Web Service Composition Approaches: From Industrial Standards to Formal Methods

Maurice ter Beek Antonio Bucchiarone Stefania Gnesi
Istituto di Scienza e Tecnologie dell'Informazione, Via G. Moruzzi 1, 56124 Pisa, Italy
Email: {maurice.terbeek,antonio.bucchiarone,stefania.gnesi}@isti.cnr.it

***Abstract*— Composition of web services is much studied to support business-to-business and enterprise application integration in e-Commerce. Current web service composition approaches range from practical languages aspiring to become standards (like BPEL, WS-CDL, OWL-S and WSMO) to theoretical models (like automata, Petri nets and process algebras). In this paper we compare these approaches w.r.t. a selected set of characteristics (like trust, security and performance) and we advocate the use of formal models, and their tool support, to increase one's confidence in web service compositions. This paper can assist web service composition designers and developers to deliver lasting solutions, in concordance with the technology's critical needs.**

I. Introduction

Web services (WSs) are interacting computer applications running on different platforms, managed by different organizations. Current research studies how to specify them (in a formal and expressive enough language), how to (automatically) compose them, how to discover them (on the Internet) and how to ensure their correctness. We focus on WS composition.

Several organizations are developing languages for WS composition, the most important ones are Business Process Execution Language for Web Services BPEL [1] and the Web Services Choreography Description Language WS-CDL [2]. Many of these languages however have only a limited ability to support automatic WS composition, mostly due to the absence of semantic representations of the WSs available on the Internet. In response to these limitations several solutions, among which the Web Ontology Language for Web Services OWL-S [3] and the Web Service Modeling Ontology WSMO [4], were proposed by the Semantic Web community and others.

In this paper we first describe and compare these approaches to WS composition w.r.t. a selected set of main characteristics to assess their quality. We then survey the increasing use of formal methods (mainly state-action models like automata and Petri nets or process models like the π-calculus) and tools to formally specify, compose and verify WS composition, and compare also these w.r.t. the set of characteristics. Finally, we discuss the expected advantage of using formal methods—in particular their tool support—to perform appropriate mathematical analyses to increase confidence in WS compositions. Our aim is to provide a reference for WS composition designers and developers willing to use formal methods and tools.

II. WS Composition Approaches

A main feature of WSs is the reuse mechanism to build new applications, which often need to be defined out of finer-grained subtasks that are likely available as WSs. Composition rules describe how to compose coherent global services. In particular, they specify the order in which, and the conditions under which, WSs may be invoked. We distinguish *syntactic* (XML-based) and *semantic* (ontology-based) WS composition.

A. Syntactic WS Composition

In the field of syntactic WS composition there are currently two main approaches. The first approach, referred to as WS *orchestration*, combines available WSs by adding a central coordinator (the orchestrator) that is responsible for invoking and combining the single subactivities. The second approach, referred to as WS *choreography*, instead does not assume a central coordinator but rather defines complex tasks via the definition of the conversation that should be undertaken by each participant; the overall activity is then achieved as the composition of peer-to-peer interactions among the collaborating WSs. While several proposals exist for orchestration languages, the most important one being BPEL, choreography languages are still in a preliminary stage of definition.

BPEL4WS: this XML-based language was designed to enable the coordination and composition of a set of WSs. It is based on the Web Services Description Language WSDL [5], which is basically an interface description language for WS providers. BPEL is a behavioral extension of WSDL using a workflow-based approach. It expresses relationships between multiple invocations by means of control and data flow links and it employs a distributed concurrent computation model with variables. A main construct to model the flow of WSs is a *process*, which is a net-based concurrent description connecting *activities* that send/receive messages to/from external WS providers. Each WS provider can be seen as a *port* of a particular *port type*, which has an appropriate WSDL description. A *partner link* specifies which activity is linked to a particular WS provider of the port.

