. 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 [15]:

[⋆] 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"

- **SOAP[13]:** It defines the basic formatting of a message and the basic delivery options independent of programming language, operating system, or platform.
- **WSDL[18]:** 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.
- **Registry[7]:** It makes visible an available Web Service, and it also describes the concrete capabilities of a Web Service.
- **Security layer:** It ensures 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.
- **Context, Coordination and Transaction layer:** It defines interoperable 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[2,8]:** 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[15]:** 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 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 [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 [9,10,16] and KHRONOS [3].

Then, our goal with this paper is to describe how we can translate Web Services with time constraints into a formalism using automatic techniques in order to verify it. 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 Business Process Execution Language (WS-BPEL) [2], and the concrete Choreography Layer that we use is the Web Service Choreography Description Language (WS-CDL) [15]. Therefore, the starting point are specification documents written in WS-CDL and WS-BPEL. However, these description languages are not very useful for the verification process. Thus, these descriptions are automatically translated into timed automata, and the UPPAAL tool is used to simulate and verify the system correctness.

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 WSBPEL - WSCDL. The translation of WSCDL documents into timed automata is presented in Section 3. In Section 4 we apply this methodology to the case study, and the UPPAAL tool is used to describe, simulate and analyze the obtained timed automata. Finally, the conclusions and the future work are presented in Section 5.

2 WSBPEL - 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 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 1 demonstrates a possible usage of the Choreography Description Language, where we see that we use WS-BPEL as the Business Process Execution Layer (BPEL for short).

WS-CDL describes interoperable, collaborations between parties. In order to facilitate these collaborations, services commit to mutual responsibilities by

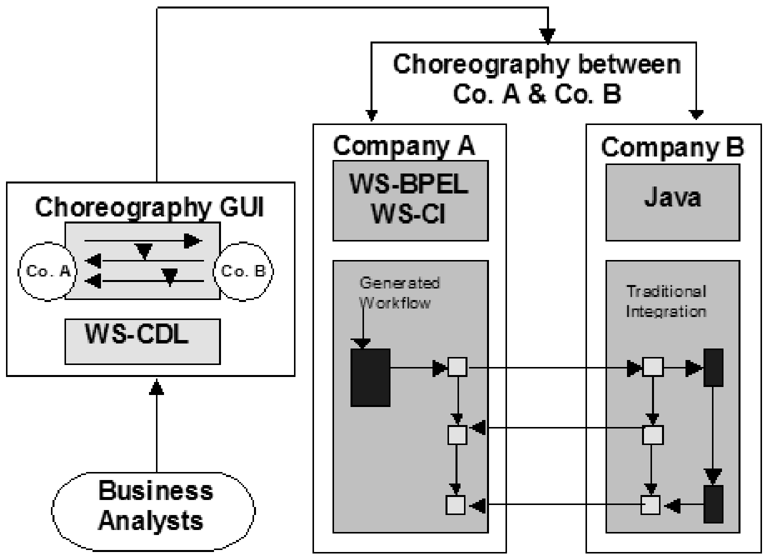

Fig. 1. WS-CDL and WS-BPEL usage.

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 Variable contains or the Token references.
- **Choreographies** define collaborations between interacting parties:
  - **Choreography Life-line**: It shows the progression of a collaboration. Initially, the collaboration is established between the parties; then, some work is performed within it, and finally it completes either normally or abnormally.

    - **Choreography Exception Block**: It specifies the additional interactions that should occur when a Choreography behaves in an abnormal way.
    - **Choreography Finalizer Block**: It 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** establish 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 synchronizations of their observable information changes, and the actual values of the exchanged information.

2.1 WS-BPEL

WS-BPEL is an interface description language. It 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.

3 Translation

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.

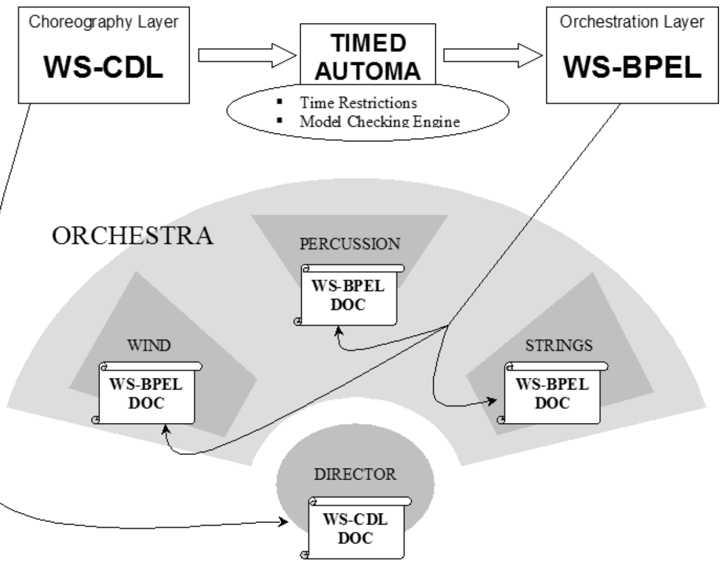

Fig. 2. From the Choreography layer to the Orchestration layer

From this Figure we can also see that WS-CDL documents are translated into timed automata in a first step, which is the main goal covered with this paper, and in a second step we intend to translate the timed automata thus obtained into WS-BPEL documents. Therefore, we now present the automatic translation from WS-CDL documents into timed automata. For this purpose, we must first analyse 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. 3 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.

4 Case Study: Travel Reservation System

Some examples of the use of WS-CDL 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.

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

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, and & is used to join information

Fig. 3. Schematic view of the translation

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 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. 4.

