Analyzing Web Service Contracts - an aspect oriented approach

Emilia Cambronero
Department of Computer Science
University of Castilla-La Mancha
SPAIN
Email: emicp@info-ab.uclm.es

Joseph C. Okika
Department of Computer Science
Aalborg University, Aalborg
DENMARK
Email: ojc@cs.aau.dk

Anders P. Ravn
Department of Computer Science
Aalborg University, Aalborg
DENMARK
Email: apr@cs.aau.dk

***Abstract*—Web services should be dependable, because businesses rely on them. For that purpose the Service Oriented Architecture has standardized specifications at a syntactical level. In this paper, we demonstrate how such specifications are used to derive semantic models in the form of (timed) automata. These can be used to model check functional and behavioural properties of a given service. Since there might be several specifications dealing with different aspects, one must also check that these automata are consistent, where we propose to set up a suitable simulation relation. The proposed techniques are illustrated with a small case study.**

I. Introduction

The interest in web services has grown in recent times as more and more intra/inter-organizational applications use this model. Thus there is consensus today, that a web service is a programmable component that provides a service and is accessible over the Internet. They are based on standards like SOAP [1], [2], [3], can be standalone, or linked together to provide enhanced functionality. Managing Aero-Electric Wind Turbines, buying airline tickets, accessing an on-line calendar, and obtaining tracking information for your shipments are all business functions that are implemented as web services.

Businesses depend on web services, therefore their properties are of great importance, and informal checking and consensus approaches to when a service is good enough may not suffice. A business will only reluctantly use the offered enterprise applications, because of the high risks involved in using untrusted services from unknown providers. Formal contracts defining the desired properties are therefore studied intensively today, because they are a way to manage the risks that come with the interaction among these inter-organizational services.

Traditionally, contracts in an object oriented setting consider only the functional aspect (pre-condition, post-condition, invariant) of an interface specification. A pre-condition is a constraint than must be satisfied before calling a method or operation; it checks for valid arguments. A post-condition is a corresponding property that is true when the call completes; it is the input-output relation. Finally, an invariant is a constraint on the state of an object; it must hold before or after any operation, and clearly after initialization of the object. These concepts, as popularized by Meyer's "Design by Contract" [4], are, however, part of the complex picture for web services. Since web services are intrinsically distributed, they are by nature concurrent programs, and thus their overall functionality depends not only on correct implementation of the local functionality by sequential algorithms, but even more on the interplay between local functionality and global behavior (protocols and timing).

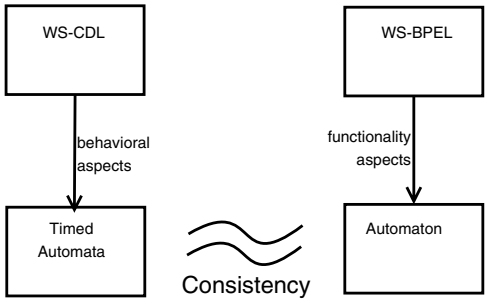

Fig. 1. Analysis of Web Service Contracts

With SOA it is possible to have a detailed and standardized contract specification as found in WS-BPEL [5] and WS-CDL [6]. That leaves the interesting question of how the contracts are used in web service development. Everyone can agree that the contracts have to be satisfied by all the parties involved, but this means that there should be a possibility to take a contract for a web service, and

- check implementations for conformance,
- and analyze the contract for consistency.

The contribution of this paper is a solution to these questions following the approach in Figure 1 by

- a translation of the behavioral aspects of a contract to a timed automata for model checking,
- a translation of the functionality to another automaton,
- a check for the consistency of the two automata.

The approach thus covers two major aspects of contracts, and the approach lends itself to generalization to further quantitative aspects, e.g. performance analysis with queuing models. Here, performance would be analyzed with model, and consistency checked with the other models.

Overview: In Section II, we give a detailed presentation of Web Service contracts where the aspects of contracts are

described. We introduce in this section, a case study of an Wind Turbine Management System. Section III details the analysis of Web Service contracts. General consistency, satisfiability, consistency and application specific issues are presented. A discussion of related work follows and finally, we conclude in Section V.

