A Survey on Service Composition Approaches: From Industrial Standards to Formal Methods*

Maurice H. ter Beek[1], Antonio Bucchiarone[1,2], and Stefania Gnesi[1]

[1] Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo" (ISTI–CNR)
Area della Ricerca CNR di Pisa, Via G. Moruzzi 1, 56124 Pisa, Italy
[maurice.terbeek,antonio.bucchiarone,stefania.gnesi]@isti.cnr.it
[2] IMT Graduate School, Via San Micheletto 3, 55100 Lucca, Italy

Abstract. Composition of services has received much interest to support business-to-business and enterprise applications integration. The business world has developed a number of XML-based standards to formalize the specification of web services, their composition and their execution. On the other hand, the semantic web community focuses on reasoning about web resources by explicitly declaring their preconditions and effects with terms defined precisely in ontologies. Current service composition approaches range from practical languages aspiring to become industrial standards (e.g. BPEL and OWL-S) to more theoretical models and languages (e.g. automata, Petri nets, and process algebras). In this paper we present a survey of existing proposals for service composition and compare them among each other with respect to some key requirements. We hope this helps service composition designers and developers to focus their efforts and to deliver lasting solutions, while at the same time addressing the technology's critical needs.

1 Introduction

Service-Oriented Computing (SOC) is an emerging paradigm for distributed computing and e-business processing that has its origins in object-oriented and component computing. Services are computational entities that are autonomous and heterogeneous (e.g. running on different platforms or owned by different organizations). Services are described using appropriate service description languages, published and discovered according to predefined protocols, and combined using an engine that coordinates the interactions among collaborating services. Web service technology is a widespread and accepted instantiation of SOC that should facilitate the integration of newly built and legacy applications both within and across organizational boundaries, avoiding difficulties due to different platforms, heterogeneous programming languages, etc.. Exploiting this kind of ubiquitous network fabric should result in an increased productivity and in a reduction of costs in business-to-business (B2B) processes [37].

* This work was partly supported by the EU project IST-3-016004-IP-09 SENSORIA (*Software Engineering for Service-Oriented Overlay Computers*).

Web services are distributed and independent pieces of code solving specific tasks and communicating with each other through the exchange of messages. They are built on XML-based open standards, promise the interoperability of a variety of applications running on heterogeneous platforms, and enable dynamic connections and the automation of business processes—both within and across enterprises—for enterprise application integration and B2B integration.

This raises the need for *web service composition* to provide the mechanism to fulfill the complexity of the execution of business processes. Several organizations are currently working on new service composition proposals. The most important existing languages are IBM's WSFL [44] and Microsoft's XLANG [52], which have converged into the Business Process Execution Language for Web Services (BPEL [16]). BPEL is now a working draft by the Organization for the Advancement of Structured Information Standards (OASIS [28]). Current web service technologies however fall short on their restricted capability to support static service composition. Their limit comes from the total absence of semantic representations of the services available on the Internet. In response to these limitations, a number of solutions have been proposed by the semantic web community, among which OWL-S [1].

Some well-known problems related to web services are how to specify them in a formal and expressive enough language, how to compose them (automatically), how to discover them through the web, and how to ensure their correctness. Mathematical techniques tailored for the specification and verification of systems are known as *formal methods*. This field of research cuts across many areas of computer science and comes with an impressive body of literature on numerous specification languages and verification tools. Formal methods are particularly well suited to address most of the aforementioned issues (e.g. composition and correctness). Recently a variety of concrete proposals to formally describe, compose and verify web services have emerged. The majority of these are based on state-action models (e.g. labelled transition systems, timed automata, and Petri nets) or process models (e.g. π-calculus and other calculi).

In this paper we present a survey of existing proposals for service composition and compare them among each other with respect to some key requirements. We hope this helps service composition designers and developers to focus their efforts and to deliver lasting solutions, while at the same time addressing the technology's critical needs. The paper is structured as follows. After this introduction, we present some approaches to service-oriented composition (e.g. BPEL and OWL-S) in Section 2. In Section 3 we move the attention to the formal methods that can be used for secure service composition. We introduce a selective overview of current methods: timed automata, Petri nets, and process algebras. We subsequently compare all these approaches with respect to some service-composition requirements in Section 4. Finally, Section 5 reports some conclusive considerations and some indications for future work.

