

# Formal Methods for Service Composition<sup>\*</sup>

Maurice H. ter Beek<sup>1</sup>, Antonio Bucchiarone<sup>2,\*\*</sup>, and Stefania Gnesi<sup>1</sup>

<sup>1</sup> Istituto di Scienza e Tecnologie dell'Informazione, CNR  
Via G. Moruzzi 1, 56124 Pisa, Italy

{maurice.terbeek,stefania.gnesi}@isti.cnr.it

<sup>2</sup> IMT Institute for Advanced Studies Lucca  
Piazza S. Ponziano 6, 55100 Lucca, Italy  
antonio.bucchiarone@imtlucca.it

**Abstract.** Current approaches to service composition range from industrial standards (like BPEL and OWL-S) to formal methods (like Petri nets and process algebras). In this paper, we survey several such approaches and compare them on the basis of a selected set of characteristics (like compensations, trust and performance). Our conclusion is that formal methods, including tool support, are ideal to assist designers and developers in their work since their use leads to increased confidence in the obtained service compositions.

## 1 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 service composition.

Several organizations are developing languages for service composition, the most important ones are the Business Process Execution Language 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 service composition, mostly due to the absence of semantic representations of the available services. The Semantic Web community and others have proposed several solutions to these limitations, among which the Web Ontology Language for Web Services OWL-S [3] and the Web Service Modeling Ontology WSMO [4].

In this paper, we first describe and compare these approaches to service 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 Petri nets or process models like the  $\pi$ -calculus) to formally specify, compose

---

<sup>\*</sup> This work has been partially funded by the EU project SENSORIA (IST-2005-016004) and by the Italian project TOCAI.IT.

<sup>\*\*</sup> A. Bucchiarone is also supported by a Marie Curie host fellowship for transfer of knowledge (FP6-2002-014525) and he collaborates with Nokia Siemens Networks, Lisboa, Portugal.

and verify service 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 service compositions. Our aim is to provide a reference for service composition designers and developers willing to use formal methods.

## 2 Service Composition Approaches

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

### 2.1 Syntactic Service Composition

The field of syntactic service composition distinguishes two main approaches. The first approach, referred to as service *orchestration*, combines available services by adding a central coordinator (the orchestrator) that is responsible for invoking and combining the single subactivities. The second approach, referred to as service *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 services. 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 services. 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 services is a *process*, which is a net-based concurrent description connecting *activities* that send/receive messages to/from external WS providers. Each 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 port provider.

**WS-CDL** This XML-based specification language is targeted at composing interoperable, long-running, peer-to-peer collaborations between service 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 to continue an activity.

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

## 2.2 Semantic Service Composition

Current WS technologies address only the syntactic aspects of services and thus provide a set of rigid WSs that cannot adapt to a changing environment without human intervention. The vision underlying semantic web services [6] is to describe the various aspects of WSs by using explicit, machine-understandable semantics, and as such automate all stages of the service 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 services. These great potential benefits have led to major research activities, both in industry and academia, with the aim of realizing semantic web services.

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 service 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 web services, aiming to solve the integration problem. Both initiatives have as goal to provide a standard for the semantic description of services.

**OWL-S** OWL-S defines a service ontology with four main elements. The SERVICE concept serves as an organizational point of reference for declaring services. Every service is declared by creating an SERVICE instance. It links the remaining three elements of a service through properties like PRESENTS, DESCRIBEDBY

and SUPPORTS. The SERVICE PROFILE describes what a service does at a high level, describing its functionality and non-functional properties, which is used to locate services based on their semantic description. Both the service offered by a provider and those desired by a requester are described. The *service model* describes how a service 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 service (i.e. how clients can actually invoke it).

**WSMO** WSMO defines a model to describe semantic web services, based on the conceptual design set up in the WS Modeling Framework WSMF [9]. The latter distinguishes four elements: ontologies, services, 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. *Services* 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 service, i.e. the functionalities a service should provide from the user perspective. *Mediators*, finally, aim to overcome the mismatches appearing between the different elements constituting a WSMO description. They allow 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 moreover 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 services.

A significant difference between the above initiatives is that OWL-S does not separate what the user wants from what the service provides. The service profile (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 service, so there is only one way to interact with the service.

