Formal Methods for Service Composition

Maurice H. ter Beek
Istituto di Scienza e Tecnologie dell'Informazione, CNR
Via G. Moruzzi 1, 56124 Pisa, Italy
maurice.terbeek@isti.cnr.it

Antonio Bucchiarone
IMT Lucca Institute for Advanced Studies
Piazza S. Ponziano 6, 55100 Lucca, Italy
antonio.bucchiarone@imtlucca.it

Stefania Gnesi
Istituto di Scienza e Tecnologie dell'Informazione, CNR
Via G. Moruzzi 1, 56124 Pisa, Italy
stefania.gnesi@isti.cnr.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 a number of such approaches and compare them with respect to a carefully selected set of characteristics (like exception handling and quality of services). We conclude that formal methods, often including tool support, are ideal to assist designers and developers because their use leads to increased confidence in the obtained compositions.**

***Index Terms*— Formal methods, Web Services, Service Composition.**

I. Introduction

WEB services (WSs) are distributed and independent computational elements that solve specific tasks, varying from simple requests to complex business processes, and that communicate using XML messages following the SOAP standard. 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 being the Web Services Business Process Execution Language WS-BPEL [9] (BPEL for short) and the Web Services Choreography Description Language WS-CDL [74]. 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 [77].

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, much in the style of [49]. We then survey the increasing use of formal methods (mainly state-action models like Petri nets or process models like the π-calculus) to formally specify, compose and verify service compositions, and also compare these w.r.t. the selected set of characteristics. Finally, we discuss the expected advantage of using formal methods—in particular their tool support—to perform appropriate mathematical analyses in order to increase one's confidence in service compositions. By doing so, we hope to provide a reference for service composition designers and developers willing to use formal methods and tools.

II. 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.

A. Syntactic Service Composition

We recognize two main approaches. The first approach, referred to as service *orchestration*, combines available services by adding a coordinator (the orchestrator) that is responsible for invoking and combining the single subactivities.

ISSN 1109-9305

The second approach, referred to as service *choreography*, instead does not assume a 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.

1) BPEL: 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 [75], 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 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.

2) 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, which 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 the information is being exchanged. 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 types 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.

WS-CDL describes a global view of the observable behavior of message exchanges of all participating services, 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 the orchestrator.

B. 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 [47] 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* [66] 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. For the Semantic Web, the Internet is viewed 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 [46] 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 [77] is an effort to create an ontology to describe various aspects related to semantic web services, aiming to solve the integration problem. The goal of both initiatives is to provide a standard for the semantic description of WSs.

1) OWL-S: This initiative 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 a *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 the service requested by a consumer 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).

2) WSMO: This initiative defines a model to describe semantic web services, based on the conceptual design set up in the WS Modeling Framework WSMF [18]. 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 [77], which allows one to write annotations of WSs according to the conceptual model, and by an execution environment WSMX [78] for the dynamic discovery, selection, mediation, and invocation of services.

A significant difference between the abovementioned 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 (such as the Dublin Core [72]). 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.

III. A Selection of Service Composition Characteristics

In this section, we describe the set of characteristics w.r.t. which we compare the abovementioned approaches to service composition and the formal approaches that will be described later. We believe that any service composition approach should aim to support these characteristics; we of course do not claim these to be all characteristics of importance. Our choice is based on the related proposals [72], [77], [79] and in particular on [49] and on the Ontology [68] developed in the EU project SENSORIA in which we are involved.

A. 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.

1) *Reliability:* The ability to deliver responses continuously in time (service reliability) and the ability to correctly deliver messages between two endpoints (message reliability).
2) *Accessibility:* The responsiveness towards service requests.
3) *Exception handling/Compensations:* What happens in case of an error and how to undo the already completed activities.

The latter two measures in particular are receiving a lot of attention nowadays. Services often make use of external or third-party services (not owned and thus not under control) and hence one must take into account the fact that the latter services can unexpectedly fail. 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.

B. Correctness

Service composition may lead to large, complex systems of concurrently executing services. An important

aspect of such systems is the correctness of their (temporal) behavior. The behavioral properties that 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.

1) *Safety/Liveness:* Safety properties are assertions that some undesired 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).
2) *Security/Trust:* The ability of a service (compositon) 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.