II. Web Service Contracts

In order to come up with a web service contract specification, different levels of a contract are considered:

A. Contract Levels/Aspects

From an Object Oriented view, an interface describes various methods supported by an object and those which can be invoked by other objects. An Interface Definition Language (IDL) allows the definition of objects based on their interfaces without been concerned with how those interfaces are implemented. Conventionally, interface definitions specify modules and their interface names and operation signatures. Thus, the interface definitions are the contract.

In a service-oriented view, functionality is defined through services which can send and receive messages. With this, applications are composed by combining services that interact through message exchanges. This is done by forming a contract which the interacting services must agree on. Thus, a contract is a specification of the way a consumer of a service will interact with the service provider. A service contract specify functional, behavioral and other aspects. A contract thus defines a runtime dependency between the provider and the consumer.

None of the present contract frameworks combine both the functional, behavioral and QoS aspects, or say much about how the properties should be analyzed. However, we have focused on two of the more popular approaches: Business Process Execution Language (WS-BPEL) and the Web Services Choreography Description Language (WS-CDL).

They are typical and as illustrated in the case study below, they specify different aspects.

B. Example

We describe in this subsection, a case study of a Wind Turbine Management System. The system monitors and controls wind turbines, and it has several components which could be web services located in different places. We focus on three of these components, because it gives us the scenario needed to specify a web service contract. The components are briefly described below and shown as an UML component diagram in Figure 2.

- Wind Turbine Management: sends a report to Productivity management every hour.
- Productivity Management: receives and analyzes the report from Wind Turbine Management.
- Demand Management: generates a report of power needs for Productivity Management.

We look at this example from two perspectives; WS-CDL and WS-PBEL. WS-CDL provides a definition of the information formats being exchanged by all participants. In other words, it specifies the protocols. WS-BPEL provides the message exchanges as viewed by one participant. It describes a business process at a high level.

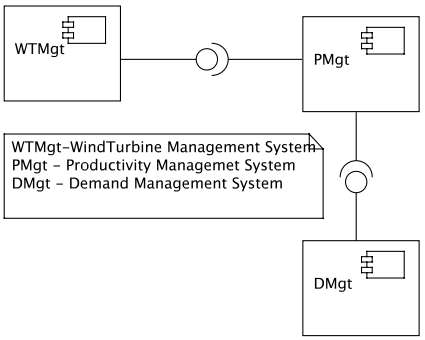

Fig. 2. Wind Turbine Management System Components

C. Contract Aspects in WS-CDL

CDL offers a model for specifying a common understanding of message exchanges. The key aspects of contracts in WS-CDL is itemized below accompanied by the syntax.

- Interface: WSDL defines where it is possible to find different kinds of interface types depending on the specific required web services *interface*.
  In WS-CDL, each interface is associated with a particular role, the syntax is the following:

```
<roleType name="NCName">
   <behavior name="NCName" interface="QName"? />+
</roleType>
```

- Functional Specification: pre-conditions, post-conditions and invariants.
  In WS-CDL these elements are defined by means of *workunits*; which are guarded activities.

```
<workunit name= "demand increase detected"
    guard="cdl:equal(cdl:getVariable
        ('tns:Clock1','',''),'0:00')"
       block="true">
   <assign roleType="tns:DemandRoleType">
    <copy name="calculateincrease"
                causeException=''true">
        <source variable="true"/>
        <target variable="cdl:getVariable
          ('detectedincreaseDone','','')"/>
    </copy>
   </assign>
</workunit>
```

  The *workunit*'s *guard* element establishes the condition, which has to be fulfilled to perform the workunit activities, this element allows us to define pre-conditions. Post-conditions and invariants can be introduced by appending a workunit with the condition as a guard at the end of the normal workunit flow. In order to define a condition we use XPath and XML Schema expressions.
- Protocol: sequence, non-deterministic choice, external choice, iterations.
  - A sequence of activities is modeled in WS-CDL using the ordering structure *sequence*.