2 Service-Oriented Composition: Current Approaches

Composition of services has received much interest to support B2B. The business world has developed a number of XML-based standards to formalize the specification of web services, their composition, and their execution. The semantic web community focuses instead on reasoning about web resources by explicitly declaring their preconditions and effects by means of terms defined precisely in ontologies. The objective of this section is to introduce the current service-composition approaches in both these worlds.

2.1 Web Services and Semantic Web Services

Web Services are self-contained, modular units of application logic which provide business functionality to other applications via an Internet connection. They support the interaction of business partners and their processes by providing a stateless model of "atomic" synchronous or asynchronous message exchanges. Web service interactions are characterized by two specification languages: the Simple Object Access Protocol (SOAP) [67] and the Web Services Definition Language (WSDL) [12]. The former is a platform- and language-independent communication protocol that defines an XML-based format for web services to exchange information over HTTP by using remote procedure calls. The latter is an XML-based language which defines the interface that a web service exhibits in order to be invoked by other services. WSDL thus provides a function-centric description of web services covering inputs, outputs, and exception handling.

The semantic web provides a process-level description of services which, in addition to functional information, models the preconditions and postconditions of the process so that the evolution of the domain can be logically inferred. It relies on ontologies to formalize the domain concepts which are shared among services. The semantic web efforts (see, e.g., [7, 66])—especially the recent trend towards *semantic web services* [50]—aim at fully automating all stages of the web services lifecycle. The semantic web considers the World Wide Web as a globally linked database where web pages are marked with semantic annotations. These annotations are assertions about web resources and their properties expressed in the Resource Description Format (RDF) [10]. Along with RDF one can use RDF Schema (RDFS) to express classes, properties, ranges, and documentation for resources and the OWL-S (formerly DAML-S [1]) ontology to represent further relationships and/or properties like equivalences, lists, and data types. With the semantic web infrastructure in place, practical and powerful applications can be written that use annotations and suitable inference engines to automatically discover, execute, compose, and interoperate web services.

Given the different information that is available to specify web services in the two worlds described above, we differentiate between *static* and *dynamic* service composition. In this paper we compare some of the approaches in these worlds with formal methods that can be used to guarantee their *secure* composition. This leads us to consider the division portrayed in Figure 1.

