Journal of Universal Computer Science, vol. 17, no. 13 (2011), 1800-1829
submitted: 29/4/09, accepted: 30/3/11, appeared: 1/9/11 © J.UCS

Design and Generation of Web Services Choreographies with Time Constraints[1]

M. Emilia Cambronero
(Dept. of Computer Science. Campus Universitario. ESII. Albacete. Spain
emicp@dsi.uclm.es)

Valentín Valero
(Dept. of Computer Science. Campus Universitario. ESII. Albacete. Spain
valentin@dsi.uclm.es)

Enrique Martínez
(Dept. of Computer Science. Campus Universitario. I3A. Albacete. Spain
emartinez@dsi.uclm.es)

Abstract: In this paper we show how UML 2.0 sequence diagrams can be used for the design of Web service choreographies with time constraints and how these sequence diagrams can be extended with frames for the description of Web service choreographies. We then show how the diagrams can be translated into WS-CDL documents. This translation is of interest, since non-XML experts can find it difficult to implement a composite web service by WS-CDL, i.e. XML code. Graphic models, such as UML sequence diagrams, are a popular and well-studied framework for a compact representation of interoperation among participants in a distributed system and can be used as a starting document for the design of a composite Web service, from which the corresponding WS-CDL document can be derived.

Key Words: UML, modeling, design, code generation, web services, WS-CDL, Real-Time systems.
Category: D.2.2, D.2.3, D.2.10, H.4.3.

1 Introduction

There has recently been a surge of interest in Web services as more and more intra/inter-organizational applications use this model. A Web service is an autonomous, standards-based component whose public interfaces are defined and described using XML [Kavantzas et al., 2005; Weerawarana et al., 2004]. Other systems may interact with a Web service in a manner prescribed by its definition, using XML-based messages conveyed by Internet protocols.

Internet and Web technologies are thus a new way of doing business more cheaply and efficiently, as enterprises can provide new and dynamic services

[1] Supported by the Spanish government (Secretaría de Estado de Universidades), and cofinanced by FEDER founds, with the project TIN2009-14312-C02-02, and the JCCLM regional project PEII09-0232-7745.

faster by means of Web Services. However,B2B e-commerce is still in the formative stage, and new software technologies are required to support their development. There is a specific need for an effective and efficient means to abstract, compose, analyze and evolve Web Services in an appropriate time-frame [Hamadi and Benatallah, 2003].

Current web services technology is based on the Web Service architecture stack proposed by the World Wide Web Consortium, W3C [W3C, 2011], which consists of the following components: SOAP, WSDL, Registry (UDDI), Security layer, Reliable Messaging layer, Context, Coordination and Transaction layer, Business Process Languages layer (WSBPEL) and Choreography layer. The three basic layers are the *SOAP*, *WSDL* and *UDDI*. The SOAP layer describes the message format and delivery options, the WSDL language describes the static interface of a Web Service, whereas the UDDI layer makes a Web Service visible and available. The intermediate layers, security, reliable messaging, context, coordination and transaction layers provide a wide range of quality properties for the communications process. Finally, the highest and most abstract layers are the Business Process Languages and the Choreography layers. The Business Process Languages layer describes the execution logic by defining its control flow and prescribing the rules for managing its non-observable data and is also known as the *Orchestration* layer. The *Choreography* layer describes the collaboration of parties by defining a global view of their common and complementary observable behavior, where information exchanges occur and when the jointly agreed ordering rules are satisfied. One of the most widely used W3C pre-standardized protocols for this layer is the Choreography Description Language (WS-CDL) [Kavantzas et al., 2005].

The choreography layer thus provides a global description of the communications that take place in the system by specifying the interactions among participants, whereas orchestration (WSBPEL) is used for the modeling and implementation of individual executable processes, describing their behavior and their interactions from a local viewpoint.

The composition of Web Services entails the integration of the requirements of each component. These requirements include the format of the messages exchanged among the parties, the channels used for communications, type of communications (request and/or response), control flow, exception handling, but timed aspects can also be considered. All of these requirements can be covered by the choreography layer.

In this paper we focus on the Choreography layer, with the goal of deriving automatic WS-CDL specifications, starting from specifications written in UML 2.0 sequence diagrams. Some interactions may have associated time restrictions, in the sense that they can only be performed within a specified time window, so another important goal is to include these time restrictions in the model.

Our final goal is the development of a methodology for the generation and verification of "correct" Web services with time constraints. Correct Web services are defined as those Web services that fulfill the established requirements, which must be defined in the analysis phase as a set of properties of interest that must be verified. Consequently, a verification phase is also needed, which is obtained by using model checking techniques.

In previous work [Cambronero et al., 2010; Díaz et al., 2006] we introduced the fundamentals of this methodology, which is based on an automatic translation of Web services descriptions with time restrictions written in WS-CDL into timed automata (supported by the WST tool [Cambronero et al., 2011]). The timed automata thus obtained can then be used to simulate the systems as well as to verify some of its properties by means of UPPAAL [Larsen et al., 1997].

The work presented in this paper corresponds to the design phase. We use Unified Modeling Language UML 2.0 [OMG, 2003], which is a well known standard for the specification of software systems. This has been extended to include time aspects in the specifications, by defining the appropriate profiles [Cambronero et al., 2006; Graf et al., 2006; OMG, 2002]. Our interest focuses specifically on the sequence diagram since it is useful to represent the main features of WS Choreographies as well as the timed constraints of the interactions. A sequence diagram depicts the sequence of actions that occur in a system. The invocation of the methods of each object, and the order in which the invocations occur are captured by a sequence diagram, whis is able represent the dynamic behavior of a system in a simple way. With these sequence diagrams we can describe how the different participants in a composite Web service operate with one another and in what order, as they represent the interactions that take place in the system arranged in a time sequence.

Timed restrictions are represented in sequence diagrams as constraints placed between two or more exchanged messages. A key element in a UML 2.0 sequence diagram for dealing with time and nesting scenarios are frames [Ambler, 2005], which extend the classical UML 1.x sequence diagrams by adding some new capabilities. One of these capabilities is the possibility of labeling the elements of the sequence diagram, another is nesting, i.e. we can use nested frames.

The paper is structured as follows. A discussion of related work is given in Section 2. In Section 3 we describe the main features of WS-CDL. The translation of UML diagrams into WS-CDL documents is then presented in Section 4. Implementation is described in Section 5. Section 6 gives a brief description of the verification phase. In Section 7 we apply this methodology to a particular case study, using UPPAAL to check some important properties. Finally, our conclusions and future lines of research are presented in Section 8.

2 Related work

Web services are now attracting a lot of attention and several studies have proposed various approaches and frameworks for their specification and analysis. Alexander Lorenz et al. [Lorenz and Six, 2006] use UML activity models for the specification of use cases in the context of interactive systems. These activities must be adapted and refined before they can successfully be applied to the context of interactive systems. They tailor the activities to these needs, thus obtaining the so-called *interaction-oriented activities*. They introduce a distinction between "user actions" and "system actions" by defining two stereotypes which define the tasks performed by the user and those performed by the system. They use UML activity diagrams to model the system instead of the sequence diagrams we use, which show how objects interact dynamically with actors and other objects.

Some studies focus on dependability analysis. [Zarras et al., 2004b] presents a methodology for performing dependability analysis of composite web services, which uses UML representation for the architecture specification of composite web services written in WSBPEL. The UML representation is then extended by adding properties which characterize the failure behavior of the elements that constitute the composite web services, such as availability, reliability and safety. This extended UML model is finally mapped in Block Diagrams, Fault Trees and Markov models, where dependability analysis techniques can be applied.

Zhang et al. [Zhang et al., 2008] propose the use of different UML diagrams to model WS-CDL. WS-CDL documents are firstly modeled by using UML specifications (Component, Sequence and State diagrams) and the UML specifications obtained are then used to analyze and verify the initial WS-CDL description. However, exception handling is not considered in this paper, and the formalism they use is simpler than timed automata. They simply obtain data-enriched State Machine Diagrams manually by combining UML class diagrams which capture the WS-CDL data model and UML state machine diagrams, which capture the abstract behavior model of each role in WS-CDL.

We based our exception modeling on [Halvorsen and Haugen, 2006], in which Halvorsen et al. present a method of handling exceptions in UML 2.0 sequence diagrams. This notation distinguishes the exception flow from the normal control flow creating visual separation and providing a way of handling exceptions in both single-threaded programs and multithreaded programs. Using this idea, we will define a new kind of frame, the *exception frame*, which will be associated with interactions (message exchanges) reflecting a fault condition, which can be of diverse types, such as a failure in transmission, a security failure, a time-out associated to the interaction that has expired, etc.