  - A non-deterministic choice is implemented in WS-CDL using the ordering structure *choice*. The WS-CDL standard says that when two or more activities are specified here, only one of these is selected and the other ones are disabled. It is assumed that the selection criteria for those activities are non-observable.

```
<choice>
     Activity-Notation+
</choice>
```

  - External choice. This element can be implemented in WS-CDL using the ordering structure *workunit*, since it allows us to establish conditions to execute the corresponding activity.
  - Iteration. In WS-CDL, we can furthermore use the workunit *repeat* to implement repetition.
- Time aspects: lower bounds, upper bounds, explicit clocks, reset and stop operations are handled by using XPath and XML Schema. Specifically, we use the XML Schema notation to specify the time aspects as follows:
  - Explicit clocks are introduced by *xs:time*.
  - Lower and upper bounds are specified inside a workunit guard, where XML Schema uses two operations to delimit the time: *op:time-less-than* and *op:time-greater-than*. We can also use the *hasDeadlinePassed* operation, which is defined in the WS-CDL specification to manage timing.
  - Reset. In WS-CDL we reset the clock using an *assign* activity, which creates or changes the variable defined by the target element using the expression defined by the source element (in the same role).
  - Stop. In order to model that a clock is stopped, we can capture the value of the time, of this specific instant, in a clock variable and then, when we want to initiate the time again, we can use the clock variable to continue from this point. We use two *assign* activities to capture and change the time value.
  - Synchronizations. The *interaction* WS-CDL element defines how the parties in a web services are synchronized. The *optional* exchange element allows information to be exchanged during an interaction. The attribute name is used to specify a name for this exchange element.

```
<interation name="The demand management system
             sends increase in power demand to
             the productivity system"
  operation= = "sendIncreasing"
   channelVariable="Demand2ProductivityC">
   <description type="description">
    Sending the necessary increase of demand
   </description>
  <participate
         relationshipType= "DemandProductivity"
         fromRole="DemandRoleType"
         toRole="ProductivityRoleType" />
  <exchange name= "CalculatedIncerasing"
     informationType="Increase_demandType"
        action="request">
  </exchange>
  <timeout
    time-to-complete= "cdl:minor(cdl:getVariable
     ('tns:Clock1','','1:00')">?
</interaction>
```

D. Contract Aspects in WS-BPEL

BPEL is a programming language to specify the behavior of a participant in a choreography. Choreography is concerned with describing the message interchanges between participants. In like manner as in WS-CDL, we present the WS-BPEL contract aspects below:

- Interface. In WS-BPEL, the services with which a business process interacts are modeled as *partnerLinks*. Each *partnerLink* is characterized by a *partnerLinkType*, which characterizes the conversational relationship between two services. It defines the roles played by each of the services in the conversation and specifies the *portType* provided by each service to receive messages within the context of the conversation. These *portTypes* are defined in the WSDL document, and each role specifies exactly one WSDL *portType*.

  In order to utilize operations via a *partnerLink*, the binding and communication data, including *endpoint references (EPR)*, for the *partnerLink* must be available. The fundamental use of endpoint references is to serve as the mechanism for dynamic communication of port-specific data for services.

  An example fragment of a *partnerLink* is:

```
<partnerLinks>
<partnerLink name="productivity">
partnerLinkType="as:productivityDemandMSLT"
   myRole="DemandMS"
   partnerRole="productivity" />
</partnerLinks>
```

  The endpoint references syntax is:

```
<service-ref reference-scheme="http://example.org">
  <foo:barEPR xmlns:foo="http://example.org">
   ... </foo:barEPR>
</service-ref>
```

- Functional Specification: preconditions, postconditions and invariants.