C. Quality of Services

There are several measures that determine the quality of service (QoS).

1) *Accuracy:* The error rate of a service, measured as the number of errors generated by a service in a certain time interval.
2) *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.
3) *Performance:* The quality 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.

IV. Comparing Standardization Approaches to Service Composition

Ideally, any approach to service 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 Fig. 1, in which $+$, $\pm$ and $-$ indicate that the particular characteristic is addressed, is hardly addressed or is not at all addressed, resp., by the particular approach.

A. 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 (e.g. interaction failures, protocol, timeout errors, application failures, etc.) using the *exception* block of a choreography [74]. 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 [82].

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" [6]. 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 service compositions at design time, to evaluate in this way its correctness. As a matter of fact, in the next section we will see that this is

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

Fig. 1. Comparing standardization approaches to service composition.

the main issue where formal methods (and tools) come into play. For instance, there have been many attempts to formally capture and analyze the (temporal) behavior of BPEL [14], [20], [24], [25], [29], [32]–[36], [38], [45], [56], [81].

C. 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 [76] 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.

V. 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 challenge to guarantee the correct interaction of independent, communicating pieces of software. Due to the message-passing nature of service interaction, many subtle errors might occur when several services 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, viz. 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

1) are in some precise sense equivalent and
2) 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 realize the approaches to service composition discussed in Section II. Subsequently we indicate which of these approaches have been used to address the service composition characteristics selected in Section III.

A. Automata

Automata or *Labeled Transition Systems* (LTSs) 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 [31], timed automata [2] and team automata [4].

Their formal basis allows automatic tool support and—as a result—automata-based models are increasingly being used to formally describe, compose, and verify service compositions. Below follow some exemplary approaches.

In [24], [25], 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 automaton whose every transition is equipped with a guard in the form of an XPath [80] expression, after which these guarded automata are translated into Promela, the input language of the model checker SPIN [30]. Also in [21] automata are used to translate BPEL processes to Promela. SPIN can then be used to verify whether service compositions satisfy certain properties expressed in the Linear Temporal Logic LTL. In [35], a (composition of) WSs is modeled by a (composition of) so-called state transition systems, after which LTL is again used to verify properties that such compositions should satisfy by means of the NuSMV model checker [12]. This approach is extended in [34], [36] by using so-called extended state transition systems and by performing (and comparing) verifications with both SPIN and NuSMV.

In [14], 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 [43]. This technique is consequently used in [15] to model check the time requirements of WS compositions specified in WS-CDL by using UPPAAL.

In [32], 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 DC [11] can be model checked. These analysis capabilities are consequently extended in [33] by using the quantified discrete-time duration calculus [57].

In [16], a framework to automatically verify systems modeled in the orchestration language Orc [54] 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.

In [52], services are modeled as I/O Automata, after which the supervision of service composition by a choreographer is verified by simulations over I/O automata, using a so-called universal service automaton.

B. Petri Nets

Petri nets are a framework to model concurrent systems [59], [60]. 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. Moreover, their ease of conceptual modeling (largely due to an easy-to-understand graphical notation) has made Petri nets the model of choice in many applications.

In fact, Petri nets are very popular in BPM-related fields due to the large variety of process control flows they can capture [37]. 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 [56], 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 is automated by the open-source tool BPEL2PNML, whose output can be used to verify BPEL processes by means of the open-source tool WofBPEL [55] (including reachability analysis). We now give some examples of such approaches.

In [53], the authors define the semantics of a relevant subset of DAML-S (now OWL-S) in terms of a first-order logic, viz. the situation calculus [61]. 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 [27], 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 [81], 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 [83], 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. To deal with real-life applications and to eliminate manual translation errors, the authors intend to develop an automatic translation engine from WSDL to their language.

In [29], 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 [45], including an effective algorithm to verify usability-based properties.

In [63], 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 [62] to deal with QoS aspects in a sound way.

Finally, a very recent Petri-net-based approach to formally model service composition can be found in [28]: It contains a rigorous account of the composition of services, represented as so-called *open Petri nets*, thus permitting correctness analyses.

C. Process Algebras

Like Petri nets, *process algebras* are precise and well-studied formalisms that allow the automatic verification of both functional and non-functional 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 *π-calculus* [51] is a process algebra that has inspired modern service 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 calculus with a rich theory provides for the automatic verification of behavioral properties expressed in such a calculus. 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 service compositions.