Finally, some tools support the formal development of Web services. Benatallah et al. [Benatallah et al., 2005] present a framework, known as *Self-Serv*, for the design and implementation of a system in which services are composed by a model-driven approach, and the resulting composite services are orchestrated following a peer-to-peer paradigm. Composite services are specified by state-charts, using data conversion rules and multi-attribute provider selection policies. These specifications are interpreted by software components that interact peer-to-peer to generate and coordinate the execution of the composite service. In [Foster et al., 2005a,b] the authors define and implement an Eclipse plug-in (LTSA) for the design of Web service compositions. They use Message Sequence Charts (MSCs) to specify interactions among the participants, and then a WS-CDL description is obtained to represent global coordination as well as a WS-BPEL description of the services involved. In contrast, we use UML sequence diagrams as the starting point and also take into account timed restrictions in the interactions.

3 Description of WS-CDL

A Web services choreography specification offers a precise description of the collaboration between the parties involved in a choreography. WS-CDL specifications are contracts containing "global" definitions of the common ordering conditions and constraints under which messages are exchanged. The contract gives a global description of 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 performed by a combination of the resulting local systems, on the basis of appropriate infrastructure support.

Since in real-world scenarios corporate entities are often unwilling to delegate control of their business processes to their integration partners, a choreography offers a means by which the rules of participation can be clearly defined and jointly agreed to. Each partner may then implement his portion of the Choreography as determined by the common or global view. The aim of WS-CDL is to make it easy to determine the conformance of each implementation to the common view expressed in the WS-CDL.

The WS-CDL model consists of the following entities [Kavantzas et al., 2005]:

– **Participant Types, Role Types and Relationship Types**. A Participant Type groups together those parts of the observable behavior that must be implemented by the same logical entity or organization. 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.

– **Information Types, Variables and Tokens.** Information Types describe the type of information used in a choreography. 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 refer to parts of a Variable. Both Variables and Tokens have Types that define the structure of what the Variable contains or the Token references.

– **Choreographies**: As mentioned above, these establish the common rules that govern the ordering of exchanged messages and collaborative behavior. A WS-CDL document, in general, consists of a hierarchy of choreographies, which are *executed* by using the WS-CDL *perform* activity. However, in our case, WS-CDL descriptions generated by translation from UML 2.0 sequence diagrams will only consist of a single choreography (the so called *root* choreography).

  A choreography in WS-CDL consists of three parts:

  • **Choreography Life-line**: This describes the progress of a collaboration. Initially, the collaboration is established between the parties; so that a task is performed within it and it finally completes either normally or abnormally.

  • **Choreography Exception Block**: This specifies the additional interactions that should occur when a Choreography behaves in an abnormal way.

  • **Choreography Finalizer Block**: This 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.

– **Activities and Ordering Structures.** The collaborative behavior of the participants in a choreography is described by means of *activities*. These are the actions performed within a choreography and are divided into three groups: *basic activities*, *ordering structures* and *workunits*. The basic activities are used to establish the variable values (*assign*), to indicate some inner action of a specific participant (*silent_action*), or that a participant does not perform any action (*noaction*), and also to establish an exchange of messages between two participants (*interaction*). An interaction can be assigned a *time-out*, i.e. a time for its completion. When this time-out expires if the interaction has not been completed the timeout is activated and the interaction finishes abnormally, causing an exception block to be executed in the choreography. Interactions can also fail due to other causes, which can be declared as part of the possible message exchanges of an interaction (see the exception frame part in Table 2, syntax of the *exchange* part of the involved interaction). The additional causes of failure of an interaction can include a failure in the transmission of a message, a security failure (e.g. an unsuccessful user authentication), application failures (e.g., the *goods ordered* are out of stock), etc.

The ordering structures are used to combine activities with other ordering structures in a nested structure to express the ordering conditions under which information within the choreography is exchanged. The ordering structures are *sequence, choice* and *parallel*, with the usual interpretation. Finally, *workunits* are used to allow the execution of certain activities when a certain condition holds. Thus, a workunit encapsulates one activity, which can only be executed if the corresponding guard is evaluated as true. There is a further guard in the workunits in order to allow the iteration of the enclosed activity.

Therefore, considering the ways in which time information can be used in WS-CDL we see that me can have *time-outs* in the interactions, but we can also use date/time variables (and expressions) in a WS-CDL document, by using XPath 2.0. XPath actually provides a number of functions to manage these datatype values, for instance, to delay the execution for a certain time, or to establish the times at which actions can or must be executed. We can then use these date/time variables in the workunits guards in order to establish the time at which the activities inside the workunits can or must be executed.

4 Translation of UML sequence diagram into WS-CDL

In this section we describe the translation of UML 2.0 sequence diagrams into the XML documents written in WS-CDL language. Our starting point is a UML 2.0 sequence diagram extended with frames [Ambler, 2005]. A frame is defined as a unit of behavior and contains, among others, related objects, and the sequence of messages between these objects. A frame is depicted by a solid-outline rectangle with a pentagon in the upper left corner name-label, which makes it easier to refer to, e.g. as a subdiagram.

We will consider the following specific frames in our translation (Figure 1):

- *Alt-labeled frames*: These allow us to specify an *if-then-else* control structure, depending on a condition that follows the *alt* label.

- *Opt-labeled frames*: These allow us to specify an *if-then* control structure, depending on the condition that follows the *opt* label.

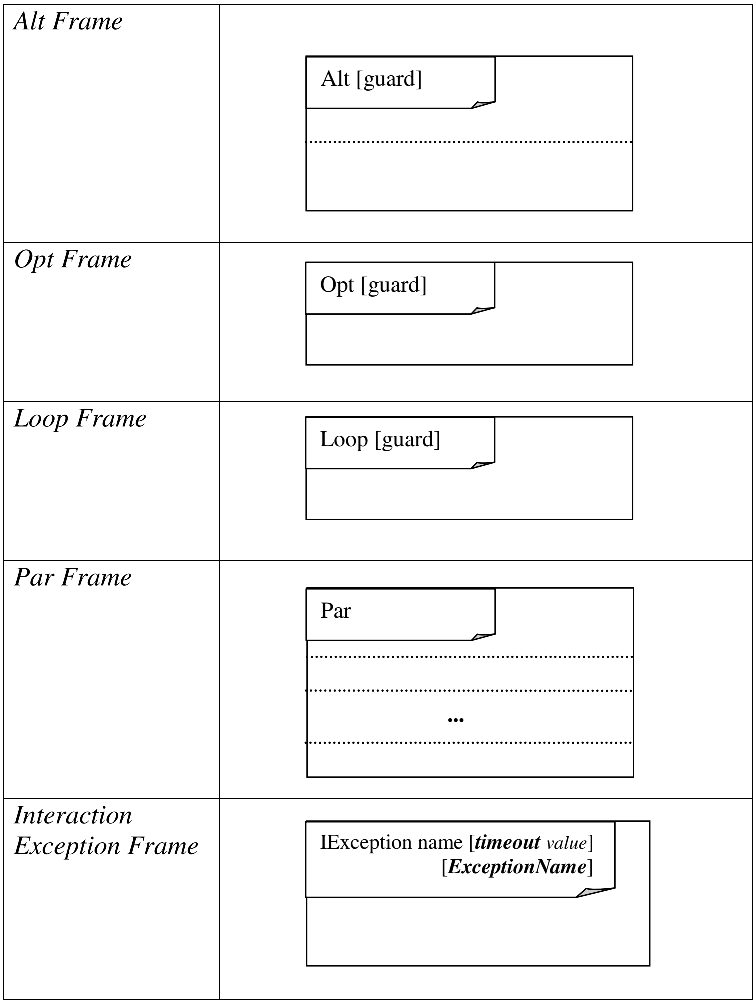

Figure 1: UML 2.0 frames considered in the model.