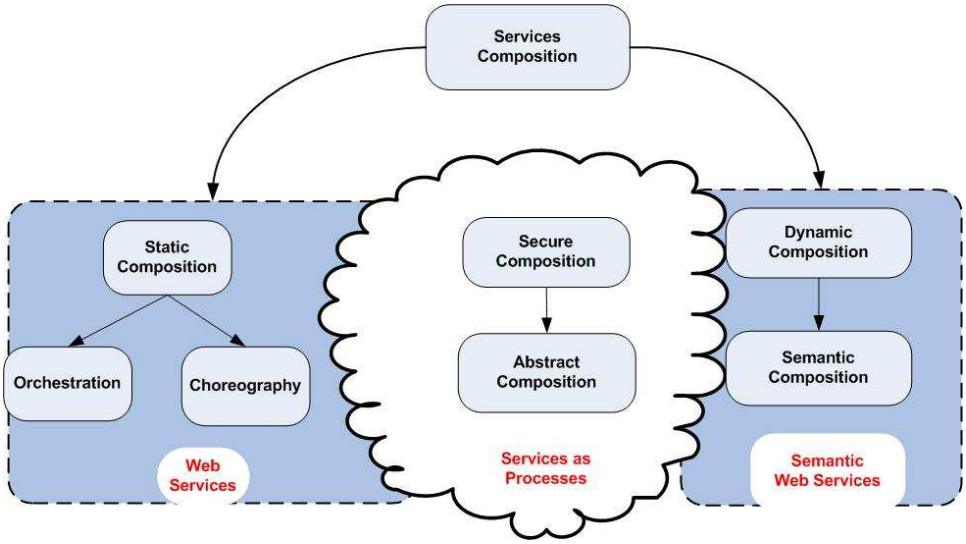

Fig. 1. Services Composition Models

2.2 Static Service Composition

A relevant feature for web services is the mechanism for their reuse when complex tasks are carried out. It is often the case that new processes need to be defined out of finer-grained subtasks that are likely available as web services. To this aim, extensions of the web service technology are considered. Composition rules describe how different services can be composed into a coherent global service. In particular, they specify the order in which services are invoked and the conditions under which a certain service may or may not be invoked.

Two main approaches are currently investigated for static service composition. The first approach, referred to as web service *orchestration*, combines available services by adding a central coordinator (the orchestrator) that is responsible for invoking and combining the single sub-activities. The second approach, referred to as web service *choreography*, does not assume the exploitation of a central coordinator but rather defines complex tasks via the definition of the conversation that should be undertaken by each participant. Following this approach, the overall activity is achieved as the composition of peer-to-peer interactions among the collaborating services. While several proposals exist for orchestration languages (e.g. BPML [34] and BPEL [16]), choreography languages are still in a preliminary stage of definition. An initial proposal—WS-CDL [38]—was issued by the World Wide Web Consortium (W3C) in December 2004.

BPEL is an XML-based language supporting process-oriented service composition [16, 21]. Originally developed by BEA, IBM, Microsoft, SAP, and Siebel, BPEL is currently being standardized by OASIS [28]. BPEL composition inter-

acts with a subset of web services to achieve a given task. The composition result is called a *process*, participating services are *partners*, and message exchanges (or intermediate result transformations) are called *activities*. A process interacts with external partner services through a WSDL interface.

BPEL has several element groups, the basic ones being process initiation (<PROCESS>), the definition of the services participating in the composition (<PARTNERLINK>), synchronous and asynchronous calls (<INVOKE>, <RECEIVE>), intermediate variables and results manipulation (<VARIABLE>, <ASSIGN>, <COPY>), exception handling (<SCOPE>, <FAULTHANDLERS>), sequential and parallel execution (<SEQUENCE>, <FLOW>), and logic control (<SWITCH>).

Researchers from IBM recently released BPELJ, a combination of BPEL and Java that allows developers to include Java code inside their BPEL code [17]. Developers can use BPEL with two additional specifications:

- Web Services-Coordination [18] coordinates the actions of web services when a consistent agreement must be reached on the service activities' outcome;
- Web Services-Transaction [19] defines the transactional behaviour of web services.

There are several BPEL orchestration server implementations for both J2EE and .NET platforms, including IBM's WebSphere [15], Oracle's BPEL process manager [20], Microsoft's BizTalk [52], OpenStorm [56], and Active BPEL [25].

The formal methods community has come up with a number of tools for the automatic translation of BPEL in automata- or Petri-net-based formalisms. We will come back to this in the next section.

2.3 Dynamic Service Composition

Web services are designed to provide interoperability between different applications. The platform- and language-independent interfaces of web services allow an easy integration of heterogeneous systems. Web languages like Universal Description, Discovery, and Integration (UDDI) [65], WSDL, and SOAP define standards for service discovery, description, and messaging protocols. These web service standards however do not deal with dynamic composition of existing services. Recent industrial initiatives to address this issue, like BPEL, focus on representing composition where information flow and the binding between services are known a priori.

A more challenging problem is to *dynamically* compose services. In particular, when a functionality that cannot be realized by the existing services is required, the existing services can be combined to fulfill the request. The dynamic composition of services requires the location of services based on their capabilities and the recognition of those services that can be matched to create a composition, as described in [47]. The full automation of this process is still the subject of ongoing research, but accomplishing this goal with a human controller as the decision mechanism can already be achieved. The main problem

for full automation of service composition is the gap between the concepts that people use and the data that computers interpret. This barrier can be overcome by using semantic web technologies. An example is OWL-S [48].