In [22], [23], a verification process is described and applied on several case studies. It is based on translating a BPEL model of a WS composition into a process calculus called Finite State Processes FSP [44] and subsequently into LTSs. Consequently, one can check the resulting FSP composition for correctness based on behavioral (trace) equivalence as well as model check the resulting LTS for safety and liveness properties.

In [65], 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 [50] to specify and compose services as processes, and the Concurrency Workbench [13] to validate properties like correct service composition. The Concurrency Workbench is also used in [38]: This time as a verification environment for the BPE-calculus, a small CCS-like language containing the main control flow constructs of BPEL while abstracting from many details (including fault and compensation handlers).

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 additional issues like the exchange of data during service interactions and dynamic service compositions. In [20], e.g., a two-way mapping is defined between BPEL and the more expressive process algebra LOTOS [7]. An advantage of the translation is the inclusion of compensations and exception handling, thus permitting the verification of temporal properties with the CADP [19] model-checking toolbox.

Finally, in [64] the authors introduce the BPEL2EC tool that can automatically transform BPEL processes into the Event Calculus EC [39]: It takes a WS composition as input and outputs its behavioral specification in the EC, after which such a specification is turned into a format that the theorem prover SPIKE [69] accepts to check for behavioral properties expressed in the EC.

1) SENSORIA *Calculi for Service Composition:* The remainder of this section is dedicated to recent process-algebraic approaches to model and verify service compositions (through orchestration and choreography) that have been proposed in the context of the EU project SENSORIA [67] in which we are involved. In particular, we briefly discuss a number of advanced process calculi that are equipped with basic primitives used to execute complex service applications (like long-running transactions).

We divide these calculi in two different categories. The first one consists of calculi that use an explicit notion of *session*, while the second one contains calculi that are based on *correlation information*. The resulting principal difference between these two categories is the way in which an orchestrator manages service communication. In the first category, the orchestrator must activate a set of different sessions that are related to the number of services in the orchestration. In the second one, the orchestrator runs only one process instance and a correlation information set is declared, which is used to allow services to interact with the right session.

a) Session-based Calculi: When a service is called, the calculi of this first category create a new session, i.e. a private bidirectional channel between caller and callee. In order to establish a communication, each side of the channel has a communication protocol installed. This category contains the Service Centered Calculus SCC [8], which was introduced as a combination of Orc's service-oriented characteristics and the name-passing communication mechanisms provided by the π-calculus. Essentially, SCC is a name-passing process calculus in which services can be created and invoked. A service invocation produces a new session in which interaction between clients and services can be modeled. Moreover, a session can be explicitly closed using the mechanism of process interruption (i.e. *close*).

SCC offers just a basic mechanism for service orchestration, which has been inspired by Orc's pipeline construct and which is expressive enough to encode most of Van der Aalst's orchestration patterns [1]. Nevertheless, it can be very useful to have some more sophisticated and flexible inter-session communication mechanism as a primitive construct in the language. For this reason, a number of variants of SCC were introduced (like SSCC [40] and CSCC [70], which make use of stream-oriented communication and message passing, resp.).

Regarding exception handling, these session-based calculi provide primitives to deal with the proper closure of a session, to install a termination handler and—in case of SCC—to signal the decision to close a session to partners.

b) Correlation-based Calculi: This second category contains SOCK: A Calculus for Service Oriented Computing [26] and COWS: A Calculus for Orchestration of Web Services [41]. Both calculi use *correlation information* to compose services. Correlation information (such as Internet cookies used by web sites) permits a flexible mechanism to manage relationships among collaborating partners. Recall from Section II that we distinguish two approaches to assemble services, viz. orchestration and choreography. In both SOCK and COWS, a choreography consists of three main elements: the roles, the initial state constraints and the conversations. Each role has a name and contains a set of variables and operations. An orchestration, on the other hand, consists of several orchestrators in parallel, each of which identifies a process by an identifier and memorizes the current values of its variables in a storage.