In WSMO, on the other hand, choreography and orchestration are specified in the interface of a service description. A choreography describes the external visible behavior of the service and an orchestration describes how other services are composed in order to achieve the required functionality of the service. Since it is expected that there could be more than one way to interact with a particular service, WSMO allows the definition of multiple interfaces for a single service.

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.

### 3 Selection of Service Composition Characteristics

In this section, we describe the set of characteristics w.r.t. which we compare the above approaches to service composition and the formal approaches below. We believe that any service 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.

#### 3.1 Connectivity

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

**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 service 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. Services often make use of external services (not owned and thus not under control) and hence one must take into account that these external services can unexpectedly fail (not respond or worse). Since services are usually long-running processes that may take hours or weeks to complete, the ability to manage compensations of service invocations is critical.

#### 3.2 Correctness

Service composition may lead to large and complex systems of concurrently executing services. 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 a service (composition).

**Security/Trust:** The ability of a service (composition) 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 services and to protect certain sensitive information or service functionality. A key property of trust is the assurance that a service (composition) will perform as expected despite possible environmental disruptions, human and operator errors, hostile attacks and design and implementation errors.

The behavioral properties a service 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.

### 3.3 Quality of Services

There are several issues that determine the quality of services (QoS).

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

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

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

## 4 Comparing Standardization Approaches

Ideally, any approach to service composition should satisfy the set of characteristics we compiled in Section 3. In this section, we compare the approaches of Section 2 w.r.t. these characteristics. The outcome is summarized in Figure 1, in which + ( $\pm$ , -) means that the particular characteristic is (a little, not, resp.) offered by the particular approach.

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

**Fig. 1.** Comparing standardization approaches to service composition.

#### 4.1 Connectivity

All industrial approaches offer connectivity, each in a slightly different way. As said before, in BPEL the result of a service composition is a process, its constituting services are partners, message exchanges are activities and a process interacts with external partner services through a WSDL interface. BPEL has several element groups that support connectivity, like INVOKE and RECEIVE for reliable (synchronous and asynchronous) information exchange, 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, and constituent processes are specified by 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 services and on the process level to combine services.

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 service in the INTERFACE description of the service 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 compensa-

tion 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.

## 4.2 Correctness

Neither of the industrial approaches offer any direct support for the verification of service 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].

## 4.3 Quality of Services

The management of QoS when composing services requires a careful consideration of the QoS characteristics of the constituent services. 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 WSLA [24] and HP’s Open View Internet Services. The latter describes a theoretic QoS parameter specification model and introduces SLAs for services 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 service description, but the specification of metrics and guarantees is missing. Moreover, there is no way to specify functional relations between metrics and therefore quality-aware service discovery is not feasible.

# 5 Formal Methods for Service Composition

Services are typically designed to interact with other services to form larger applications. From a software engineering point of view, the construction of new services by composing existing services 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 service 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 service-oriented computing that is ruled by the long-term vision of “services used by services”, 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 service 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 services are in some precise sense equivalent and
- whether services satisfy certain desirable properties.

If one should discover that a service composition 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 service 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 service composition discussed in Section 2. Subsequently we indicate which of these approaches have been used to formalize the service composition characteristics selected in Section 3.

## 5.1 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 service (compositions). Below follow some exemplary approaches.

In [16] the authors introduce a framework to analyze and verify properties of service 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 service compositions satisfy certain LTL properties. The authors are currently investigating to extend the framework to other service 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 services 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 the orchestration language Orc [34] is proposed. To this aim, the authors define a formal timed-automata semantics for Orc expressions, which conforms to Orc's operational semantics. UPPAAL can then be used to model check Orc models.

## 5.2 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 service compositions. In [39] the authors introduce a Petri-net-based algebra to compose services, based on control flows, and show how to use it for performance analysis. In [20] a Petri-net-based design and verification framework for service 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 service-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.

### 5.3 Process Algebras

Like Petri nets, *process algebras* 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 service can substitute another service in a service composition or to verify the redundancy of a service.

The  $\pi$ -calculus [43] is a process algebra that has inspired modern service composition languages like BPEL. As with Petri nets, the rationale behind using the  $\pi$ -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  $\pi$ -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 service compositions.