- *Loop-labeled frames*: These allow us to describe a repetitive behavior, depending on the condition that follows the *loop* label.
- *Par-labeled frames*: These describe parallel message exchange activities.
- *Interaction exception frames*: These are associated with interactions and express fault conditions. They are pictured as separated frames, linked by an arrow to the interaction they are associated with. As an interaction can fail for many different reasons, there may be several interaction exception frames linked to it. It should be noted that an interaction may have an associated

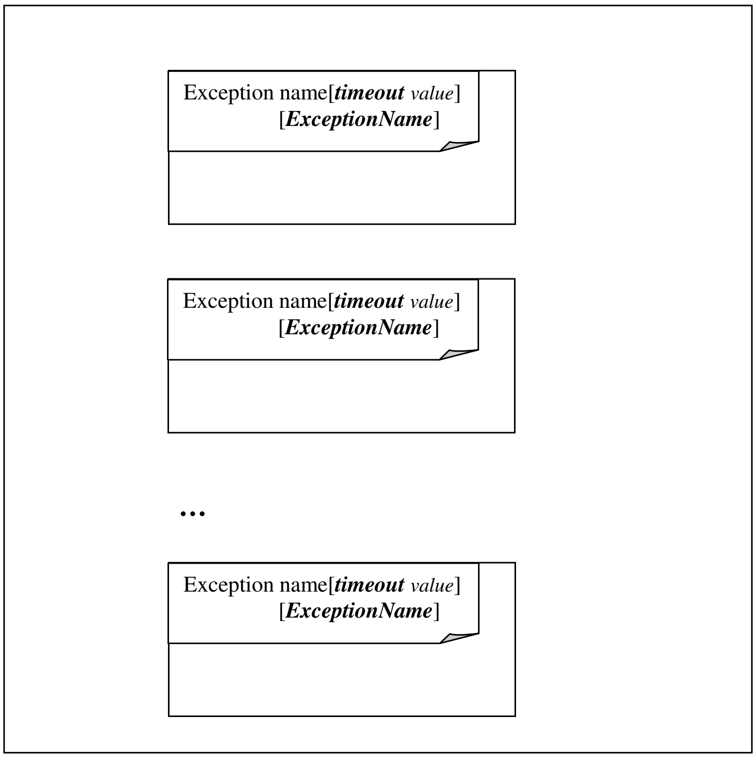

Figure 2: Structure of an exception sequence diagram.

time-out, so we consider a specific type of interaction exception frame for this case (see Figure 1).

Together with the sequence diagram capturing the main control flow, we have the so-called *exception sequence diagram*, which consists of a set of *Exception frames* (see Figure 2), one for each interaction fault condition. Each *Exception frame* is labeled with the corresponding interaction fault condition and only comes into play when this specific error has occurred.

Frames are therefore a powerful tool to describe the control structure of a composite Web service, as well as the exception situations that may arise when two parties interact. Furthermore, we can use frame conditions to specify constraints that can include a wide range of information, such as expressions (using variables), clocks, etc. Frame-extended UML 2.0 therefore allows us to describe composite Web services with time restrictions.

Once we have introduced the UML sequence diagrams with the appropriate frame extension, we can describe the translation, which is made on a structural

UML Diagrams Components	WS-CDL Components	Syntax of the WS-CDL Components
Object	Role	<roleType name="name"> <behavior name = "name" interface="iname" /> </roleType>
	Participant Type	<participantType name="ncname"> <role type="name" /> </participantType>
Messages	Relationship Type	<relationshipType name="name"> <role type="qname" behavior="list of name" /> <role type="qname" behavior="list of name" /> </relationshipType>
	Channels Types	<channelType name="ncname" > </channelType>
	Interaction	<interaction name="iname" operation="iname" channelVariable="ncname"> <participate relationshipType="QName" fromRoleTypeRef="QName" toRoleTypeRef="QName" /> <exchange name="NCname" action="request"> <send variable="XPath-expression" /> <receive variable="XPath-expression" /> </exchange> <exchange name="NCname" action="respond"> <send variable="ACKVariable" /> <receive variable=" ACKVariable " /> </exchange> </interaction>
Timeout		- The time-out line is added to the associated interaction: <interaction name="iname" <exchange name="NCname" … </exchange> **<timeout time-to-complete="XPath-expression"** </interaction> - And a new workunit in the exceptionBlock: <exceptionBlock name="handleException"> **…** **<workunit name="timeout_name"** **guard="cdl:hasExceptionOccurred(xsd:timeout) ">** **Activity** **</workunit>** **…** </exceptionBlock>
Labels and Time Constrains	Time Variables	<variableDefinitions> <variable name="ncname" informationType="qname" roleTypes="RoleTypename" /> </variableDefinitions>

Table 1: Translation of UML 2.0 sequence diagrams elements into WS-CDL.

basis, i.e. we provide the translation for each element in the frame-extended

UML 2.0 sequence diagrams (Tables 1 and 2):

- Objects: The objects in the UML 2.0 sequence diagrams correspond to the WS-CDL role types, which are used to describe the behavior of each class of party involved in the choreography. Each object is then translated into a different WS-CDL role type. We also consider a participant type for the object with its associated role type.
- Messages: Each message in the UML sequence diagram is translated as a WS-CDL interaction with a *Request* exchange element followed by a *Respond* exchange element. For this, we also need to declare a new *channel type*, as well as a new *relationship type*. The channel type declares the channel used for the communication and the relationship type specifies the two role types (objects) involved in this interaction, which has a *Request-exchange* element, corresponding to the sending of the message specified in the UML sequence diagram, as well as a *Respond-exchange* element capturing the message *ack* in the case of successful transmission.

  Furthermore, as we will see below, if this transmission has an associated time-out or any other interaction exception frame, these fault conditions will enrich the interaction syntax by adding elements to it.
- Variables, expressions and time constraints: These are translated by WS-CDL information types, variables and expressions in XPath. From the UML sequence diagram we obtain the corresponding *Information types* for the WS-CDL translation. These are the same types of variables as used in the UML sequence diagram. Each variable is then translated into a WS-CDL variable (with the same name), using the corresponding information type.

  Time constraints are translated into XPath expressions by the appropriate XPath operators, like *op:getCurrentDateTime*, *op:hasDurationPassed* or *op:hasDeadlinePassed*.
- *Par*- labeled frames are translated as *parallel* ordering structures, indicating activities running in parallel.
- *Loop*- labeled frames are translated by WS-CDL *workunits*, indicating iterative computations. The loop condition (expressed in XPath) is thus used both for the workunit guard and for its repetitive behavior.
- *Opt*- labeled frames are also translated as *workunits*. In this case, the Boolean expression following the keyword *opt* is used as the guard for the workunit.
- *Alt*- labeled frames are translated using a *choice* and two guarded workunits as alternatives. The Boolean expression following the keyword *alt* is now used as guard for the first workunit, whose actions are taken from the activities

UML Diagrams Components	Label	Syntax of the WS-CDL Components
Frames.	"alt" label	<choice> <workunit name="ncname" (if part) guard="xsd:boolean Xpath-expression" Activity1 </workunit> <workunit name="ncname" (else part) guard="xsd:boolean Xpath-expression" Activity2 </workunit> </choice>
	"opt" label	<workunit name="ncname" guard="xsd:boolean Xpath-expression" Activity </workunit>
	"loop" label	<workunit name="ncname" guard="xsd:boolean Xpath-expression" repeat="xsd:boolean Xpath-expression"> Activity </workunit>
	"par" label	<parallel> Activity </parallel>
	"IException" label	Two new exchanges are added to the corresponding interaction: <exchange name="NCname" action="respond" faultName="ExceptionName"> <send variable="ExceptionName" causeException="ExceptionName" /> <receive variable="ExceptionName" causeException="ExceptionName" /> </exchange> And a new workunit in the exceptionBlock: <exceptionBlock name="handleException"> ... **<workunit name="ncname"** **guard="cdl:hasExceptionOccurred(xsd:ExceptionName) ">** **Activity** **</workunit>** ... </exceptionBlock>

Table 2: Translation of UML Frames into WS-CDL.

inside the *then*-part of the frame, whereas the guard of the second workunit is the negation of that condition. According to WS-CDL semantics, only the workunit for which the guard is evaluated as true will thus be performed.

– *Interaction exception frames*: These are translated by adding new elements to the interaction they are associated with. We must distinguish between two cases:

  - *Time-out*: When the transmission of a message has an associated timeout, we include a time-out element in the associated interaction (see *Time-out* translation in Table 1).

- Others: For any other interaction exception frame we consider a new *Respond-exchange* element within the associated interaction, with the *faultName* attribute, which is used to indicate that this exchange corresponds to a failure, and also the *causeException* attribute must be included in both the *send* and the *receive* parts of the exchange, indicating the thrown exception.

– *Exception frames*: The exception frames within the *Exception sequence diagram* are translated into WS-CDL using the *choreography exception block*, in which there are as many exception workunits as exception types. An exception workunit is a guarded workunit, in which the *hasExceptionOccurred* WS-CDL function is used to check the exception that has been thrown. Thus, only the corresponding exception workunit is performed.

5 Implementation

This section describes the Web Services Translation tool (WST), which we are at present developing for translating UML 2.0 sequence diagrams into WS-CDL specification documents and the obtained WS-CDL specifications into Timed Automata, which are then used to simulate and verify the system behavior. This tool and its documentation is available at http://www.dsi.uclm.es/retics/WST/. There is also a folder with the XSL files, which contain the transformation rules that capture the translation described in the previous section.

WST implements a part of a top-down methodology for the development of composite Web services. This methodology consists of the following phases:

1. Analysis phase: In this phase we perform the so-called requirement engineering, in which a goal model technology is used to define the main requirements of the system. The specific methodology used is KAOS [Dardenne et al., 1993], and the result is the set of requirements that the system must fulfill.

2. Design phase: In this phase we use UML 2.0 sequence diagrams extended with frames to model the system by producing a system sequence diagram.