The principal difference between SOCK and COWS is that the latter is stateless as it bases correlation on values, while SOCK is stateful as it bases correlation on variables. Moreover, in SOCK the correlation information can change during run-time and a process might dynamically change partners (which is not permitted in COWS). Finally, while SOCK associates a new process with its state, in COWS it is added in parallel with the other processes.

Both these correlation-based calculi contain mechanisms to define fault and compensation handlers. In COWS one can force the immediate termination (by means of a *kill activity*) of concurrent threads, program ad-hoc fault and compensation handlers, and protect some sensitive activities. SOCK, on the other hand, distinguishes among fault handlers and termination/compensation handlers. In SOCK, it is possible to associate handlers to any portion of code using the *scope* construct and these handlers can be dynamically defined and updated. COWS is extended with time in [42], while a logical verification framework to check functional properties of service compositions specified with COWS is presented in [17].

VI. Comparing Approaches to Service Composition in Formal Methods

In Fig. 2 we compare the formal methods surveyed so far according to their ability to deal with the characteristics of Section III. The entries correspond to papers in the references. Each such paper appears only in the column of the formal method used in that paper, while it appears in a row if the respective characteristic is addressed in the paper.

Note that not many of the considered formal methods 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 of being accompanied by tools that allow simulation and verification of the behavior of a 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 service compositions.

In industry, various tools are being developed to support the specification and composition of services. Examples include IBM's WebSphere Choreographer [71] and Oracle's BPEL Process Manager [10]. For verification, however, formal methods are the means to use.

Finally, QoS issues like performance (analysis) are also rather well supported by formal methods, again largely due to their solid mathematical basis and their tool support. Obviously, some quantitative information from the actual use of services is needed for proper performance analyses.

VII. Conclusion

Several papers compare and analyze service composition languages [5], [48], [49], [58], [73]. Of these, [5] discusses service composition from the architectural viewpoint, both [48] and [73] contain comparisons conducted almost at the micro level, focusing on specific language structures and control patterns, and [58] hardly addresses service composition, focusing more on service-oriented computing as a whole. We instead provide a general overview, remaining closer to [49]: Seven exemplary approaches to service composition are compared against a carefully selected set of characteristics that any approach should aim to support to facilitate service composition.

	Semantic models		
Characteristics	Automata	Petri nets	Process algebras
Connectivity	[14, 15, 21, 24, 25, 34–36]	[27–29, 37, 53, 56, 62, 63, 81, 83]	[8, 20, 22, 23, 26, 38, 40, 41, 64, 65, 70]
Exception handling/ Compensations	[21, 32]	[29, 37, 45, 56, 63, 81]	[8, 20, 26, 40, 41, 70]
Correctness	[14–16, 21, 24, 25, 32–36, 52]	[27–29, 45, 53, 81]	[17, 20, 22, 23, 38, 64, 65]
QoS	[15, 16, 24, 25, 32, 33]	[53, 56, 62]	[20, 23, 42, 65]

Fig. 2. Comparing approaches to service composition in formal methods.

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 formal methods considered in this paper, the tool support that comes with them allows the simulation and verification of the behavior of a model at design time, thus enabling the detection and correction of errors as early as possible and in any case *before implementation*. As a result, these approaches help increase the correctness of service compositions. Hence formal methods, with their tool support, are ideal for assisting designers and developers, since their use leads to increased confidence in the obtained service compositions.

ACKNOWLEDGMENTS

The work presented here has been partially funded by the EU project SENSORIA (IST-2005-016004) and by the Italian project TOCAI.IT.

The authors are grateful to Roberto Bruni for his comments that have helped to improve Section V-C.1 and to the anonymous referees for their suggestions that have helped to improve the overall presentation.

REFERENCES

[1] W. M. P. van der Aalst et al., "Workflow patterns," *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003, cf. also www.workflowpatterns.com.

[2] R. Alur and D. L. Dill, "A Theory of Timed Automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.

[3] A. Ankolekar et al., "DAML-S: Web Service Description for the Semantic Web," in *Proc. ISWC'02*, ser. LNCS, no. 2342. Springer, 2002, pp. 348–363.

[4] M. H. ter Beek et al., "Synchronizations in Team Automata for Groupware Systems," *Computer Supported Cooperative Work*, vol. 12, no. 1, pp. 21–69, 2003.