In [44] the authors advocate the use of process algebras to describe, compose and verify services, with a particular focus on their interactions. Therefore they present a case study that uses CCS [45] to specify and compose services as processes, and the Concurrency Workbench [46] to validate properties like correct service composition. To be of use in real-life applications one needs to use more advanced calculi than CCS (e.g. the  $\pi$ -calculus) in order to consider also issues like the exchange of data during service interactions and dynamic service compositions. In fact, in [23] a two-way mapping is defined between BPEL and the more expressive process algebra 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.

## 6 Comparing Approaches in Formal Methods

In Figure 2 we compare the formal methods surveyed so far according to their ability to deal with the characteristics of Section 3. The entries correspond directly to the articles in which example uses of the respective formal method can be found that satisfy the specific characteristic under scrutiny.

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

	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 approaches to service composition in formal methods.

the detection and correction of errors as early as possible. As such, these formal approaches can be used to increase the correctness of service (compositions). It should be noted that in industry various tools are being developed to support the specification and composition of services. 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 services is needed for proper performance analyses.

## 7 Conclusion

While there exist several papers that compare and analyze service 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 service composition are compared against a set of characteristics that any approach should aim to support to facilitate service composition.

The main problems with most practical approaches to service composition are the verification of (behavioral) correctness of service 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 service (compositions). Hence formal methods, including tool support, are ideal to assist designers and developers in their work since their use leads to increased confidence in the obtained service compositions.

## References

1. BPEL v1.1: <http://www.ibm.com/developerworks/library/ws-bpel>.
2. WSCDL v1.0: <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. Number 2342 in LNCS, Springer (2002) 348–363
4. WSMO working group: <http://www.wsmo.org>.
5. WSDL v1.1: <http://www.w3.org/TR/wsdl>.
6. McIlraith, S.A., Son, T.C., Zeng, H.: Semantic Web Services. *IEEE Intelligent Systems* **16**(2) (2001) 46–53
7. Semantic Web: <http://www.w3.org/sw/>.
8. McGuinness, D.L., van Harmelen, F.: OWL Web Ontology Language Overview <http://www.w3.org/TR/owl-features/>.
9. Fensel, D., Bussler, C.: The Web Service Modeling Framework WSMF. *Electr. Commerce Res. Apps.* **1**(2) (2002) 113–137
10. WSMX working group: <http://www.wsmx.org>.
11. S. Weibel et al.: Dublin Core Metadata for Resource Discovery. IETF 2413 (1998) <http://www.ietf.org/rfc/rfc2413.txt>.
12. WSs Architecture: <http://www.w3.org/TR/ws-arch/>.
13. Sensoria Ontology: Prototype language for service modelling—ontology for SOAs presented through structured natural language (deliverable 1.1a) (2006)
14. Zhang, L., Jeckle, M.: Conceptual Comparison of WSMO and OWL-S. In: ECOWS'04. Number 3250 in LNCS, Springer (2004) 254–269
15. Biswas, D.: Compensation in the World of Web Services Composition. In: Proc. SWSWPC'04. Number 3387 in LNCS, Springer (2004) 69–80
16. Fu, X., Bultan, T., Su, J.: Analysis of Interacting BPEL Web Services. In: Proc. WWW'04, ACM Press (2004) 621–630
17. G. Díaz et al.: Automatic Translation of WS-CDL Choreographies to Timed Automata. In: Proc. WS-FM'05. Number 3670 in LNCS, Springer (2005) 230–242
18. Kazmiakin, R., Pandya, P., Pistore, M.: Timed Modelling and Analysis in Web Service Compositions. In: ARES, IEEE Press (2006) 840–846
19. C. Ouyang et al.: Formal Semantics and Analysis of Control Flow in WS-BPEL. Technical Report BPM-05-15, BPM Center (2005)
20. Yi, X., Kochut, K.: A CP-nets-based Design and Verification Framework for Web Services Composition. In: [53]. (2004) 756–760
21. Hinz, S., Schmidt, K., Stahl, C.: Transforming BPEL to Petri Nets. In: Proc. BPM'05. Number 3649 in LNCS, Springer (2005) 220–235
22. Martens, A.: Analyzing Web Service Based Business Processes. In: Proc. FASE'05. Number 3442 in LNCS, Springer (2005) 19–33
23. Ferrara, A.: Web Services: a Process Algebra Approach. In: Proc. ICSOC'04, ACM Press (2004) 242–251
24. WSLA v1.0: <http://www.research.ibm.com/wsla/>.
25. D. Kaynar et al.: Theory of Timed I/O Automata. Morgan Claypool (2006)
26. Alur, R., Dill, D.L.: A Theory of Timed Automata. *Theoret. Comp. Sci.* **126**(2) (1994) 183–235
27. M. H. ter Beek et al.: Synchronizations in Team Automata for Groupware Systems. *Computer Supported Cooperative Work* **12**(1) (2003) 21–69
28. XML Path Language XPath v1.0: <http://www.w3.org/TR/xpath>.
29. Holzmann, G.J.: The SPIN Model Checker. Addison Wesley (2003)