3. Choreography generation phase: A WS-CDL choreography is automatically generated.

4. Validation phase: The generated WS-CDL documents are translated into Timed Automata, which can be used in UPPAAL to simulate the system, by running it and checking that it works properly under normal conditions. Some mistakes may appear, in which case we return to the design phase to correct them.

5. Verification phase: Timed Automata are a powerful formalism, which allows us not only to carry out simulations, but also to analyze whether the properties identified in the analysis phase are fulfilled or not. We use the UPPAAL model checker to verify the properties of interest, which have been identified in the first phase. Again, if failures are detected here, we must return to the design phase.

   The requirements defined in the analysis phase are then used in the verification process as the set of properties that must be satisfied. As they are also used to identify the entities and the functionalities of the system, they are also taken into account in the design phase. In the UPPAAL verification process, when a property does not hold, we obtain an execution trace that leads to the point where the property is not satisfied.

   This information can be used to recreate the trace on the sequence diagram in order to identify the design mistake.

WST basically works by applying a number of XSL style sheets [Clark, 1998] to the initial XMI document (representing the UML sequence diagram) in order to obtain the corresponding WS-CDL XML document. Figure 3 shows the XSL transformation rule for the *opt* frame, whereas figures 4 and 5 show in two parts the XSL rule for messages.

6 Verification

WST also translates the obtained WS-CDL specification into a network of timed automata (NTA). The NTA system obtained allows us to validate and verify the expected requirements of the system. The steps followed by WST to translate a WS-CDL document into a timed automata system are described in detail in [Cambronero et al., 2009]; here we only give a brief description of how the translation works.

In the timed automata model that we consider we have non-negative integer variables and urgent edges. The variables can be assigned a value when executing an edge, and their values can be checked in the guards and invariants. Urgent edges inhibit time elapsing when they are enabled.

A function φ is first defined which associates an NTA to every WS-CDL activity, where $\varphi : \mathit{Activities} \times \mathcal{P}_{\mathcal{F}}(C) \times \mathcal{N} \longrightarrow \mathit{NTA} \times \mathcal{P}_{\mathcal{F}}(C)$. The main argument of this function is the activity for which the translation is made, but it has two additional arguments: one set of clocks ($\mathcal{P}_{\mathcal{F}}(C)$) and one location ($\mathcal{N}$). The set of clocks must be reset just before finishing the execution of the generated timed automata (for compositional purposes).

The location is used to transfer the control flow there in the event of a failure. We use this location as an argument in order to link the normal activity flow with the exception part, as we will see below.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:UML="omg.org/UML1.3">

 …

 <xsl:if test="name()='frames'">

     <xsl:if test="starts-with(@name,'opt')">
         <workunit>
              <xsl:attribute name="name">
                    <xsl:value-of select="@name" />
             </xsl:attribute>
              <xsl:attribute name="guard">
                    <xsl:value-of select="@guard" />
            </xsl:attribute>
             <xsl:attribute name="block">
                    <xsl:value-of select="'true'" />
           </xsl:attribute>

          <sequence>
              <xsl:call-template name="frame_content">
                    <xsl:with-param name="identifier" select="@idframe" />
                    <xsl:with-param name="type" select="@name" />
            </xsl:call-template>
          </sequence>
        </workunit>
     </xsl:if>

…

</xsl:if>
```

Figure 3: XSL rule for “opt” frame.

We will denote the first projection of φ by $\varphi_1(A, C, l)$, i.e. the obtained NTA, and its second projection by $\varphi_2(A, C, l)$, i.e. the set of clocks that should be reset when using this NTA compositionally.

A choreography is now defined as a pair $(A_1, \{A_{2_i}\}_{i \in I})$, where A_1 is the activity of the choreography *life-line*, and $\{A_{2_i}\}_{i \in I}$ is the set of exception workunits of its exception block, which can be empty (denoted by $\emptyset$), because the exception block is optional.

Thus, given a choreography $\mathcal{C} = (A_1, \{A_{2_i}\}_{i \in I})$, we define its associated NTA as follows (Figure 8):

- We first create a location ‘*de*’, which we call the “double exception location”, which is used as the location to which the control flow is transferred in the event of a failure within an exception activity A_{2_i}. We then generate $\varphi(A_{2_i}, \emptyset, de)$, for $i \in I$.
- We now create the exception location ‘*e*’, to which the control flow is transferred in the event of failure in A_1, and then we generate $\varphi(A_1, \emptyset, e)$.
- We connect the exception location ‘*e*’ with the initial locations of the NTAs

corresponding to the exception workunits, according to the exception thrown, by means of an urgent edge, which must reset all the clocks in $\varphi_2(A_{2_i}, \emptyset, de)$, with $i \in I$. As can be seen in Figure 8, urgent edges are graphically distinguished by a white arrowhead and are given top priority when enabled.

Figures 6 and 7 show how the function φ is defined for the different activities. It can be seen, that all the obtained automata have both one initial and one final location, this property being preserved by all the constructions. It should be noted that according to our description, the current translation for interactions only considers the failures due to time-outs and unassigned source variables. Furthermore, in the event of a failure, all of these constructions transfer the control flow to the location indicated as parameter in the function φ, and reset the clocks indicated as parameters in all the edges reaching the final location.

As a result of this translation, we obtain an NTA representing the behavior

```
<xsl:if test="name()='interaction'">
<interaction>
     <xsl:attribute name="name">
        <xsl:value-of select="@name" />
        <xsl:text>_interaction</xsl:text>
     </xsl:attribute>
     <xsl:attribute name="operation">
            <xsl:value-of select="@operation" />
     </xsl:attribute>
     <xsl:variable name="var1" select="participate/@fromRole" />
     <xsl:variable name="var2" select="participate/@toRole" />
<xsl:for-each select="//package/relation">
 <xsl:for-each select="child::*">
     <xsl:variable name="rol2" select="role2" />
     <xsl:variable name="rol1" select="role1" />

  <xsl:if test="not(repe='si')">
    <xsl:if test="$rol1!=$rol2">
      <xsl:if test="not(canalrep)">
               <xsl:variable name="canal1" select="substring-
                                                before(role1,'RoleType')" />
  ...
        <xsl:if test="((($var1=role1) or (role2=$var1)) and (($var2=role1)
                                                 or  (role2=$var2)))">
          <xsl:attribute name="channelVariable">
             <xsl:value-of select="$canal1" />
             <xsl:value-of select="$canal2" />
             <xsl:text>Channel</xsl:text>
          </xsl:attribute>
        </xsl:if>
      </xsl:if>
    </xsl:if>
  </xsl:if>
 </xsl:for-each>
</xsl:for-each>
```

Figure 4: XSL rule translating a UML message into a WS-CDL interaction (I).

```
 …
<participate>
   <xsl:attribute name="relationshipType">... </xsl:attribute>
   <xsl:attribute name="fromRole"> ... </xsl:attribute>
   <xsl:attribute name="toRole"> ...   </xsl:attribute>
</participate>

<exchange>
     <xsl:attribute name="name">
         <xsl:value-of select="exchange/@name" />
     </xsl:attribute>
     <xsl:attribute name="action">
        <xsl:value-of select="exchange/@action" />
     </xsl:attribute>
  <xsl:if test="exchange/send">
     <send>
        <xsl:attribute name="variable">
             <xsl:value-of select="exchange/send/@variable" />
        </xsl:attribute>
     </send>
 </xsl:if>