OWL-S (previously known as DAML-S) is a service ontology that enables automatic service discovery, invocation, composition, interoperation, and execution monitoring [1]. OWL-S models services using a three-way ontology:

- a service *profile* describes what the service requires from and gives to users;
- a service *model* specifies how the service works; and
- a service *grounding* gives information on how to use the service.

The process model is a service model subclass that describes a service in terms of inputs, outputs, preconditions, postconditions, and—if necessary—its own subprocesses. In the process model, one can describe composite processes together with their dependencies and their interactions. OWL-S distinguishes three types of processes: atomic, which have no subprocesses; simple, which are not directly invocable and are used as an abstraction element for either atomic or composite processes; and composite, which consist of subprocesses. Constituent processes are specified using flow-control constructs. SEQUENCE, SPLIT, SPLIT+JOIN, UNORDERED, CHOICE, IF-THEN-ELSE, ITERATE, and REPEAT-UNTIL.

There have been several proposals of methods to transfer OWL-S descriptions to Prolog [49] and to Petri-net-based models [55] in order to further analyze them. In the Prolog approach, the developer manually translates an OWL-S description, which allows one to find an adequate plan to compose web services for a target description. That is, for a given pool of available web services, it is possible to use logical inference rules to automate service allocation for the required task. In the Petri-net-based approach, an OWL-S description is automatically translated. Developers then use the result to automate tasks like simulation, validation, verification, composition, and performance analysis.

3 A Selective Overview of Existing Formal Methods to Secure Service Composition

In this section we describe a few well-known languages and models that have been used by the formal methods community to guarantee *secure* service compositions in the two approaches discussed in the previous section (cf. Figure 1). To begin with, we explain what is meant by secure service composition.

3.1 Secure Service Composition

Services are software applications to be used through a network via the exchange of messages. They are meant to be frequently reused and they are typically designed to interact with other services in order to form larger applications. From a software engineering point of view, the construction of new services by static or

dynamic composition of 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 web service interaction, many subtle errors might occur when several of them are put together (messages that are never received, deadlocks, incompatible behaviours, etc.). These problems are well known and recurrent in distributed applications. However, they become even more critical in the open-end world of services 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 as automatic as possible.

It is for the above reasons that formal methods should be used. The major advantage of using languages and models with a clear and formal semantics, is that this enables the use of automatic tools to verify whether a system matches its requirements and works properly. Specifically, formal methods and tools can be used to decide *i)* whether two services are in some precise sense equivalent and *ii)* wether a service satisfies certain desirable properties (e.g. the property that the system will never reach a certain unexpected state). Finding out that the composition of existing services does not match an abstract specification of what is desired, or that it violates a property which absolutely needs to hold, can help to correct a design or to diagnose bugs in an existing service. Very recently, several formal methods—most of them with a semantics based on transition systems (timed automata, Petri nets, process algebras, etc.)—have been suggested to guarantee secure service compositions. In the remainder of this section we will present a selective overview of some of these approaches.

3.2 Automata

Automata or *labelled transition systems* are a well-known model underlying formal specifications of systems. An automaton consists of a set of states, a set of actions, a set of labelled transitions between states, and a set of initial states. Labels represent actions and a transition's label indicates the action causing the transition from one state to another. The intuitive way in which an automaton can model a system's behaviour has lead to a variety of automata-based specification models such as Input/Output (I/O) automata and their many variants [46, 45, 39], timed automata [4, 3] and team automata [24, 5], to name but a few.

I/O automata were originally introduced to model distributed computations in asynchronous networks and as a means of constructing correctness proofs of distributed algorithms. Basically, an I/O automaton is an automaton whose set of actions is partitioned into input, output, and internal actions. A distinction is made between internal and external (input and output) actions used to communicate with the environment, which may consist of other I/O automata. I/O automata can be composed using a synchronous product construction yielding a new I/O automaton. Many variants of I/O automata were considered and the model is now widely used for describing reactive, distributed systems.