WS-CDL: this XML-based specification language is targeted at composing interoperable, long-running, peer-to-peer collaborations between WS participants with different roles, as defined by a choreography description. Its most important element is the INTERACTION activity. An interaction describes an information exchange between parties, with a focus on the receiver. It consists of three main parts, corresponding to the *participants* involved, the *information* being exchanged and the *channel* over which to exchange the information. Exception handling and compensations are supported through

so-called *exception* and *finalizer* work units. Messages that are exchanged between participants are modeled with variables and tokens, whose type can be specified in XML schema or in WSDL. Channels are used to specify how and where message exchanges can take place. Synchronization among activities can be achieved via a *work unit*, which defines the guard condition that must be fulfilled in order to continue an activity.

Contrary to BPEL, WS-CDL describes a global view of the observable behavior of message exchanges of all WSs, intended for abstract process specification (independent of the platform or programming language used to implement the WSs). WS-CDL thus complements languages like BPEL, in which such behavior is defined from the viewpoint of one WS.

B. Semantic WS Composition

Current WS technologies address only the syntactic aspects of WSs and thus provide a set of rigid WSs that cannot adapt to a changing environment without human intervention. The vision underlying semantic WSs [6] is to describe the various aspects of WSs by using explicit, machine-understandable semantics, and as such automate all stages of the WS lifecycle.

The *Semantic Web* [7] provides a process-level description of WSs which, in addition to functional information, models the pre- and postconditions of processes so that the evolution of the domain can be logically inferred. It relies on ontologies to formalize the domain concepts that are shared among WSs. The Internet is seen as a globally linked database in which web pages are marked with semantic annotations. Given this infrastructure, powerful applications can be written that use the annotations and suitable inference engines to automatically discover, execute, compose and interoperate WSs. These great potential benefits have led to major research activity, both in industry and academia, with aiming to realize semantic WSs.

We consider two main initiatives. OWL-S [8] is an effort to define an ontology for the semantic markup of WSs, intended to enable the automation of WS discovery, invocation, composition, interoperation and execution monitoring by providing appropriate semantic descriptions of WSs. The WS Modeling Ontology WSMO [4] is an effort to create an ontology to describe various aspects related to semantic WSs, aiming to solve the integration problem. Both initiatives have as goal to provide a standard for the semantic description of WSs.

OWL-S: defines a WS ontology with four main elements. The SERVICE concept serves as an organizational point of reference for declaring WSs. Every WS is declared by creating an SERVICE instance. It links the remaining three elements of a WS through properties like PRESENTS, DESCRIBEDBY and SUPPORTS. The SERVICE PROFILE describes what a WS does at a high level, describing its functionality and non-functional properties, which is used to locate WSs based on their semantic description. Both the WSs offered by a provider and the WSs desired by a requester are described. The *service model* describes how a WS achieves its functionality, including a detailed description of its constituent processes (if any) as a *process model*. The *service grounding*, finally, describes how to use a WS (i.e. how clients can actually invoke it).

WSMO: defines a model to describe semantic WSs, based on the conceptual design set up in the WS Modeling Framework WSMF [9]. The latter distinguishes four elements: ontologies, WSs, goals and mediators. WSMO inherits these elements and further refines and extends them as follows. *Ontologies* are a key element, since they provide (domain-specific) terminologies to describe the other elements. They moreover link machine and human terminologies by formal semantics. *WSs* use the standard web-based protocols to exchange and combine data in new ways. They are described from three different perspectives: non-functional properties, functionality and behavior. *Goals* specify the objectives of a client when consulting a WS, i.e. the functionalities a WS should provide from the user perspective. *Mediators*, finally, aim to overcome the mismatches appearing between the different elements constituting a WSMO description. Their existence allows one to link possibly heterogeneous resources.

In addition to these core elements, WSMO introduces a set of core non-functional properties that are defined globally and that can be used by all its modeling elements. WSMO is furthermore accompanied by a formal language, the WS Modeling Language WSML [4], which allows one to write annotations of WSs according to the conceptual model, and by an execution environment WSMX [10] for the dynamic discovery, selection, mediation, and invocation of WSs.