[5] B. Benatallah et al., "Service Composition: Concepts, Techniques, Tools and Trends," in *Service-Oriented Software System Engineering—Challenges and Practices*, Z. Stojanovic and A. Dahanayake, Eds. Idea Group, 2005, pp. 48–66.

[6] D. Biswas, "Compensation in the World of Web Services Composition," in *Proc. SWSWPC'04*, ser. LNCS, no. 3387. Springer, 2004, pp. 69–80.

[7] T. Bolognesi and E. Brinksma, "Introduction to the ISO Specification Language LOTOS," *Computer Networks*, vol. 14, pp. 25–59, 1987.

[8] M. Boreale et al., "SCC: A Service Centered Calculus," in *Proc. WS-FM'06*, ser. LNCS, no. 4184. Springer, 2006, pp. 38–57.

[9] BPEL v1.1, www.ibm.com/developerworks/library/ws-bpel.

[10] BPEL process manager, www.oracle.com/technology/bpel.

[11] Z. Chaochen, C. A. R. Hoare, and A. P. Ravn, "A Calculus of Durations," *Information Processing Letters*, vol. 40, no. 5, pp. 269–276, 1991.

[12] A. Cimatti et al., "NUSMV: A New Symbolic Model Checker," *International Journal on Software Tools for Technology Transfer*, vol. 2, no. 4, pp. 410–425, 2000.

[13] R. Cleaveland, T. Li, and S. Sims, "Concurrency Workbench of the New Century v1.2."

[14] G. Díaz et al., "Automatic Translation of WS-CDL Choreographies to Timed Automata," in *Proc. WS-FM'05*, ser. LNCS, no. 3670. Springer, 2005, pp. 230–242.

[15] G. Díaz et al., "Analysis and Verification of Time Requirements Applied to the Web Services Composition," in *Proc. WS-FM'06*, ser. LNCS, no. 4184. Springer, 2006, pp. 178–192.

[16] J. Dong et al., "Verification of Computation Orchestration via Timed Automata," in *Proc. ICFEM'06*, ser. LNCS, 2006.

[17] A. Fantechi et al., "A model checking approach for verifying COWS specifications," in *Proc. FASE'08*, ser. LNCS. Springer, 2008.

[18] D. Fensel and C. Bussler, "The Web Service Modeling Framework WSMF," *Electronic Commerce Research and Applications*, vol. 1, no. 2, pp. 113–137, 2002.

[19] J. Fernandez et al., "CADP: A Protocol Validation and Verification Toolbox," in *Proc. CAV'96*, ser. LNCS, no. 1102. Springer, 1996, pp. 437–440.

[20] A. Ferrara, "Web Services: a Process Algebra Approach," in *Proc. ICSOC'04*. ACM, 2004, pp. 242–251.

[21] J. A. Fisteus, L. Sánchez Fernández, and C. D. Kloos, "Formal Verification of BPEL4WS Business Collaborations," in *Proc. EC-Web'04*, ser. LNCS, no. 3182. Springer, 2004, pp. 76–85.

[22] H. Foster et al., "Model-based Verification of Web Service Compositions," in *Proc. ASE'03*. IEEE, 2003, pp. 152–163.

[23] H. Foster et al., "Model checking service compositions under resource constraints," in *Proc. ESEC/FSE'07*. ACM, 2007, pp. 225–234.

[24] X. Fu, T. Bultan, and J. Su, "Analysis of Interacting BPEL Web Services," in *Proc. WWW'04*. ACM, 2004, pp. 621–630.

[25] X. Fu, T. Bultan, and J. Su, "WSAT: A Tool for Formal Analysis of Web Services," in *Proc. CAV'04*, ser. LNCS, no. 3114. Springer, 2004, pp. 510–514.

[26] C. Guidi et al., "SOCK: A Calculus for Service Oriented Computing," in *Proc. ICSOC'06*, ser. LNCS, no. 4294. Springer, 2006, pp. 327–338.

[27] R. Hamadi and B. Benatallah, "A Petri Net-based Model for Web Service Composition," in *Proc. ADC'03*, ser. CRPIT, no. 17, 2003, pp. 191–200.

[28] C. Hartonas, "Modeling Service Communication with Open Petri Nets," in *Proc. SEEFM'07*. South-East European Research Center, 2007, pp. 79–97.