  WS-BPEL uses expressions to implement the functional part of a web service contract. WS-BPEL uses several types of expressions, as follows:
  - Boolean expressions. These expressions can appear inside a transition, a join, a while, and an if condition.
  - Deadline expressions. The WS-BPEL elements that use these expressions are until-expressions of onAlarm and wait.
  - Duration expressions. These appear in the *for* expression of *onAlarm* and *wait*, and the *repeatEvery* expression of *onAlarm*.
  - Unsigned Integer expressions combined with *startCounterValue*, *finalCounterValue*, and the branches in a *forEach*.
  - General expressions inside assign activities.
- Protocol: sequence, non-deterministic choice, external choice, iterations.
  - A sequence of activities is modeled by the *sequence* structured activity. It contains one or more activities that are performed sequentially, in the lexical order in which they appear.

An example of an activity in a Productivity process is given as a sequence as follows:

```
<sequence>
 <if
 bpel:getVariableProperty('x','time:level')==0>
   <then>
  <!-Process productivity (invoke) - ->
   <assign>
<copy>
<from partnerLink="productivityMS"
endpointReference="myRole" />
<to>&increaseData.productivityMSRef </to>
</copy>
 </assign>
 <invoke name="increaseDemand"
          partnerLink="productivity"
          portType="as:productivityPT"
          operation="process"
          inputVariable="increaseData">
   <correlations>
   <correlation set="increaseIdentification"
   </correlations>
 </invoke>
 </if>
</sequence>
```

Both non-deterministic choice and external choice are expressed in WS-BPEL by means of *pick* activities, which waits for the occurrence of an event and then executes the activity associated with that event. When several events occur simultaneously, an implementation dependent choice is made. Thus in an analysis, the choice must be modeled as non-deterministic.

- Conditional. WS-BPEL contains a conventional conditional statement as well. Some (for instance if) is shown in the code fragment below.
- Iteration. In WS-BPEL we can use the *while* and the *repeatUntil* activities, to model iteration.

```
<while>
   <condition>
      $numberWindTurbine < 10
   </condition>
   <scope>
     ...
   </scope>
</while>

<repeatUntil standard-attributes>
    standard-elements
    activity
    <condition expressionLanguage="anyURI"?>
       ... bool-expr ...
    </condition>
</repeatUntil>
```

- Time aspects: lower bounds, upper bounds, explicit clocks, reset and stop operations are specified as in WS-CDL using XPath and XML Schema.
  - Explicit clocks, lower and upper bounds are defined using XML Scheme notations, as explained before.
  - Reset. In WS-BPEL we can reset the clock using an *assign* activity, which copies data from one variable to another.

```
<assign validate="yes|no"? standard-attributes>
   standard-elements
     (<copy keepSrcElementName="yes|no"?>
     from-spec
        to-spec
     </copy> |
     <extensibleAssign>
       ...assign-element-of-other-namespace...
     </extensibleAssign>) +
</assign>
```

  - Stop. In order to model that a clock is stopped in WS-BPEL we do as in WS-CDL.
  - Synchronizations are implemented in WS-BPEL using a *flow* activity, which provides concurrency and synchronization. A *flow* completes when all of the activities enclosed by it have completed.

```
<flow standard-attributes>
    standard-elements
    <links>?
        <link name="NCName">+
</links>
        activity+
</flow>
```

III. Analyzing Web Service Contracts

Having described all the elements of specifications, we now present the translation to automata. In order to perform this translation, we note that WS-CDL and WS-BPEL are XML based languages for describing Web Services. The timed automata formalism we use is UppAal [7]; and it is represented by another XML document, thus, the translation has been developed with XSLT [8], XML Style sheets Language for Transformation, which is a language for transforming XML documents into other XML documents.

Figure 3 shows how the translation works: we have created some XSL style sheets, where we use XSLT instructions to extract the information from the WS-CDL document, and then the UppAal document is automatically generated. This document can be opened with the UppAal tool, and thus, we can use the model-checker of UppAal to verify some properties of interest. The tool can also run simulations of the model. We have also created some XSL style sheets to perform the same translation for WS-BPEL documents.

For the two aspects we can check the following.