 <xsl:if test="exchange/receive">
     <receive>
         <xsl:attribute name="variable">
         <xsl:value-of select="exchange/receive/@variable" />
         </xsl:attribute>
     </receive>
 </xsl:if>
</exchange>
<xsl:if test="timeout">
    <timeout>
        <xsl:attribute name="time-to-complete">
             <xsl:value-of select="timeout" />
        </xsl:attribute>
    </timeout>
</xsl:if>
  ...
</interaction>
```

Figure 5: XSL rule translating a UML message into a WS-CDL interaction (II).

of the composite Web service. This NTA representation can be directly used in the UPPAAL tool, to carry out simulations and also to verify the properties that were identified in the analysis phase. The following case study is an illustration of the above explanation.

7 Case Study: Purchase Process

Let us consider a typical purchase process in an Internet business context. There are three participants involved in this example: a client, a supplier and a deliverer. The Internet purchase works as follows: *"A client wants to buy a product using the Internet. There are several suppliers that offer different products on*

Internet Servers based on Web-pages. The client contacts a supplier in order to buy the desired product. The supplier confirms the order and contacts a deliverer, who transports the product to the client".

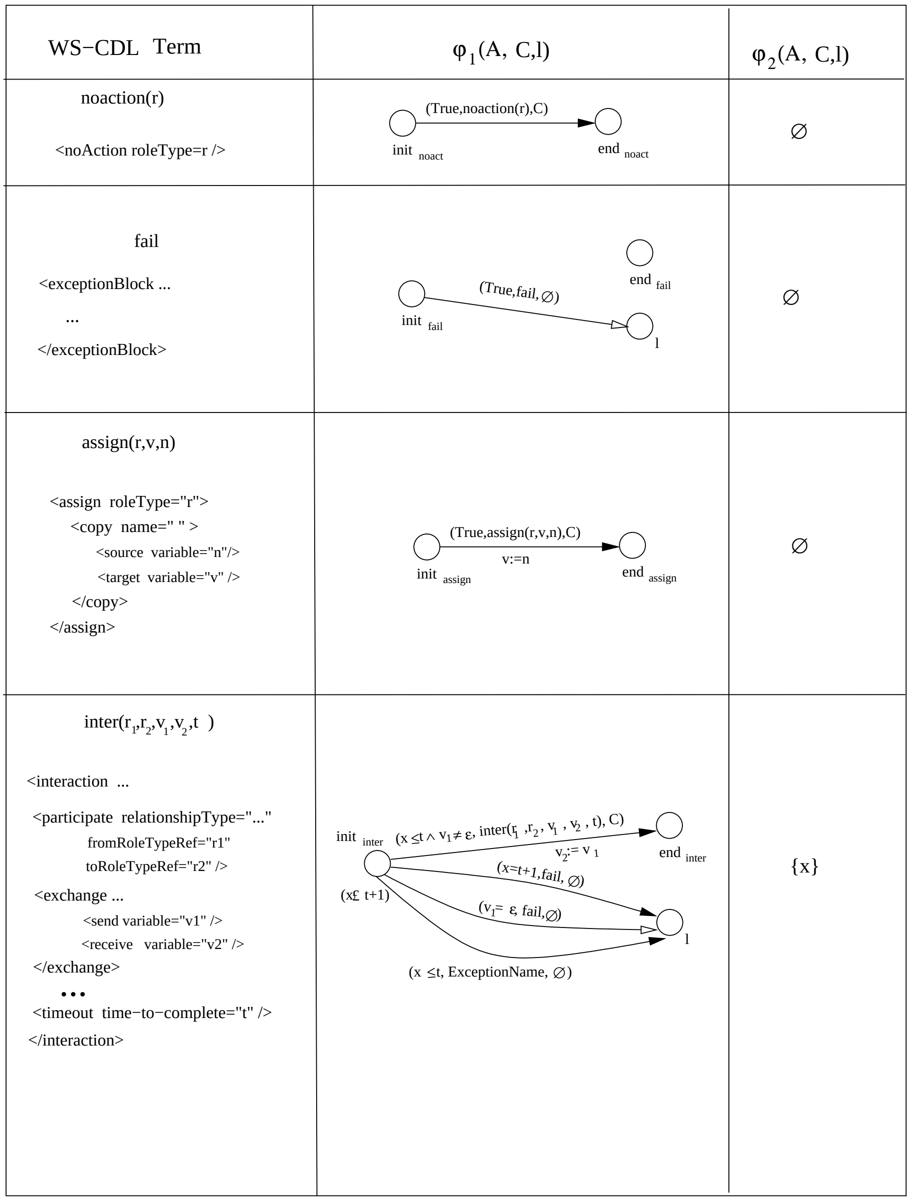

Figure 6: From WS-CDL to NTA (I).

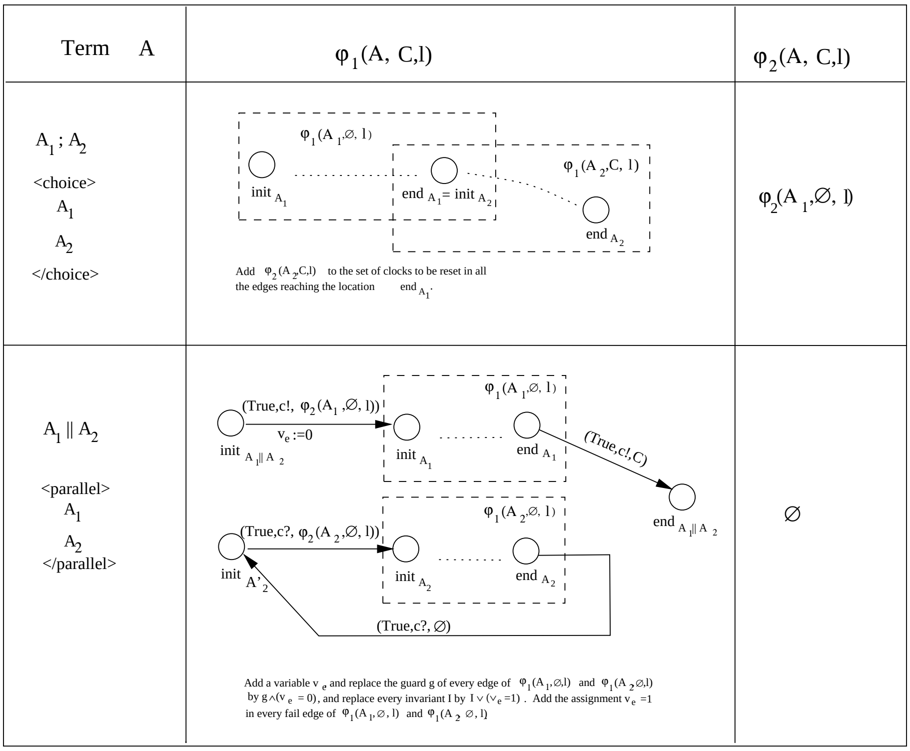

Figure 7: From WS-CDL to NTA (II).

The behavior of each participant is the following:

- Client: He contacts the supplier to buy a product. He must send the supplier the appropriate information on the product and payment data. After the payment has been correctly processed, he expects to receive the product from the deliverer within the agreed time (48 hours). If he does not receive the product in the agreed time, the payment is refunded.
- Supplier: He receives the order and payment information. The supplier sends an acknowledgment to the client, unless the product is out of stock, and contacts a deliverer to transport the product if the payment has been correctly processed.
- Deliverer: He picks up the order with the client information and delivers it to the client.

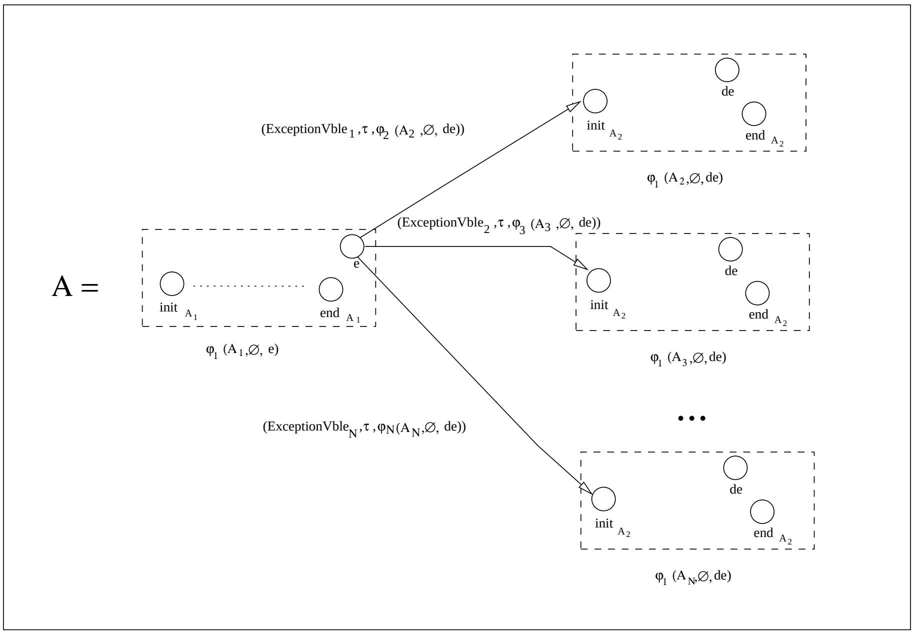

Figure 8: From WS-CDL to NTA (III).

7.1 Analysis phase

We have identified two different kinds of requirement for this system. One refers to the correct behavior of the system, while the other refers to the quality of the service offered. In the first requirement there is a timing constraint, which is the maximum delivery time. Payment information must also be checked for processing and security issues. The second class, related to performance issues, also has two requirements: the speed of the service and efficient request processing.

Figure 9 depicts the KAOS goal-model that we developed for this case study. The root goal *"CorrectInternetPurchaseProcess"* consists of two subgoals joined by an *And-refinement*, which means that both subgoals must be fulfilled to achieve the root goal:

– The first goal-model, *"BehavesProperly"*, which is of the "maintain" type, is refined by another *And-refinement*, with two leaf goals: *"PickupOnTime"*, of "Unbound Respond" type, which requires the deliverer to pick up the order on time, otherwise the payment will be refunded, and *"CorrectPaymentProcess"*, of "maintain" type, which specifies that the delivery process will not occur if the payment information is not valid.

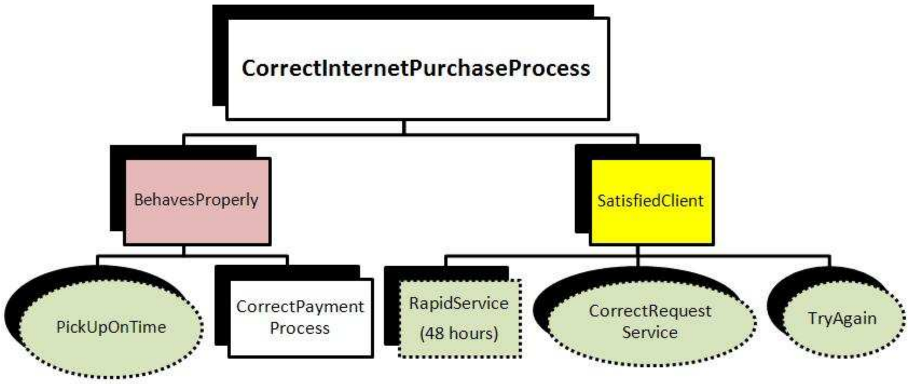

Figure 9: The goal-model for the Internet Purchase Process.

- The second *"SatisfiedClient"*, of "PossibleAlways" type, consists of three leaf goals that refine the parent goal by an *And-refinement*:
  - *"RapidService"*, of "Achieve" type, determines that the client will receive the product on time, i.e., within 48 hours after payment.
  - *"CorrectRequestService"*, of "Unbound Respond" type, indicates that the product request will only be initiated if the product is not out of stock.
  - *"TryAgain"*, of "Unbound Respond" type , specifies that the client must be able to repeat the purchase process if the payment is not correct.

7.2 Design phase

Figure 10 depicts the UML sequence diagram corresponding to the purchase process, created with WST. In this snapshot we can see the three actors involved in this system with two alt-frames below. The first alt-frame captures the possibility of sending correct or incorrect payment information. The second alt-frame uses a Boolean variable called "ValidPayment". Thus, when the payment information is valid, the Supplier contacts with the Deliverer, who must send the product within the stipulated time (otherwise an exception is thrown).

We can also see two exception-frames at the end of the diagram. The first one sends a notification to the Client when the Supplier is out of stock. The other exception frame states the refund if the product has not been delivered in the stipulated time.

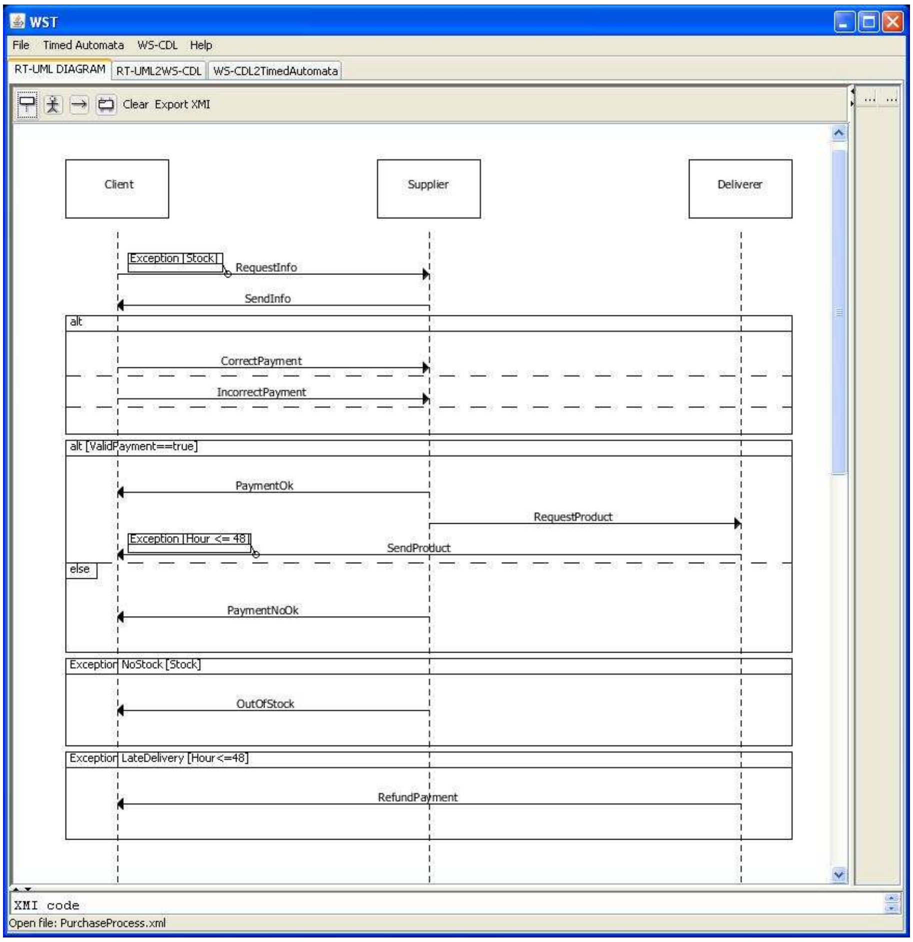

Figure 10: UML sequence diagram for the Internet Purchase Process.

From this UML sequence diagram we obtain the corresponding WS-CDL code by applying the translation introduced in Section 4. A piece of the generated WS-CDL code is shown in Figure 11, and a snapshot of WST producing this code in Figure 12.

7.3 Validation and Verification phase

The WS-CDL code can now be translated into an NTA, by using another tab of the WST interface (Figure 13), which implements the translation described in Section 6. This NTA (Figure 14) can then be used to accomplish the validation and verification phase.