[29] S. Hinz, K. Schmidt, and C. Stahl, "Transforming BPEL to Petri Nets," in *Proc. BPM'05*, ser. LNCS, no. 3649. Springer, 2005, pp. 220–235.

[30] G. J. Holzmann, *The SPIN Model Checker*. Addison Wesley, 2003.

[31] D. Kaynar et al., *Theory of Timed I/O Automata*. Morgan Claypool, 2006.

[32] R. Kazhamiakin, P. Pandya, and M. Pistore, "Timed Modelling and Analysis in Web Service Compositions," in *Proc. ARES'06*. IEEE, 2006, pp. 840–846.

[33] R. Kazhamiakin, P. Pandya, and M. Pistore, "Representation, Verification, and Computation of Timed Properties in Web Service Compositions," in *Proc. ICWS'06*. IEEE, 2006, pp. 497–504.

[34] R. Kazhamiakin and M. Pistore, "A Parametric Communication Model for the Verification of BPEL4WS Compositions," in *Proc. WS-FM'05*, ser. LNCS, no. 3670. Springer, 2005, pp. 318–332.

[35] R. Kazhamiakin and M. Pistore, "Static Verification of Control and Data in Web Service Compositions," in *Proc. ICWS'06*. IEEE, 2006, pp. 83–90.

[36] R. Kazhamiakin, M. Pistore, and L. Santuari, "Analysis of communication models in web service compositions," in *Proc. WWW'06*. ACM, 2006, pp. 267–276.

[37] B. Kiepusewski, A. ter Hofstede, and W. van der Aalst, "Fundamentals of Control Flow in Workflows," *Acta Informatica*, vol. 39, no. 3, pp. 143–209, 2003.

[38] M. Koshkina and F. van Breugel, "Modelling and Verifying Web Service Orchestration by means of the Concurrency Workbench," in *Proc. TAV-WEB'04*. ACM, 2004, pp. 1–10.

[39] R. A. Kowalski and M. J. Sergot, "A Logic-based Calculus of Events," *New Generation Computing*, vol. 4, no. 1, pp. 67–95, 1986.

[40] I. Lanese et al., "Disciplining Orchestration and Conversation in Service-Oriented Computing," in *Proc. SEFM'07*. IEEE, 2007, pp. 305–314.

[41] A. Lapadula, R. Pugliese, and F. Tiezzi, "A Calculus for Orchestration of Web Services," in *Proc. ESOP'07*, ser. LNCS, no. 4421. Springer, 2007, pp. 33–47.

[42] A. Lapadula, R. Pugliese, and F. Tiezzi, "C⏲WS: A timed service-oriented calculus," in *Proc. ICTAC'07*, ser. LNCS, vol. 4711. Springer, 2007, pp. 275–290.

[43] K. G. Larsen, P. Pettersson, and W. Yi, "UPPAAL in a Nutshell," *International Journal on Software Tools for Technology Transfer*, vol. 1, pp. 134–152, 1997.

[44] J. Magee and J. Kramer, *Concurrency—State Models and Java Programs*. John Wiley, 2006.

[45] A. Martens, "Analyzing Web Service Based Business Processes," in *Proc. FASE'05*, ser. LNCS, no. 3442. Springer, 2005, pp. 19–33.

[46] D. L. McGuinness and F. van Harmelen, "OWL Web Ontology Language Overview," www.w3.org/TR/owl-features.

[47] S. A. McIlraith, T. C. Son, and H. Zeng, "Semantic Web Services," *IEEE Intelligent Systems*, vol. 16, no. 2, pp. 46–53, 2001.

[48] J. Mendling and M. Müller, "A Comparison of BPML and BPEL4WS," in *Proc. 1st Conf. Berliner XML-Tage*, 2003, pp. 305–316.

[49] N. Milanovic and M. Malek, "Current Solutions for Web Service Composition," *IEEE Internet Computing*, vol. 8, no. 6, pp. 51–59, 2004.

[50] R. Milner, *Communication and Concurrency*. Prentice Hall, 1989.

[51] 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.

[52] S. Mitra, R. Kumar, and S. Basu, "Automated Choreographer Synthesis for Web Services Composition Using I/O Automata," in *Proc. ICWS'07*. IEEE, 2007, pp. 364–371.