Team automata are an extension of I/O automata, originally introduced to model components of groupware systems and their interconnections. They were further developed as a formal model to provide a solid and general theoretical framework for the study of synchronization mechanisms in automata models. By dropping a number of the restrictions of I/O automata, team automata allow the flexible modelling of various kinds of collaboration in groupware systems: Team automata impose hardly any restrictions on the role of the actions in the various components and their composition is not based on an a priori fixed way of synchronizing their actions. This allows the definition of a wide variety of protocols for the interaction between a system and its environment.

Timed automata, finally, were introduced to model the behaviour of real-time systems in a formal way. They extend automata with timing constraints by using a finite number of real-valued clocks, which can be reset and whose values increase uniformly with time. At any moment in time, the value of a clock equals the time elapsed since the last time it was reset. These clocks can be used to guard the transition from one state (location) to another: A transition is enabled if and only if the timing constraint associated with it is satisfied by the current values of the clocks. One of the main attractions of timed automata is the well-developed automatic tool support: Model checkers like UPPAAL [43] and KRONOS [71], allow one to effectively verify timed automata models.

Automata-based models are more and more being used to formally describe, compose, and verify (compositions of) web services. Below follow some exemplary approaches, without claiming completeness.

In [29] the authors introduce a framework to analyze and verify properties of web service compositions of BPEL processes that communicate via asynchronous XML messages. Their 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 [35] expression, after which these guarded automata are translated into Promela, the input language of the model checker SPIN [33]. Finally, SPIN can be used to verify whether web service compositions satisfy certain LTL properties. The authors are currently investigating to extend their framework to other web service specification languages such as OWL-S.

In [22] a case study is presented that shows how descriptions of web services written in BPEL-WSCDL can be automatically translated to timed automata and subsequently be verified by UPPAAL. The authors are currently implementing this translation in a tool that should use UPPAAL as its engine. The development of this tool is of crucial importance for the authors' methodology to be embraced by industry.

In [40] the authors provide an encoding of BPEL processes into web service timed state transition systems, a formalism that is closely related to timed automata, and discuss a framework in which timed assumptions expressed in the duration calculus [11] can be model checked.

In [23] a framework to automatically verify systems that are modelled in Orc is proposed. To this aim, the authors define a formal timed-automata semantics for Orc [14] expressions, which confirms to Orc's operational semantics. Con-

sequently, UPPAAL can be used to model check Orc models. The approach is demonstrated through a small case study.

Team automata allow one to separately specify the components of a system, to describe their interactions, and to reuse the system as a component of a higher-level team automaton, thus supporting component-based system design in a natural way. Their main feature is a flexible technique for specifying coordination patterns among distributed systems, extending classical I/O automata. This makes team automata a promising model for the formal description of secure service compositions. This is the subject of ongoing research.

3.3 Petri Nets

Petri nets were introduced in [59] as a framework to model concurrent systems. Their main attraction is the natural way in which many basic aspects of concurrent systems are identified both mathematically and conceptually. This has contributed greatly to the development of a rich theory of concurrent systems based on Petri nets [60]. Their ease of conceptual modelling (largely due to an easy-to-understand graphical notation) has moreover made Petri nets the model of choice in many applications [61].

Actually "Petri net" is a generic name for a class of net-based models, consisting of an underlying structure (a net) together with rules describing its dynamics. Within a net one distinguishes places (representing local aspects of global states) and transitions (representing actions). Transitions (drawn as rectangles) are connected to places (circles) and places to transitions, by arcs (arrows). Hence a net is a bipartite directed graph. In some models, certain elements may be labelled. The dynamics of a net is given in the form of rules defining when (in which states) a transition can occur ("fire") and its effect on the current state if it occurs. It is fundamental to Petri nets that both the conditions allowing a transition to occur and its effect on the global state are local, in the sense that they only involve places in the immediate neighbourhood of (adjacent to) the transition.

Petri nets are very popular in BPM and related fields due to the variety of process control flows that they can capture [41, 69]. In particular, the dead-path-elimination technique that is used in BPEL to bypass activities whose preconditions are not met, can be readily modelled in Petri nets. In [57] it is shown how to map all BPEL control-flow constructs into labelled Petri nets (thus including control flows for exception handling and compensation).[3] 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, again without claiming completeness.