A first significant difference between the above initiatives is that OWL-S does not separate what the user wants from what the WS provides. The service profile of a WS (such as its name, a human-readable description and contact information) is not explicitly based on standard metadata specification. WSMO recommends the use of widely-accepted vocabularies (like the Dublin Core [11]). Another difference is that non-functional properties can be expressed in any WSMO element, whereas in OWL-S this is restricted to the service profile. Furthermore, in OWL-S the service model does not clearly distinguish between choreography and orchestration; it is not based on any formal model, even if some work on defining the formal semantics of OWL-S processes has been done. OWL-S defines only one service model per WS, so there is only one way to interact with the WS. In WSMO, on the other hand, choreography and orchestration are specified in the interface of a WS description. A choreography describes the external visible behavior of the WS and an orchestration describes how other WSs are composed in order to achieve the required functionality of the WS. Since it is expected that there could be more than one way to interact with a particular WS, WSMO allows the definition of multiple interfaces for a single WS. To facilitate linkage of heterogeneous resources between one another, various kinds of mediation are required. Therefore WSMO explicitly defines mediators in the conceptual model. OWL-S does not explicitly do so: the underlying infrastructure is assumed to handle this. To summarize, OWL-S is more mature in certain aspects (like choreography), whereas WSMO provides a more complete conceptual model because it addresses aspects like goals and mediators.

III. Selection of WS Composition Characteristics

In this section we describe the set of characteristics w.r.t. which we compare the above approaches to WS composition and the formal approaches below. We believe that any WS composition approach should aim to support these characteristics, without pretending these to be all characteristics of importance. Our choice is based on related proposals [4], [11], [12] and in particular on the Ontology [13] developed in the EU project SENSORIA in which we are involved.

A. Connectivity

Reliable connectivity is needed to reason about WS interactions before composition, in order to guarantee the continuity of WS delivery after composition. Measures of interest include

Reliability: the ability to deliver responses continuously in time (service reliability) and the ability to correctly deliver messages between two endpoints (message reliability).

Accessibility: the percentage of responses per WS request.

Exception handling/Compensations: what happens in case of an error and how to undo the already completed activities.

In particular the latter two measures are receiving a lot of attention nowadays. WSs often make use of external WSs (not owned and thus not under control) and hence one must take into account that these external WSs can unexpectedly fail (not respond or worse). Since WSs are usually long-running processes that may take hours or weeks to complete, the ability to manage compensations of WS invocations is critical.

B. Correctness

The composition of WSs may lead to large and complex systems of concurrently executing WSs. An important aspect of such systems is the correctness of their (temporal) behavior.

Safety/Liveness: safety properties are assertions that some bad event never happens in the course of a computation, while liveness properties assert that some event does eventually happen. By verifying such properties, one obtains measures of correctness of (the composition of) WSs.

Security/Trust: the ability of a (compositon of) WSs to provide proper authentication, authorization, confidentiality and data encryption. This requires the means to validate the credentials of a WS client, to grant, deny and revoke access to WSs and to protect certain sensitive information or functionality of WSs. A key property of trust is the assurance that a WS (composition) will perform as expected despite possible environmental disruptions, human and operator errors, hostile attacks and design and implementation errors.

The behavioral properties a WS should satisfy are usually defined by a specification that precisely documents the desired behavior. Formal methods then provide rigorous mathematical means to guarantee a system's conformance to a specification.

C. Quality of Services (QoS)

There are several issues that determine the quality of WSs.

Accuracy: the error rate of a WS, measured as the number of errors generated by a WS in a certain time interval.

Availability: the probability that a WS is available at any given time, measured as the percentage of time a WS is available over an extended period of time.

Performance: the success rate of WS requests, measured as response time, throughput and latency. Response time is the guaranteed maximum time needed to complete a request, throughput is the number of completed requests over a period of time and latency is the time a WS needs to process a request.

IV. Comparing Standardization Approaches

Ideally, any approach to WS composition should satisfy the set of characteristics we compiled in the previous section. In this section we compare the approaches of Section II w.r.t. these characteristics. The outcome is summarized in Figure 1.

	Syntax-based		Semantics-based	
Characteristics	BPEL	WS-CDL	OWL-S	WSMO
Connectivity	+	+	+	+
Exception handling	+	+	±	+
Compensations	+	+	−	+
Correctness	−	−	−	−
QoS	±	±	+	+

Fig. 1. Comparing standardization approaches to WS composition.

A. Connectivity

All industrial approaches offer connectivity although the WSs themselves, at the lowest level, are modeled differently.