```
<?xml version="1.0" encoding="UTF-8" ?>
<package author="SCTR Group" name="" version="1.0">
  ...
<choice>
  <sequence>
    <interaction name="RequestInfo_interaction" operation="RequestInfo" channelVariable="Client2SupplierChannel">
        <participate relationshipType="ClientSupplier" fromRole="ClientRoleType"  toRole="SupplierRoleType" />
        <exchange name="RequestInfoExchange" action="request" />
        <exchange name="RequestInfoExchangeACK" action="respond">
            <send variable="ACKVariable" />
            <receive variable="ACKVariable" />
        </exchange>
        <exchange name="ExchangeExceptionStock"  faultName="ExchangeExceptionStock" action="respond">
              <send variable="Stock" causeException="tns:Stock" />
              <receive variable="Stock" causeException="tns:Stock" />
        </exchange>
    </interaction>                                    RequestInfo Message with Exception
    <interaction name="SendInfo_interaction" operation="SendInfo" channelVariable="Client2SupplierChannel">
        <participate relationshipType="SupplierClient" fromRole="SupplierRoleType"  toRole="ClientRoleType" />
            ...
    </interaction>
    <choice>
      <sequence>
        <interaction name="CorrectPayment_interaction"
            operation="CorrectPayment" channelVariable="Client2SupplierChannel">
              <participate relationshipType="ClientSupplier" fromRole="ClientRoleType" toRole="SupplierRoleType" />
          ...
        </interaction>
      </sequence>