General Properties: We check the absence of deadlock for the CDL and for the BPEL, we check the property that the system is able to progress from start to termination; in UppAal:

$A[]not\ deadlock$

If there are enough available turbines to fulfill the increase of demand, then the Productivity Management system will send the order of turning on some of them to the Wind Turbine management system:

$A[]\ WindTurbineMS.AvailableT \rightarrow ProductivityMS.OrderTurnOn$

Satisfiability: Here we check for a BPEL property that the methods can be executed satisfying the contracts or generating the exceptions. For instance, if the demand system always send a message to the productivity system, when it detects an increase in the power demand (the message increase_demand). Also, the Wind Turbine Management system always sends the number of available turbines on Productivity Management system's demand.

This is represented in UppAal as follows:

$A[]\ ProductivityMS.NuTurbines \rightarrow WindTurbineMS.CalculateTA$

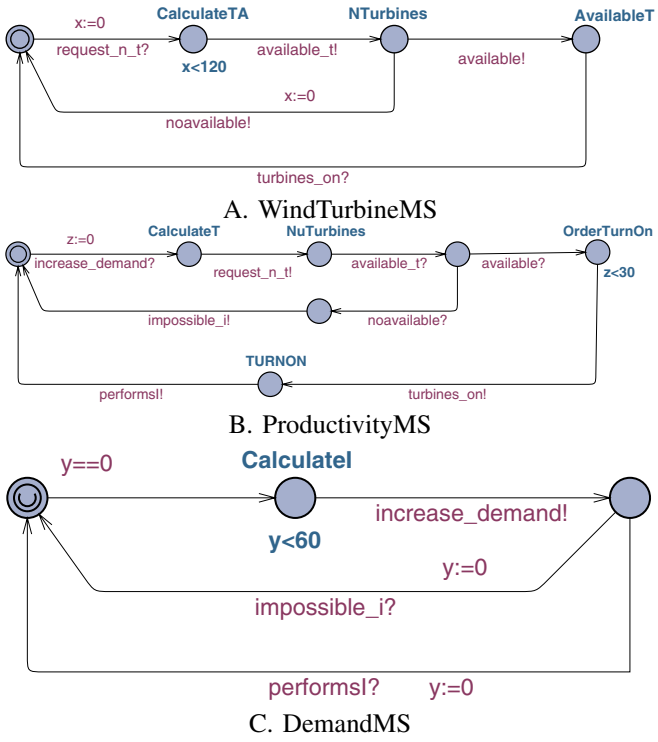

Fig. 3. Aero-Electric Management System modeled in UppAal

Overall Consistency: To check that the two individually derived models are consistent, we use bi-simulation. A bi-simulation is an equivalence relation between state transition systems, associating systems which behave in the same way in the sense that one system simulates the other and vice-versa. The automata generated from the two contract aspects specification systems (WS-CDL, WS-BPEL) are bi-similar in the following aspects:

- they both accept the same operation sequence; since the WS-CDL specified the protocols, while WS-BPEL contains the operation names but with more information.
- they also accept the same message sequence. Thus, the state that receives the message (for instance $increase\_demand$ in the example in Figure 3) is followed by a state that sends the message ($request\_n\_t$) in both automata. The automaton from WS-BPEL may contain some internal states.

Application specific: This form of checking is closer to systematic analysis of a design by some review process, for instance Software Reviews, Code Inspections, and other proactive management processes whose purpose is to eliminate or to find and remove errors in product design as early as possible.

IV. Related work

Web Service contracts is attracting a lot of attention and several researchers propose various approaches and frameworks toward specification and analysis. For instance [9], [10], [11] looks at it in a theoretical dimension, whereas [12], [13] propose a language for contracts. All these points to the fact that there is an important need for contracts to be specified and analyzed.

An earlier treatment of contracts in an object-oriented paradigm is Design by Contract [4]. Similar treatment concerning components is found in [14]. Here, the functional specification is achieved through assertions; which consists of preconditions, post-conditions and invariants. The framework in [15] takes a pragmatic approach at code level where the assertions are part of the language. We agree that these functional specifications are important in order to specify a formal agreement between a service provider and its clients. Thus expressing what a client should do in order to make a service request and what the provider will do in return.