As said before, in BPEL the result of a WS composition is a process, its constituting WSs are partners, message exchanges are activities and a process interacts with external partner WSs through a WSDL interface. BPEL has several element groups that support reliability, like INVOKE and RECEIVE for synchronous and asynchronous calls, SEQUENCE and FLOW for sequential and parallel execution and SWITCH for logic control. In WS-CDL, the lowest level actions performed within a choreography are described by basic activities like INTERACTION, SEND and RECEIVE for the reliable exchange of information between the participants and PARTICIPATE to indicate a participant's role. OWL-S distinguishes the process types ATOMIC, SIMPLE and COMPOSITE, while constituent processes are specified using flow-control constructs like SEQUENCE, SPLIT and ITERATE. In WSMO, mediators can be used on the protocol level to communicate in a reliable way between WSs and on the process level to combine WSs.

Regarding exception handling, BPEL has a mechanism to catch and handle faults, similar to common programming languages like Java. We recall that in WS-CDL exception handling is supported through the exception and finalizer work units. WS-CDL handles a lot of errors (i.e. interaction failures, protocol, timeout errors, application failures, etc.) using the EXCEPTION BLOCK of a choreography [2]. WSMO explicitly models the error information of a WS in the INTERFACE description of the WS specification. OWL-S does not consider these details directly, but errors can be captured by using conditional outputs. This characterization of errors is not explicit,

as the definition of a conditional output does not necessarily imply that one of the possible outputs is an error [14].

Regarding compensations, WS-CDL uses the exception and finalizer work units. In BPEL, one may define a compensation handler to enable compensation activities if actions cannot be explicitly undone. OWL-S cannot be used to describe compensation operations. In fact, a goal of the OWL-S specification is "the ability to find out where in the process the request is and whether any unanticipated glitches have appeared" [15]. In WSMO, finally, when an invoked WS fails, the WS that invoked it may implement a strategy for compensation.

B. Correctness

Neither of the industrial approaches offer any direct support for the verification of WS compositions at design time, to evaluate in this way its correctness. In the next section we will see that this is the main issue where formal methods come into play. For instance, there are many attempts to formally capture and analyze the (temporal) behavior of BPEL [16]–[23].

C. QoS

The management of QoS when composing WSs requires a careful consideration of the QoS characteristics of the constituent WSs. BPEL and WS-CDL do not directly support the specification of most QoS measures. To enable the specification and monitoring of QoS aspects like accuracy, availability and performance, various approaches have been developed. Examples include IBM's Web Service Level Agreement framework WSLA [24] and HP's Open View Internet Services product. The latter describes a theoretic QoS parameter specification model and introduces SLAs for WSs in the form of WSML. In OWL-S and WSMO, on the other hand, QoS measures like accuracy and availability are specified as service parameters in the WS description definition, but the specification of metrics and guarantees is missing. Moreover, there is no way to specify functional relations between metrics and therefore quality-aware WS discovery is not feasible.

V. Formal Methods for WS Composition

WSs are typically designed to interact with other WSs to form larger applications. From a software engineering point of view, the construction of new WSs by composing existing WSs raises exciting perspectives, which can significantly impact the way future industrial applications will be developed. It also raises a number of challenges, however, one of them being the one of guaranteeing the correct interaction of independent, communicating software pieces. Due to the message-passing nature of WS interaction, many subtle errors might occur when several of them are put together (unreceived messages, deadlocks, incompatible behaviors, etc.). These problems are well known and recurrent in distributed applications, but they become even more critical in the world of WSs that is ruled by the long-term vision of "WSs used by WSs", rather than by humans, and in which interactions should—ideally—be as transparent and automatic as possible.

A major problem of the approaches we met in the previous section, namely the lack of software tools to verify the correctness of WS compositions, is at the same time the main advantage of most formal methods. In particular, formal methods and tools can be used to decide:

- Whether WSs are in some precise sense equivalent;
- Whether WSs satisfy certain desirable properties.

If one should discover that the composition of WSs does not match an abstract specification of what is desired, or that a main property is violated, this can be of help to correct a design or to diagnose bugs in a service. Recently several formal methods, most of them with a semantics based on transition systems (e.g. automata, Petri nets, process algebras), have been used to guarantee correct WS compositions.