      <sequence>
        <interaction name="IncorrectPayment_interaction"
            ...
        </interaction>
      </sequence>                                     Frame labelled with alt
    </choice>
     ...
    <choice>
     <workunit name="alt_else1_if" guard="ValidPayment==true" block="false">
       <sequence>
         <interaction name="SendProduct_interaction" operation="SendProduct"
                         channelVariable="Deliverer2ClientChannel">
              <participate relationshipType="DelivererClient" fromRole="DelivererRoleType"  toRole="ClientRoleType" />
              <exchange name="SendProductExchange" action="request" />
              <exchange name="SendProductExchangeACK" action="respond">
                  <send variable="ACKVariable" />
                  <receive variable="ACKVariable" />
              </exchange>
              <timeout time-to-complete="Hour <= 48" />
         </interaction>
       </sequence>
     </workunit>
     <workunit name="alt_else1_else" guard="ValidPayment!=true" block="false">
      ...
     </workunit>                                Frame labelled with alt[ValidPayment==true]
    </choice>
  </sequence>
   ...
 <exceptionBlock name="handleException">
   <workunit name="exception1" guard="hasExceptionOccurred(Stock)">
     <sequence>
        <interaction name="OutOfStock_interaction" operation="OutOfStock"
              channelVariable="Client2SupplierChannel">
               <participate relationshipType="SupplierClient" fromRole="SupplierRoleType" toRole="ClientRoleType" />
                <exchange name="OutOfStockExchange" action="request" />
                <exchange name="OutOfStockExchangeACK" action="respond">
                     <send variable="ACKVariable" />
                     <receive variable="ACKVariable" />
                </exchange>
        </interaction>
     </sequence>                                Exception Block for Out of Stock Exception
   </workunit>
   <workunit name="exception2" guard="hasExceptionOccurred(Hour<=48)">  ... </workunit>
 </exceptionBlock>
 </choreography>
</package>
```

Figure 11: A piece of the Internet purchase process WS-CDL code.

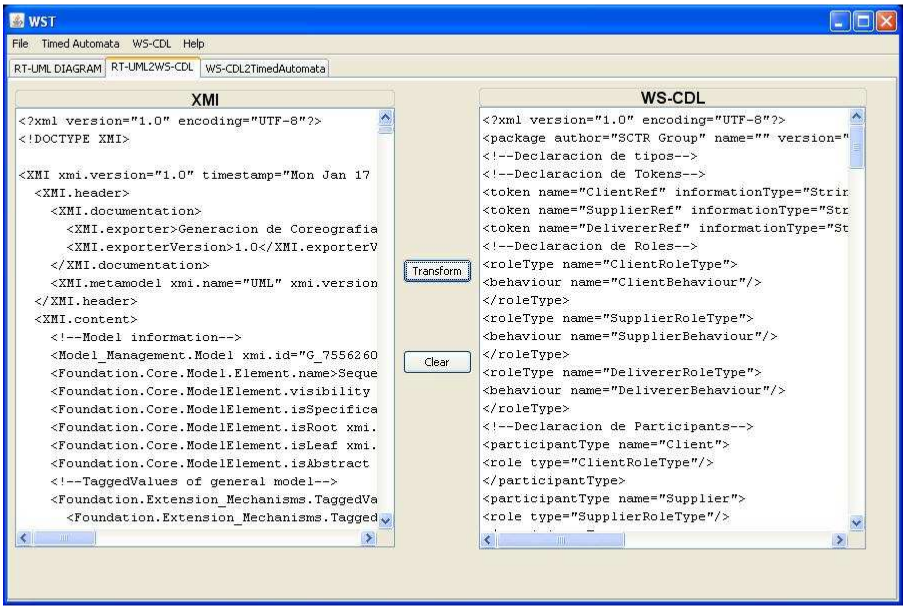

Figure 12: From XMI-UML representation to WS-CDL with WST.

Validation is by means of simulation, where we check that the system behaves as expected in normal conditions. For this we use the UPPAAL simulator, which can be used in three different ways:

– The system can be run manually, selecting the transitions to be executed at each step.

– The system can run on its own and the transitions to be performed are therefore selected randomly.

– The user can run a trace extracted from the verifier. This is usually done in the event of a failure when testing a property, in order to analyze the trace that the verifier provides us, which leads to the state at which the property does not hold.

Figure 15 depicts a snapshot of the UPPAAL simulator in which we can see the simulation of the Internet Purchase Process.

After validation, verification starts, in which we check the properties identified in the analysis phase (leaf goals of the KAOS goal-model).

– The first leaf goal, *"PickupOnTime"*, specifies that the deliverer must pick up the order on time, i.e., it cannot take more than 48 hours after receiving the

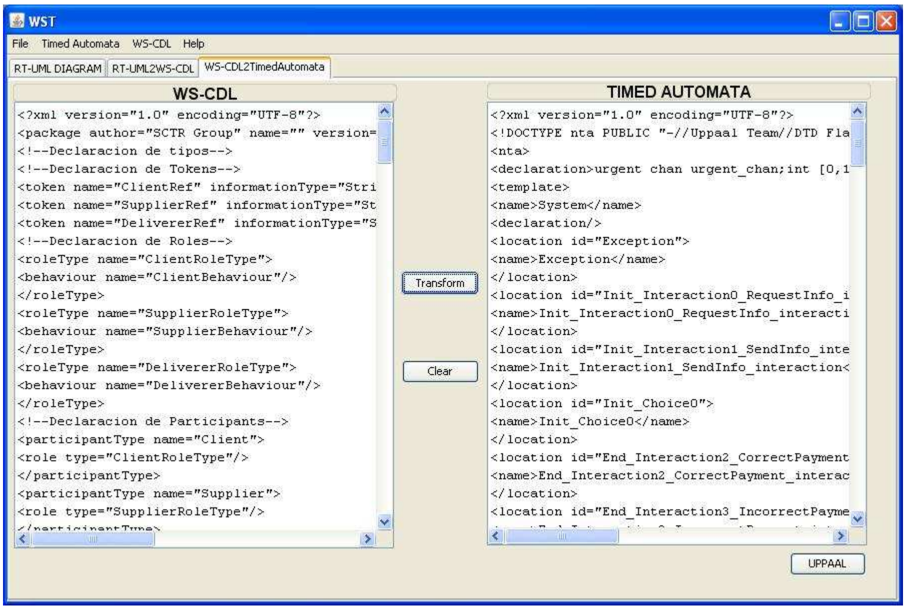

Figure 13: Translating WS-CDL code to NTA with WST.

request, or an exception will be thrown and the payment will be refunded. This property can be written as follows:

$$(System.Init\_Interaction6\_SendProduct\_interaction \wedge Hour > 48) --> \\ System.Init\_Interaction9\_RefundPayment\_interaction$$

We obtain that this formula is **satisfied**.

– The second leaf goal, *"CorrectPaymentProcess"*, specifies that the product will not be delivered if the payment information is not valid. This property is written as follows:

$$A[] \quad ValidPayment == false \quad imply \\ not \quad System.Init\_Interaction5\_RequestProduct\_interaction$$

We obtain that this formula is **satisfied**.

– The third leaf goal, *"RapidService"*, establishes that the client will receive the product on time. This property is written as follows:

$$E <> \quad (System.End\_Interaction6\_SendProduct\_interaction \wedge Hour < 48)$$

We obtain that this formula is **satisfied**.

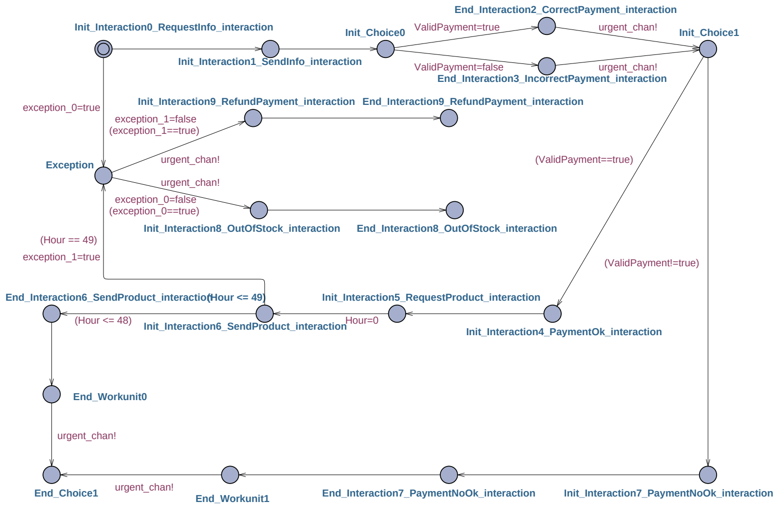

Figure 14: NTA obtained for the Internet Purchase Process.

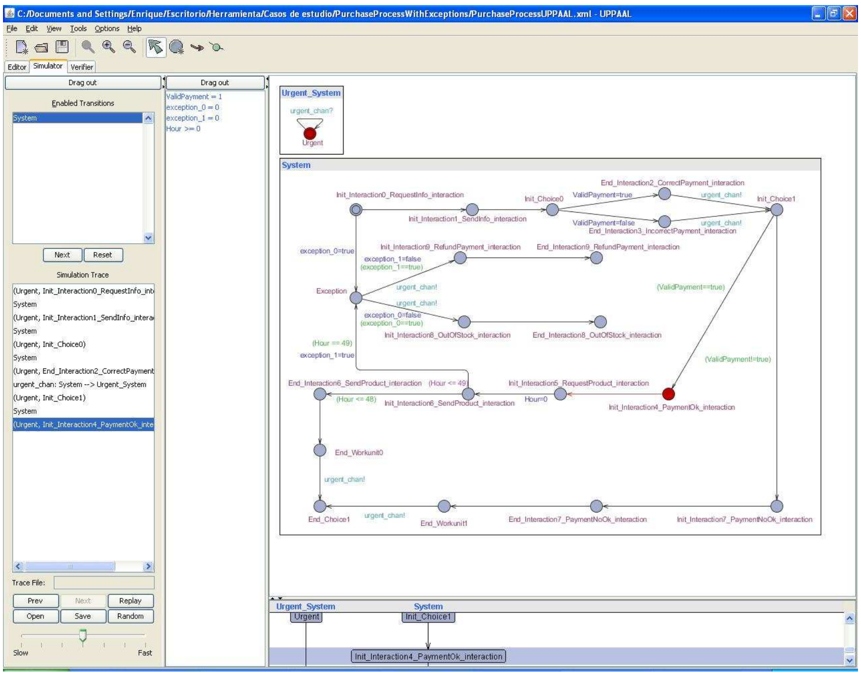

Figure 15: Simulation of the Internet Purchase Process.

- The fourth leaf goal, *"CorrectRequestService"*, specifies that the purchase process will only be initiated if the product is not out of stock. This property is specified as follows:

$$System.End\_Interaction8\_OutOfStock\_interaction --> \\ not \quad System.Init\_Interaction1\_SendInfo\_interaction$$

We obtain that this formula is also **satisfied**.

- The last leaf goal, *"TryAgain"*, specifies that the client must be able to repeat the purchase process if the payment is not correct. This property is written as follows:

$$System.End\_Interaction7\_PaymentNoOk\_interaction --> \\ System.Init\_Interaction1\_SendInfo\_interaction$$

We now obtain that this formula is **not satisfied**.

At this point we have found an error in our design, so we have to go back to the sequence diagram and fix the problem. In this example the error can be easily solved by adding a "loop" type frame wrapping all the payment process. Thus, the payment process can be repeated until it finishes correctly. After this modification, we must again perform the translation into WS-CDL and timed automata, and repeat the validation and verification phase.

8 Conclusions and Future Work

In this paper we have presented a general methodology for the correct development of Web services based on UML 2.0 sequence diagrams as the starting specifications of composite Web services. These diagrams are then translated into WS-CDL descriptions, which, in turn, are translated into a network of timed automata.

We used UML sequence diagrams extended with frames, in order to include the required control structures. They provide a precise description of the collaboration and message flow between the parties involved in a composite Web service and also a choreographic viewpoint of the system by using the sequence diagrams with frames. Furthermore, we introduced a new type of frame (interaction exception frames and the associated exception frames in the exception sequence diagram) to capture the fault conditions that can be associated with interactions, including the possible associated time-outs.