Among the related work of Web Service contracts is [16]. It proposes to visualize contracts by graph transformation rules. Apart from expressing contracts in terms of pre- and post-conditions of operations together with invariants, they introduced the notions of provided and required contracts. With this, they use the provided contracts to create the test cases and test oracles whereas the required interfaces are used to drive the simulation. We like their treatment of functional specifications, but it needs to be supplemented with other aspects, and one may gain something by investigating model checking as a supplement to testing.

A different quantitative aspect is researched in [17], [18], [19]. The Web Service Level Agreement (WSLA) framework [17] is targeted at defining and monitoring SLAs for Web Services. WSLA enables service customers and providers to unambiguously define the agreed performance characteristics and the way to evaluate and measure them. We want to mention here that WSLA complements Web Service Definition Language (WSDL) [20], [21], an XML grammar that describes the capabilities of Web services through its interface descriptions. It serves as contracts between service provider and service requestor, but its treatment of functional behavior is limited.

A remarking difference between this work and the above mentioned contributions which mainly focus on either only the functional side of a contract or only the behavioral side of a contract is that a multi view (functional, behavioral) of a web service contract and a set of tools are proposed for the analysis while ensuring consistency.

V. Conclusion

We have presented an approach for the analysis of web service contracts which uses model checking as its prime tool. The analysis is kept manageable by separating contract aspects and analyzing them individually. The price we pay for this aspect oriented analysis is a check for consistency between the individually derived models. However, this check by setting up a bi-simulation between automata can perhaps be automated, because the configurations of the two automata are systematically related through naming conventions and similarities in the WS-CDL and WS-BPEL constructs.

In the current contribution, we demonstrate the approach using timed automata as used in the UppAal tool [7], but in other contexts [22] we have experimented with using JML [23] for the functional aspects. The ideas are illustrated with an example specification of a Wind Turbine Management System which consists of three major components (with their services).

We have not touched on verification of timing aspects, although this initiated this work [24], [25]. Thus the use of UppAal is to some extent a practical decision. We feel that it is well justified for the kinds of analyses that we discuss, because they are concerned with checking the properties of the service as such. For checking implementation conformance, it may not be ideal, and a translation to JML may be much more useful, in particular since Java may be an underlying implementation language, and JML is a formal specification language tailored to Java.

Its basic use is thus the formal specification of the behavior of Java program modules. This direction is, however, not the main line of our investigation. The immediate work facing us is to streamline the tool fragments developed for these experiments, and in particular to make true the claim that the bi-simulation can be integrated in a more automated analysis process. It is well known that model checking has its limits, and investigations are also being done of theorem proving approaches [26] which may be more suitable for implementation conformance checking.

ACKNOWLEDGMENT

The second author is funded by the Nordunet3 Project "Contract-Oriented Software Development for Internet Services".

REFERENCES

[1] Y. Lafon and N. Mitra, "SOAP Version 1.2 Part 0: Primer (Second Edition)," W3C, W3C Recommendation, Apr. 2007, http://www.w3.org/TR/2007/REC-soap12-part0-20070427/.

[2] S. Seely, *SOAP: Cross Platform Web Service Development Using XML*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001, foreword By-Kent Sharkey.

[3] A. Karmarkar, M. Gudgin, M. Hadley, Y. Lafon, J.-J. Moreau, H. F. Nielsen, and N. Mendelsohn, "SOAP version 1.2 part 1: Messaging framework (second edition)," W3C, W3C Recommendation, Apr. 2007, http://www.w3.org/TR/2007/REC-soap12-part1-20070427/.

[4] B. Meyer, *Object-oriented software construction (2nd ed.)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1997.

[5] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, *BPEL4WS, Business Process Execution Language for Web Services Version 1.1*, IBM, 2003. [Online]. Available: http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf

[6] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon, "Web services choreography description language version 1.0," W3C, W3C Working Draft, Dec. 2004, http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/.