Below we first present a selective overview of the use of well-known languages and models by the formal methods community to define the types of WS composition discussed in Section II. Subsequently we indicate which of these approaches have been used to formalize the WS composition characteristics selected in Section III.

A. Automata

Automata or *labeled transition systems* are a well-known model underlying formal system specifications. The intuitive way in which automata can model system behavior has lead to several automata-based specification models, like (variants of) I/O automata [25], timed automata [26] and team automata [27]. Their formal basis allows automatic tool support and—as a result—automata-based models are more and more used to formally describe, compose, and verify (compositions of) WSs. Below follow some exemplary approaches.

In [16] the authors introduce a framework to analyze and verify properties of WS compositions of BPEL processes communicating via asynchronous XML messages. This framework first translates the BPEL processes to a particular type of automata whose every transition is equipped with a guard in the form of an XPath [28] expression, after which these *guarded automata* are translated into Promela, the input language of the model checker SPIN [29]. Finally, SPIN can be used to verify whether WS compositions satisfy certain LTL properties. The authors are currently investigating to extend the framework to other WS specification languages like OWL-S. Also in [30] automata are used to translate BPEL processes to Promela.

In [17] a case study shows how descriptions of WSs written in BPEL/WS-CDL can be automatically translated to timed automata and subsequently be verified by the model checker UPPAAL [31]. In [18] the authors provide an encoding of BPEL processes into WS timed state transition systems, a formalism that is closely related to timed automata, and discuss a framework in which timed properties (both qualitative and quantitative) expressed in the duration calculus [32] can be model checked. In [33] a framework to automatically verify systems modelled in Orc is proposed. To this aim, the authors define a formal timed-automata semantics for Orc [34] expressions, which conforms to Orc's operational semantics. UPPAAL can then be used to model check Orc models.

B. Petri Nets

Petri nets are a framework to model concurrent systems [35]. Their main attraction is the natural way of identifying basic aspects of concurrent systems, both mathematically and conceptually. This has contributed greatly to the development of a rich theory of concurrent systems based on Petri nets. Their ease of conceptual modeling (largely due to an easy-to-understand graphical notation) has moreover made Petri nets the model of choice in many applications.

In fact, Petri nets are very popular in BPM-related fields due to the many process control flows they can capture [36]. In particular, the dead-path-elimination technique that is used in BPEL to bypass activities whose preconditions are not met, can be readily modeled in Petri nets. In [19] it is shown how to map all BPEL control-flow constructs into labeled Petri nets (thus including control flows for exception handling and compensations). This output can subsequently be used to verify BPEL processes by means of the open-source tools BPEL2PNML and WofBPEL (including reachability analysis). We now give some examples of such approaches.

In [37] the authors define the semantics of a relevant subset of DAML-S (now OWL-S) in terms of a first-order logic, namely the situation calculus [38]. Based on this semantics they describe WS compositions in a Petri-net-based formalism, complete with an operational semantics. They discuss the implementation of a tool to describe and automatically verify composition of WSs. In [39] the authors introduce a Petri-net-based algebra to compose WSs, based on control flows, and show how to use it for performance analysis. In [20] a Petri-net-based design and verification framework for WS composition is proposed, which can be used to visualize, create and verify existing BPEL processes. The authors still need to develop a graphical interface, with a Petri-net view and a BPEL view, to assist the creation of WS compositions. In [40] a Petri-net-based architectural description language, in which WS-oriented systems can be modeled and analyzed in an automatic way, is introduced and a small case study is presented. In order to deal with real-life applications and to eliminate manual translation errors, the authors are currently developing an automatic translation engine from WSDL to their language. In [21] a complete and formal Petri-net semantics for BPEL is presented, thus including exception handling and compensations. Furthermore, the authors present their BPEL2PN parser which can automatically transform BPEL processes into Petri nets. As a result, a variety of Petri-net verification tools are applicable to automatically analyze BPEL processes. Yet another framework for modeling and analyzing BPEL processes by means of Petri nets is presented in [22]. In [41] Orc is translated into colored Petri nets, which is a generalization of Petri nets that can deal with recursion and data handling. The authors extend their framework in [42] to deal with QoS aspects in a sound way.