In [55] 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 [62]. Based on this semantics they describe web service compositions in a Petri-net-based

[3] This formalisation revealed several unambiguities in BPEL that have been reported to the BPEL standardization committee.

formalism, complete with an operational semantics. They discuss the implementation of a tool to describe and automatically verify composition of web services.

In [30] the authors introduce a Petri-net-based algebra to compose web services, based on control flows.

In [70] a Petri-net-based design and verification framework for web 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, which can be used to assist the creation of web service compositions.

In [72] a Petri-net-based architectural description language in which web-service-oriented systems can be modelled 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 [31] 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.

In [63] Orc is translated into colored Petri nets, which is a generalization of Petri nets that allows one to deal with recursion and data handling.

The frameworks and tools described above have the advantage that they allow one to simulate and verify the behaviour of one's model *at design time*, thus enabling the detection and correction of errors as early as possible. As such, these approaches help increase the reliability of web service applications.

3.4 Process Algebras

Process algebras are a popular means to describe and reason about process behaviours. Their underlying semantic foundation is based on labelled transition systems, i.e. automata. Many variants have been defined and the field comes with a rich body of literature. The most well-known process algebras are Milner's Calculus of Communicating Systems (CCS [53]), Hoare's Calculus of Sequential Processes (CSP [32]), the Algebra of Communicating Processes (ACP [6]) by Bergstra and Klop, and the Language of Temporal Ordered Systems (LOTOS [8]) that was standardised by ISO. Like Petri nets, process algebras are precise and well-studied formalisms that allow the automatic verification of certain properties of their behaviours. Likewise, they provide a rich theory on bisimulation analysis, i.e. one can establish whether two processes have equivalent behaviours. Such analyses are useful to establish whether a service can substitute another service in a composition [9] or to verify the redundancy of a service.

The *π-calculus* [54] is a process algebra that has inspired modern composition languages such as XLANG and, subsequently, 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 behaviour 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. Once again without claiming completeness, we now give some examples of process-algebraic approaches to specify and verify secure compositions of web services.

In [64] the authors advocate the use of process algebras to describe, compose, and verify web services, with a particular focus on their interactions. To this aim, they present a case study in which they use CCS to specify and compose web services as processes, and the Concurrency Workbench [13] to validate properties such as correct web service composition. To apply this approach to 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 web service interactions and dynamic compositions. In fact, in [27] a two-way mapping is defined between BPEL and the more expressive process algebra LOTOS. An advantage of this translation is that it includes compensations and exception handling. As such, it permits the verification of temporal properties with the CADP [26] model-checking toolbox.

As is the case for Petri-net-based frameworks and tools, also process-algebraic tools are well suited to improve the reliability of web service development by simulating and verifying the behaviour of one's model at design time.

4 A Comparison of Service Composition Languages and Models: From Industrial Standards to Formal Methods

In this section we present a comparison of the various approaches described above with respect to the service-composition requirements we describe next.

4.1 Service Composition Requirements

We have compared the languages and models that we considered in Section 3 with respect to the following service-composition requirements.

Connectivity and Nonfunctional Properties Every composition approach must guarantee connectivity. With reliable connectivity, one can determine which services are composed and reason about their interactions. Since services are based on message passing, however, developers must also address nonfunctional Quality of Services (QoS) properties, such as timeliness, security, and dependability. It is important in a B2B scenario to define agreement between involved parties. Business agreement defines the contract between two or more parties on QoS. It is necessary to represent required QoS in composed web services.

Composition Correctness Our interest is in large systems of concurrently executing services. A crucial aspect of the correctness of such systems is their temporal behaviour. Behavioural properties can be classified as follows [42].

1) Safety properties: "nothing bad ever happens" (e.g. when an elevator is moving up, it does not attempt to move down without stopping first); and *2)* Liveness properties: "progress takes place" (e.g. if a button inside an elevator is pressed, then the elevator eventually arrives at the corresponding floor). Such behavioural properties are given by a specification which precisely documents a system's desired behaviour. Formal methods provide rigorous mathematical means of guaranteeing large software systems to conform to a specification.