[7] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi, "UPPAAL in 1995," in *Tools and Algorithms for Construction and Analysis of Systems*, 1996, pp. 431–434. [Online]. Available: citeseer.ist.psu.edu/article/bengtsson96uppaal.html

[8] J. Clark, "XSL Transformations (XSLT) Version 1.0," W3C, Tech. Rep. REC-xml-19980210, 1998, http://www.w3.org/TR/xslt. [Online]. Available: citeseer.nj.nec.com/bray98extensible.html

[9] G. Castagna, N. Gesbert, and L. Padovani, "A theory of contracts for web services," in *PLAN-X '07, 5th ACM-SIGPLAN Workshop on Programming Language Technologies for XML*, jan 2007.

[10] S. Carpineti, G. Castagna, C. Laneve, and L. Padovani, "A formal account of contracts for Web Services," in *WS-FM, 3rd Int.l Workshop on Web Services and Formal Methods*, ser. LNCS, no. 4184. Springer, 2006, pp. 148–162.

[11] H. Davulcu, M. Kifer, and I. V. Ramakrishnan, "CTR-S: A Logic for Specifying Contracts in Semantic Web Services," in *Proceedings of WWW2004*, May 2004, pp. 144–153.

[12] D. Reeves, B. Grosof, M. Wellman, and H. Chan, "Toward a declarative language for negotiating executable contracts," in *In Proc. AAAI-99*, 1999.

[13] C. Prisacariu and G. Schneider, "A formal language for electronic contracts," in *9th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'07)*, ser. Lecture Notes in Computer Science, M. Bonsangue and E. B. Johnsen, Eds., vol. 4468. Springer, June 2007, pp. 174–189.

[14] A. Beugnard, J.-M. Jezequel, N. Plouzeau, and D. Watkins, "Making Components Contract Aware," *Computer*, vol. 32, no. 7, pp. 38–45, 1999.

[15] B. Meyer, *Eiffel: the language*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1992.

[16] R. Heckel and M. Lohmann, "Towards contract-based testing of web services," 2004.

[17] A. Keller and H. Ludwig, "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services," *Journal of Network and Systems Management*, vol. 11, no. 1, pp. 57–81, March 2003.

[18] "Web Services Agreement Specification (WS-Agreement)," https://forge.gridforum.org/projects/graap-wg/document/WS-AgreementSpecification/en/7, 2004.

[19] "Web Services Architecture," W3C Working Group Note, www.w3.org/TR/ws-arch/, Feb 2004.

[20] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, *Web Services Description Language (WSDL) 1.1*, 1st ed., W3C, March 2001, URL: http://www.w3c.org/TR/wsdl.

[21] D. Booth and C. K. Liu, "Web services description language (WSDL) version 2.0 part 0: Primer," W3C, Candidate Recommendation, March 2006.

[22] GI-Dagstuhl, "Modelling contest: Common component modelling example (cocome)." [Online]. Available: http://agrausch.informatik.uni-kl.de/CoCoME

[23] J. Leavens, "JML's rich, inherited specification for behavioural subtypes," in *Proc. 8th International Conference on Formal Engineering Methods (ICFEM06)*, ser. LNCS, vol. 4260. Springer, 2006.

[24] G. Diaz, J. J. Pardo, M. E. Cambronero, V. Valero, and F. Cuartero, "Verification of Web Services with Timed Automata," in *Proceedings of First International Workshop on Automated Specification and Verification of Web Sites*, vol. 157. Springer Verlags Electronics Notes in Theoretical Computer Science series, 2005, pp. 19–34.

[25] G. Diaz, 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 *IEEE International Conference on Internet and Web Applications and Services ICIW'06*, vol. 1257.

[26] P. Giambiagi, O. Owe, A. Ravn, and G. Schneider, "Language-based support for service oriented architectures: Future directions," in *Proceedings of ICSOFT'06*, Setúbal, Portugal, September 2006.