C. Process Algebras

Like Petri nets, *process algebras* (PAs) are precise and well-studied formalisms that allow the automatic verification of certain behavioral properties. They come with a rich theory on bisimulation analysis, i.e. to establish whether two processes have equivalent behaviors. Such analyses are useful to establish whether one WS can substitute another WS in a composition or to verify the redundancy of a WS. The π-*calculus* [43] is a PA that has inspired modern WS composition languages like BPEL. As with Petri nets, the rationale behind using the π-calculus to describe processes lies in the advantages that a formal model with a rich theory provides for the automatic verification of properties of the behavior of models expressed in such a model. From a compositional perspective, the π-calculus offers constructs to compose activities in terms of sequential, parallel, and conditional execution, combinations of which can lead to compositions of arbitrary complexity. We now give some examples of process-algebraic approaches to specify and verify WS compositions.

In [44] the authors advocate the use of PAs to describe, compose and verify WSs, with a particular focus on their interactions. Therefore they present a case study that uses CCS [45] to specify and compose WSs as processes, and the Concurrency Workbench [46] to validate properties like correct WS composition. To be of use in real-life applications one needs to use more advanced calculi than CCS (e.g. the π-calculus) in order to consider also issues like the exchange of data during WS interactions and dynamic WS compositions. In fact, in [23] a two-way mapping is defined between BPEL and the more expressive PA LOTOS [47]. An advantage of the translation is the inclusion of compensations and exception handling, thus permitting the verification of temporal properties with the CADP [48] model-checking toolbox.

D. Comparison w.r.t. WS Composition Characteristics

In Figure 2 we compare the formal methods surveyed so far according to their ability to deal with the characteristics of Section III. The entries correspond directly to the articles in which example uses of the respective formal method can be found that satisify the specific characteristic under scrutiny.

	Semantic models		
Characteristics	Automata	Petri nets	Process algebras
Connectivity	[16, 17]	[20, 21, 37, 39, 40, 41]	[23, 44]
Exception handling/ Compensations	[18, 30]	[19, 21, 36, 41]	[23]
Correctness	[16, 17, 18, 30, 33]	[20, 21, 22, 37, 39]	[23, 44]
QoS	[16, 18, 33]	[19, 37, 42]	[23, 44]

Fig. 2. Comparing standardization approaches to WS composition.

We see that not many of the formal methods we consider deal with connectivity in a satisfactory way, even though a growing lot of them deal with exception handling and compensations. As said before, their solid mathematical basis does make formal methods very well suitable to verify the (behavioral) correctness properties under consideration. In fact, most of the considered formal methods have the advantage that they are accompanied by tools that allow one to simulate and verify the behavior of one's model *at design time*, thus enabling the detection and correction of errors as early as possible. As such, these formal approaches can be used to

increase the correctness of (compositions of) WSs. It should be noted that in industry various tools are being developed to support the specification and composition of WSs. Examples include IBM's WebSphere Choreographer [49] and Oracle's BPEL Process Manager [50]. For verification, however, formal methods are the means to use. Finally, also QoS issues like performance (analysis) are rather well supported by formal methods, again largely due to their solid mathematical basis and by means of their tool support. Obviously, some quantitative information from the actual use of WSs is needed for proper performance analyses.

VI. Conclusion

While there exist several papers that compare and analyze WS composition languages [51], [52], these comparisons are conducted almost at the micro level, focusing on specific language structures and control patterns. We instead provide a general overview: Seven exemplary approaches to WS composition are compared against a set of characteristics that any approach should aim to support to facilitate WS composition.

The main problems with most practical approaches to WS composition are the verification of (behavioral) correctness of WS compositions and the (quantitative) analysis of QoS aspects. We hope to have convinced the reader that this is where formal methods can be of use. Due to the solid theoretical basis of all the formal methods considered in this paper, the tool support that comes with them allows one to simulate and verify the behavior of one's model at design time, thus enabling the detection and correction of errors as early as possible and in any case *before implementation*. Consequently, these approaches help increase the correctness of WSs.

Acknowledgment

The presented research was supported by the Italian project TOCAI.IT and by the EU project Sensoria.