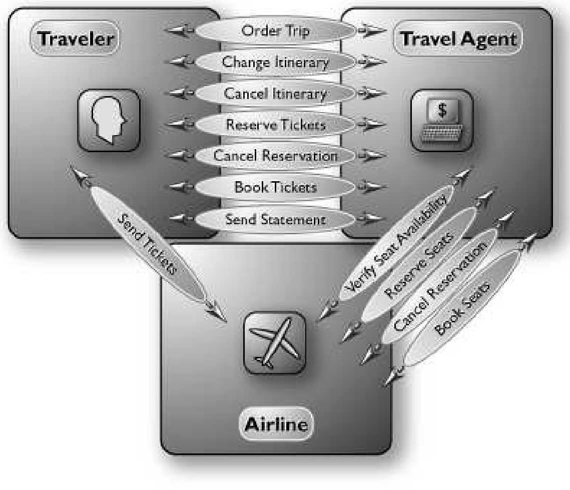

Fig. 4. Flow of the messages exchanged.

4.1 Translation of the Case Study

Figure 5 presents a detailed piece of the WS-CDL document describing our example. It describes part of the relationship between the Airline and the Travel Agent. This interaction establishes the time in which the reservation is available, in this case one day.

We have used this WSCDL document 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 done.

```
<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. 5. Part of WS-CDL especification

5 Conclusions and Future Work

Nowdays 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 a translation of the specifications written in a Choreography language (WS-CDL) into timed automata in order to exploit these capabilities that timed automata can provide us. Thus, in this paper we have seen how this translation can be made, and it has been applied to a particular case study. We are currently implementing this translation in a tool that uses UPPAAL as the engine for the simulation and verification.

Our future work will focus on the second step of this methodology of traduction, in which our intention is the generation of WS-BPEL documents from WS-CDL documents, using as intermediary objects the timed automata obtained with the translation presented in this paper. Notice that these timed automata will have some internal information, which will not be used by UPPAAL, but that will be necessary in order to obtain the corresponding WS-BPEL documents.

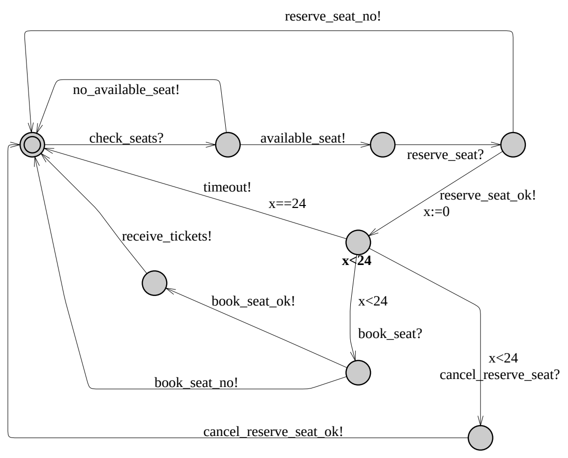

Fig. 6. Timed automata for airline Reservation System.

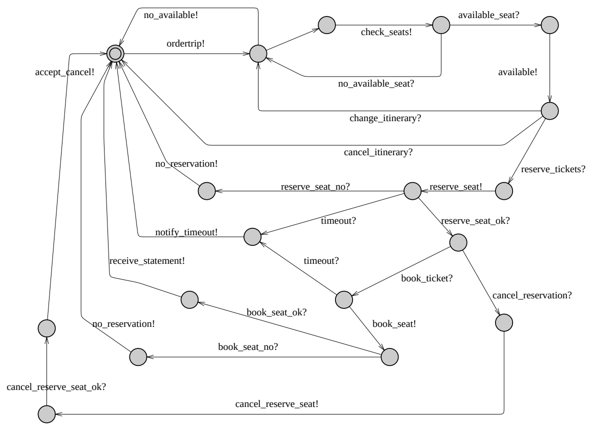

Fig. 7. Timed automata for Travel agent web service.

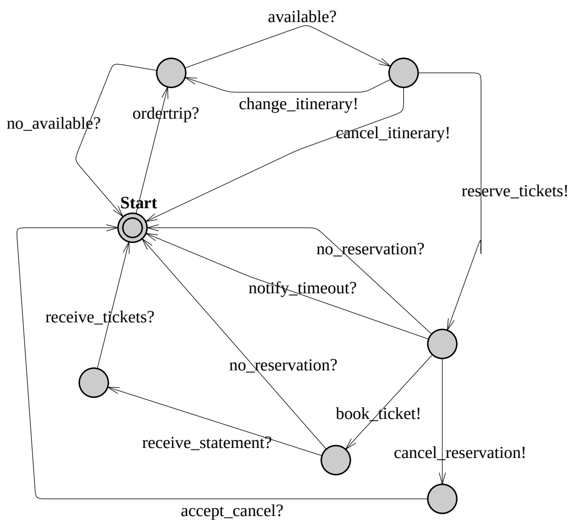

Fig. 8. Timed automata for traveler.

Once this second step has been completed we will have a complete methodology for obtaining correct orchestration descriptions of Web Services from choreography descriptions.

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. Eurostat yearbook 2004. *The statistical guide to Europe. Data 1992-2002.* European Commission: EUROSTAT, Office for Official Publications of the European Communities, 2004
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. Nickolas Kavantzas et al. *Web Service Choreography Description Language (WSCDL) 1.0.* In http://www.w3.org/TR/ws-cdl-10/.
16. 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.
17. 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.
18. 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/.
19. Simon Woodman, et al., *Specification and Verification of Composite Web Services*, In proocedings of The 8th Enterprise Distributed Object Computing Conference 2004.