30. Fisteus, J.A., Sánchez Fernández, L., Kloos, C.D.: Formal Verification of BPEL4WS Business Collaborations. In: Proc. EC-Web'04. Number 3182 in LNCS, Springer (2004) 76–85
31. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a Nutshell. *Int. J. Software Tools for Technology Transfer* **1** (1997) 134–152
32. Chaochen, Z., Hoare, C.A.R., Ravn, A.P.: A Calculus of Durations. *Inform. Process. Lett.* **40**(5) (1991) 269–276
33. J. Dong et al.: Verification of Computation Orchestration via Timed Automata. In: Proc. ICFEM'06. LNCS (2006)
34. Orc v0.5: <http://www.cs.utexas.edu/users/wcook/projects/orc/>.
35. Reisig, W., Rozenberg, G., eds.: Lectures on Petri Nets I: Basic Models & II: Applications. Number 1491–1492 in LNCS. Springer (1998)
36. Kiepusewski, B., ter Hofstede, A., van der Aalst, W.: Fundamentals of Control Flow in Workflows. *Acta Inform.* **39**(3) (2003) 143–209
37. Narayanan, S., McIlraith, S.: Simulation, Verification and Automated Composition of Web Services. In: WWW'02, ACM Press (2002) 77–88
38. Reiter, R.: Knowledge in Action—Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press (2001)
39. Hamadi, R., Benatallah, B.: A Petri Net-based Model for Web Service Composition. In: ADC'03. Number 17 in CRPIT (2003) 191–200
40. J. Zhang et al.: WS-Net: A Petri-net Based Specification Model for Web Services. In: [53]. (2004) 420–427
41. S. Rosario et al.: Net system semantics of Web Service Orchestration modeled in Orc. Technical Report 1780, IRISA (2006)
42. S. Rosario et al.: Foundations of web service orchestrations: functional and QoS aspects. In: Proc. ISOLA'06. (2006)
43. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes I & II. *Inform. and Comput.* **100**(1) (1992) 1–77
44. Salaün, G., Bordeaux, L., Schaerf, M.: Describing and Reasoning on Web Services using Process Algebra. In: [53]. (2004) 43–50
45. Milner, R.: Communication and Concurrency. Prentice Hall (1989)
46. Cleaveland, R., Li, T., Sims, S.: Concurrency Workbench of the New Century v1.2 <http://www.cs.sunysb.edu/~cwb/>.
47. Bolognesi, T., Brinksma, E.: Introduction to the ISO Specification Language LOTOS. *Computer Networks* **14** (1987) 25–59
48. J. Fernandez et al.: CADP: A Protocol Validation and Verification Toolbox. In: Proc. CAV. Number 1102 in LNCS, Springer (1996) 437–440
49. WebSphere: <http://www.ibm.com/software/info1/websphere>.
50. BPEL process manager: <http://www.oracle.com/technology/bpel>.
51. Mendling, J., Müller, M.: A Comparison of BPML and BPEL4WS. In: Proc. 1st Conf. Berliner XML-Tage. (2003) 305–316
52. P. Wohed et al.: Pattern-Based Analysis of BPEL4WS. Technical Report FIT-TR-2002-04, Queensland University of Technology, Brisbane (2002)
53. Jain, H., Liu, L., Zhang, L., eds.: Proc. ICWS'04. IEEE Press (2004)