Antonio Bucchiarone is also supported by the IMT Graduate School of Lucca, Piazza S. Ponziano 6, 55100 Lucca, Italy.

References

[1] BPEL 1.1. [Online]. Available: ibm.com/developerworks/library/ws-bpel
[2] N. Kavantzas, D. Burdett, and G. Ritzinger. (2004) WSCDL v1.0. [Online]. Available: http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/
[3] A. Ankolekar et al., "DAML-S: Web Service Description for the Semantic Web," in *Proc. ISWC'02*, ser. LNCS, vol. 2342. Springer, 2002, pp. 348–363.
[4] WSMO working group. [Online]. Available: http://www.wsmo.org
[5] WSDL v1.1. (2001). [Online]. Available: http://www.w3.org/TR/wsdl
[6] S. A. McIlraith, T. C. Son, and H. Zeng, "Semantic Web Services," *IEEE Intelligent Systems*, vol. 16, no. 2, pp. 46–53, 2001.
[7] W3C. Semantic Web. [Online]. Available: http://www.w3.org/sw/
[8] D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. [Online]. Available: http://www.w3.org/TR/owl-features/
[9] D. Fensel and C. Bussler, "The Web Service Modeling Framework WSMF," *Electr. Commerce Res. Apps.*, vol. 1, no. 2, pp. 113–137, 2002.
[10] WSMX working group. [Online]. Available: http://www.wsmx.org
[11] S. Weibel et al. (1998) Dublin Core Metadata for Resource Discovery. IETF 2413. [Online]. Available: http://www.ietf.org/rfc/rfc2413.txt
[12] WSs Architecture. [Online]. Available: http://www.w3.org/TR/ws-arch/
[13] Sensoria, "Prototype language for service modelling: ontology for SOAs presented through structured natural language (deliverable 1.1a)," 2006.
[14] L. Zhang and M. Jeckle, "Conceptual Comparison of WSMO and OWL-S," in *ECOWS'04*, ser. LNCS, vol. 3250. Springer, 2004, pp. 254–269.
[15] D. Biswas, "Compensation in the World of Web Services Composition," in *Proc. SWSWPC'04*, ser. LNCS, vol. 3387. Springer, 2004, pp. 69–80.
[16] X. Fu, T. Bultan, and J. Su, "Analysis of Interacting BPEL Web Services," in *Proc. WWW'04*. ACM Press, 2004, pp. 621–630.
[17] G. Díaz et al., "Automatic Translation of WS-CDL Choreographies to Timed Automata," in *Proc. WS-FM'05*, ser. LNCS, vol. 3670. Springer, 2005, pp. 230–242.
[18] R. Kazmiakin, P. Pandya, and M. Pistore, "Timed Modelling and Analysis in Web Service Compositions," in *ARES*. IEEE, 2006, pp. 840–846.
[19] C. Ouyang et al., "Formal Semantics and Analysis of Control Flow in WS-BPEL," BPM Center, Tech. Rep. BPM-05-15, 2005.
[20] X. Yi and K. Kochut, "A CP-nets-based Design and Verification Framework for Web Services Composition," in *[53]*, 2004, pp. 756–760.
[21] S. Hinz, K. Schmidt, and C. Stahl, "Transforming BPEL to Petri Nets," in *Proc. BPM'05*, ser. LNCS, vol. 3649. Springer, 2005, pp. 220–235.
[22] A. Martens, "Analyzing Web Service Based Business Processes," in *Proc. FASE'05*, ser. LNCS, vol. 3442. Springer, 2005, pp. 19–33.
[23] A. Ferrara, "Web Services: a Process Algebra Approach," in *Proc. ICSOC'04*. ACM Press, 2004, pp. 242–251.
[24] WSLA v1.0. [Online]. Available: http://www.research.ibm.com/wsla/
[25] D. Kaynar et al., *Theory of Timed I/O Automata*. Morgan Claypool, 2006.
[26] R. Alur and D. L. Dill, "A Theory of Timed Automata," *TCS*, vol. 126, no. 2, pp. 183–235, 1994.
[27] M. H. ter Beek et al., "Synchronizations in Team Automata for Groupware Systems," *CSCW*, vol. 12, no. 1, pp. 21–69, 2003.