[53] S. Narayanan and S. McIlraith, "Simulation, Verification and Automated Composition of Web Services," in *Proc. WWW'02*. ACM, 2002, pp. 77–88.

[54] Orc v0.5, www.cs.utexas.edu/users/wcook/projects/orc.

[55] C. Ouyang et al., "WofBPEL: A Tool for Automated Analysis of BPEL Processes," in *Proc. ICSOC'05*, ser. LNCS, no. 3826. Springer, 2005, pp. 484–489.

[56] C. Ouyang et al., "Formal semantics and analysis of control flow in WS-BPEL," *Science of Computer Programming*, vol. 67, pp. 162–198, 2007.

[57] P. K. Pandya, "Specifying and Deciding Qauntified Discrete-time Duration Calculus formulae using DCVALID," in *Proc. RTTOOLS'01*, 2001.

[58] M. P. Papazoglou et al., "Service-Oriented Computing: State of the Art and Research Challenges," *Computer*, vol. 40, no. 11, pp. 38–45, 2007.

[59] W. Reisig and G. Rozenberg, Eds., *Lectures on Petri Nets I: Basic Models*, ser. LNCS. Springer, 1998, no. 1491.

[60] W. Reisig and G. Rozenberg, Eds., *Lectures on Petri Nets II: Applications*, ser. LNCS. Springer, 1998, no. 1492.

[61] R. Reiter, *Knowledge in Action—Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT, 2001.

[62] S. Rosario et al., "Foundations for Web services Orchestrations: functional and QoS aspects," in *Proc. ISOLA'06*, 2006.

[63] S. Rosario et al., "Net system semantics of Web Service Orchestrations modeled in Orc," IRISA, Technical Report 1780, 2006.

[64] M. Rouached and C. Godart, "Requirements-driven Verification of WSBPEL Processes," in *Proc. ICWS'07*. IEEE, 2007, pp. 354–363.

[65] G. Salaün, L. Bordeaux, and M. Schaerf, "Describing and Reasoning on Web Services using Process Algebra," in *Proc. ICWS'04*. IEEE, 2004, pp. 43–50.

[66] Semantic Web, www.w3.org/sw.

[67] Sensoria, www.sensoria-ist.eu.

[68] Sensoria Ontology, "Prototype language for service modelling-ontology for SOAs presented through structured natural language (deliverable 1.1a)," 2006, available via [67].

[69] S. Stratulat, "A General Framework to Build Contextual Cover Set Induction Provers," *Journal of Symbolic Computation*, vol. 32, no. 4, pp. 403–445, 2001.

[70] H. T. Vieira, L. Caires, and J. C. Seco, "The Conversation Calculus: a Model of Service Oriented Computation," in *Proc. ESOP'08*, ser. LNCS. Springer, 2008.

[71] WebSphere, www.ibm.com/software/info1/websphere.

[72] S. Weibel et al., "Dublin Core Metadata for Resource Discovery. IETF 2413," 1998, www.ietf.org/rfc/rfc2413.txt.

[73] P. Wohed et al., "Pattern-Based Analysis of BPEL4WS," Queensland University of Technology, Brisbane, Technical Report FIT-TR-2002-04, 2002.

[74] WSCDL v1.0, www.w3.org/TR/2004/WD-ws-cdl-10-20040427.

[75] WSDL v1.1, www.w3.org/TR/wsdl.

[76] WSLA v1.0, www.research.ibm.com/wsla.

[77] WSMO working group, www.wsmo.org.

[78] WSMX working group, www.wsmx.org.

[79] WSs Architecture, www.w3.org/TR/ws-arch.

[80] XML Path Language XPath v1.0, www.w3.org/TR/xpath.

[81] X. Yi and K. Kochut, "A CP-nets-based Design and Verification Framework for Web Services Composition," in *Proc. ICWS'04*. IEEE, 2004, pp. 756–760.

[82] L. Zhang and M. Jeckle, "Conceptual Comparison of WSMO and OWL-S," in *Proc. ECOWS'04*, ser. LNCS, no. 3250. Springer, 2004, pp. 254–269.

[83] J. Zhang et al., "WS-Net: A Petri-net Based Specification Model for Web Services," in *Proc. ICWS'04*. IEEE, 2004, pp. 420–427.