The translations presented are therefore a powerful tool for the development of correct Web services, since the final product obtained, the timed automata representation, can be used to accomplish the validation and verification of the system.

In future work we will address the issue of completing these translations by including additional features, such as a hierarchy of choreographies, finalizer blocks, and more rigorous treatment of variables (alignments).

References

[Alor-Hernández et al., 2009] G. Alor-Hernández et al. Mapping UML Diagrams for Generating WS-CDL Code. In *ICDS '09:The Third International Conference on Digital Society*, pages 229-234, Cancun, Mexico, 2009. IEEE Computer Society.

[Ambler, 2005] S. W. Ambler. *The Elements of UML(TM) 2.0 Style*. Cambridge University Press, New York, NY, USA, 2005.

[Arkin et al., 2004] A. Arkin et al. Web Services Business Process Execution Language 2.0, December 2004. http://www.oasis-open.org/committees/download.php/10347/wsbpel-specification-draft-120204.htm.

[Benatallah et al., 2005] B. Benatallah, M. Dumas and Q. Z. Sheng. Facilitating the Rapid Development and Scalable Orchestration of Composite Web Services. Volume 17, number 1 of *Distrib. Parallel Databases*, pages 5–37, 2005.

[Cambronero et al., 2011] M. E. Cambronero, G. Díaz, E. Martínez and V. Valero. WST: A Tool Supporting Timed Composite Web Services Model Transformation. In *Simulation: Transactions of the Society for Modeling and Simulation International journal*, to appear in 2011.

[Cambronero et al., 2006] M. E. Cambronero, G. Díaz, J. J. Pardo, V. Valero and F. Pelayo. RT-UML forModeling Real-TimeWeb Services. In *In proceedings of Modeling, Design, and Analysis for Service Oriented Architecture Workshop, mda4soa, SCC 2006*, pages 131–139, Chicago, USA, 2006. IEEE Computer Society.

[Cambronero et al., 2009] M. E. Cambronero, V. Valero, G. Díaz and E. Martínez. Web Services Choreographies Verification. Technical Report DIAB-09-04-1. University of Castilla-La Mancha, 2009.

[Cambronero et al., 2010] M. E. Cambronero, V. Valero and G. Díaz. Verification of Real-Time Systems Design. Volume 17 of *Software Testing, Verification and Reliability Journal*, pages 3–37, 2010.

[Clark, 1998] J. Clark. XSL Transformations (XSLT) Version 1.0. Technical Report REC-xml-19980210, W3C, 1998. `http://www.w3.org/TR/xslt`.

[Dardenne et al., 1993] A. Dardenne, A.van Lamsweerde and S. Fickas. Goal-directed Requirements Acquisition, 1993, volume 20, pages 3–50. http://www.w3.org/TR/2004/WD-wsdl20.

[Díaz et al., 2006] G. Díaz and M. E. Cambronero, J. J. Pardo, V. Valero and F. Cuartero. Automatic Generation of Correct Web Services Choreographies and Orchestrations with Model Checking Techniques. In *Proceedings of International Conference on Internet and Web Applications and Services ICIW*, vol. 1257, pages 33-40, 2006. IEEE Computer Society Press.

[Foster et al., 2005a] H. Foster, S. Uchitel, J. Magee and J. Kramer. Leveraging eclipse for integrated model-based engineering of web service compositions. In *eclipse: Proceedings of the OOPSLA workshop on Eclipse technology eXchange*, pages 95–99, New York, NY, USA, 2005. ACM.

[Foster et al., 2005b] H. Foster, S. Uchitel, J. Magee and J. Kramer. Tool support for model-based engineering of web service compositions. In *ICWS: Proceedings of the IEEE International Conference on Web Services*, pages 95–102, Washington, DC, USA, 2005. IEEE Computer Society.

[Graf et al., 2006] S. Graf, I. Ober and I. Ober. Timed annotations with UML. Volume 8, number 2 of *International Journal on Software Tools for Technology Transfer*, pages 113–127, 2006.

[Halvorsen and Haugen, 2006] O. Halvorsen and O. Haugen. Proposed Notation for Exception Handling in UML 2 Sequence Diagrams. In *ICWS: Proceedings of the IEEE International Software Engineering Conference, Australian*, p ages 29-40, WLos Alamitos, CA, USA, 2006. IEEE Computer Society.

[Hamadi and Benatallah, 2003] R. Hamadi and B. Benatallah. A Petri Net-based Model for Web Service Composition. In *ADC '03: Proceedings of the 14th Australasian database conference*, 2003.

[Kavantzas et al., 2005] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon and C. Barreto. Web Service Choreography Description Language (WSCDL) 1.0. http://www.w3.org/TR/ws-cdl-10/.

[Larsen et al., 1997] K. G. Larsen, P. Pettersson and W. Yi. Uppaal in a nutshell. Volume 1, numbers 1-2 of *International Journal on Software Tools for Technology Transfer (STTT)*, pages 134–152, December 1997.

[OMG, 2002] OMG. Response to the OMG RFP for Schedulability, Performance and Time, v. 2.0. OMG document ad/2002-03-04, March 2002.

[Lorenz and Six, 2006] A. Lorenz and H.-W. Six. Tailoring uml activities to use case modeling for web application development. In *CASCON: Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research*, pages 333–338, New York, NY, USA, 2006. ACM.

[OMG, 2003] OMG *UML 2.0 Superstructure proposal v.2.0.* In *OMG document ad/03-01-02*, January, 2003.

[Piotrowski and Krawczyk, 2008] M. Piotrowski and H. Krawczyk. Using UML/WS-CDL for Modeling Negotiation Scenarios. In *IFIP International Federation for Information Processing*, pages 119-126, Volume 283, 2008. Springer Boston.

[Weerawarana et al., 2004] S. Weerawarana et al. Web services description language (wsdl) version 2.0 part 1: Core language, 2004. http://www.w3.org/TR/2004/WD-wsdl20.

[W3C, 2011] World Wide Web Consortium (W3C). *http://www.w3.org/.*

[Zarras et al., 2004a] A. Zarras, P. Vassiliadis and V. Issarny. UML Profile for Schedulability, Performance, and Time Specification, Version 1.1. OMG, December 2004 http://www.omg.org/docs/smsc/04-12-05.pdf.

[Zarras et al., 2004b] A. Zarras, P. Vassiliadis and V. Issarny. Model-driven dependability analysis of webservices. In Robert Meersman and Zahir Tari, editors, *CoopIS/DOA/ODBASE (2)*, volume 3291 of *Lecture Notes in Computer Science*, pages 1608–1625. Springer, 2004.

[Zhang et al., 2008] P. Zhang, B. Li, H. Muccini, Y. Zhou and M. Sun. Data-Enriched Modeling and Verification of WS-CDL Based on UML Models. In *ICWS '08: Proceedings of the 2008 IEEE International Conference on Web Services*, pages 752–753, Washington, DC, USA, 2008. IEEE Computer Society.