[28] J. Clark et al. (1999) XML Path Language XPath v1.0. W3C. [Online]. Available: http://www.w3.org/TR/xpath
[29] G. J. Holzmann, *The SPIN Model Checker*. Addison Wesley, 2003.
[30] J. A. Fisteus, L. Sánchez Fernández, and C. D. Kloos, "Formal Verification of BPEL4WS Business Collaborations," in *Proc. EC-Web'04*, ser. LNCS, vol. 3182. Springer, 2004, pp. 76–85.
[31] K. G. Larsen, P. Pettersson, and W. Yi, "UPPAAL in a Nutshell," *Int. J. Software Tools for Technology Transfer*, vol. 1, pp. 134–152, 1997.
[32] Z. Chaochen, C. A. R. Hoare, and A. P. Ravn, "A Calculus of Durations," *IPL*, vol. 40, no. 5, pp. 269–276, 1991.
[33] J. Dong et al., "Verification of Computation Orchestration via Timed Automata," in *Proc. ICFEM'06*, ser. LNCS, 2006.
[34] W. Cook and J. Misra. Orc—An Orchestration Language v0.5. [Online]. Available: http://www.cs.utexas.edu/users/wcook/projects/orc/
[35] W. Reisig and G. Rozenberg, Eds., *Lectures on Petri Nets I: Basic Models & II: Applications*, ser. LNCS. Springer, 1998, vol. 1491-1492.
[36] B. Kiepusewski, A. ter Hofstede, and W. van der Aalst, "Fundamentals of Control Flow in Workflows," *Acta Inf.*, vol. 39, no. 3, pp. 143–209, 2003.
[37] S. Narayanan and S. McIlraith, "Simulation, Verification and Automated Composition of Web Services," in *WWW'02*. ACM, 2002, pp. 77–88.
[38] R. Reiter, *Knowledge in Action—Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.
[39] R. Hamadi and B. Benatallah, "A Petri Net-based Model for Web Service Composition," in *ADC'03*, ser. CRPIT, vol. 17, 2003, pp. 191–200.
[40] J. Zhang et al., "WS-Net: A Petri-net Based Specification Model for Web Services," in *[53]*, 2004, pp. 420–427.
[41] S. Rosario et al., "Net system semantics of Web Service Orchestrations modeled in Orc," IRISA, Tech. Rep. 1780, 2006.
[42] ——, "Foundations of web service orchestrations: functional and QoS aspects," in *Proc. ISOLA'06*, 2006.
[43] R. Milner, J. Parrow, and D. Walker, "A calculus of mobile processes I & II," *Information and Computation*, vol. 100, no. 1, pp. 1–77, 1992.
[44] G. Salaün, L. Bordeaux, and M. Schaerf, "Describing and Reasoning on Web Services using Process Algebra," in *[53]*, 2004, pp. 43–50.
[45] R. Milner, *Communication and Concurrency*. Prentice Hall, 1989.
[46] R. Cleaveland, T. Li, and S. Sims. (2000) Concurrency Workbench of the New Century v1.2. [Online]. Available: http://www.cs.sunysb.edu/~cwb/
[47] T. Bolognesi and E. Brinksma, "Introduction to the ISO Specification Language LOTOS," *Computer Networks*, vol. 14, pp. 25–59, 1987.
[48] J. Fernandez et al., "CADP: A Protocol Validation and Verification Toolbox," in *Proc. CAV*, ser. LNCS, vol. 1102. Springer, 1996, pp. 437–440.
[49] WebSphere. [Online]. Available: ibm.com/software/info1/websphere
[50] BPEL process manager. [Online]. Available: oracle.com/technology/bpel
[51] J. Mendling and M. Müller, "A Comparison of BPML and BPEL4WS," in *Proc. 1st Conf. Berliner XML-Tage*, 2003, pp. 305–316.
[52] P. Wohed et al., "Pattern-Based Analysis of BPEL4WS," Queensland University of Technology, Brisbane, Tech. Rep. FIT-TR-2002-04, 2002.
[53] H. Jain, L. Liu, and L. Zhang, Eds., *Proc. ICWS'04*. IEEE Press, 2004.