Automatic Composition Many compositional approaches aim to automate the composition, promising faster application development and safer reuse, and facilitating user interaction with complex service sets. By automated composition, the end user or application developer specifies a business goal (e.g. expressed in a description language or in a mathematical notation) and an "intelligent" composition engine selects adequate services and offers the composition to the user in a transparent way. The main problems are how to identify candidate services, how to compose them, and how to verify how closely they match a request. Web service composition languages should enable the representation of semantics of composed services, in order to facilitate the automated composition.

Composition Scalability Composing two services is not the same as composing tens or hundreds of them. In a real-world scenario, end users would typically want to interact with many services, while enterprise applications will invoke chains of possibly several hundreds of services. Therefore, one of the critical issues is how the proposed approaches scale with the number of services involved.

Exception Handling and Compensations Composition of web services uses external web services that are controlled by the web service owner. It must take into account exception handling during the process of invocation in case external web services do not respond. Moreover, business processes are usually long-running processes that may take hours or weeks to complete, and therefore the ability to manage compensations of service invocations is critical for composition as well. Compensations are activities programmed ad hoc to recover (or undo) the effects of completed activities when a long-running transaction fails.

Tool Support Whether a particular approach come with software support.

4.2 Results

Service composition approaches range from those aspiring to become industrial standards (e.g. BPEL and OWL-S) to formal methods. An ideal approach would cover all the requirements we defined above. Below we discuss our comparison of the approaches presented in the previous sections with respect to the above requirements. The outcome is summarized in Figure 2.

	Static Composition	Dynamic Composition	Secure Composition		
	BPEL	OWL-S	Automata	Petri Nets	Process Algebras
Service Connectivity	√	√	√	√	√
Nonfunctional Properties		√			
Composition Correctness			√	√	√
Automatic Composition		√	√	√	√
Composition Scalability			√	√	√
Exception Handling	√			√	√
Compensation	√			√	√
Tool Support (Verification)			√	√	√

Fig. 2. Comparing service composition requirements

Connectivity and Nonfunctional Properties All approaches offer service connectivity. Although the services themselves are modelled in different ways, at the lowest level, the connection comes down to mapping and orchestrating input and output messages between the partner services' service ports. Most approaches neglect the specification of nonfunctional QoS properties such as security, dependability, and performance. Only OWL-S allows users to define some nonfunctional properties, but this capability has yet to be fully specified.

Composition Correctness Verifying correctness depends on the service and composition specifications. BPEL and OWL-S provide no way to verify correctness. BPEL is a language dealing more with implementation than specification, and thus it is difficult to provide a formalism to verify the correctness of BPEL flows. All other approaches support verification in one way or another. Even OWL-S, when combined with Prolog or Petri nets, allows reasoning about correctness. However, the extent to which correctness is verified varies. In the process-algebraic approaches, the π-calculus offers powerful algebraic verification mechanisms to determine liveness, security, and QoS. However, applying such verifications depends on what is typed when you model services as processes. Petri nets use elaborate algebra for verification. In this way one can verify whether a composition has deadlocks by determining whether the corresponding Petri net is live and bounded. Many formal methods are available to prove that a composed service's specification conforms to the model. The issue is to decide what needs to be specified for model checking to produce useful results. Another problem is computing resources (such as CPU time and storage

space). Given the vast state space one must examine, one can easily run out of resources and still not know whether the composition conforms to the model.

Automatic Composition The OWL-S ServiceProfile and ServiceModel provide sufficient information to enable automated discovery, composition, and execution based on well-defined descriptions of a service's inputs, outputs, preconditions, effects, and process model. BPEL does not provide a well-defined semantics. Partners are restricted by structured XML content in WSDL port type definitions. All formal methods discussed allow automatic composition.

Composition Scalability In BPEL, multiple service composition is tedious because XML files quickly grow. Since BPEL composition is recursive one can modularize the composition. Unfortunately, however, BPEL has no standard graphical notation. Some orchestration servers offer a graphical representation and there are proposals to use UML-like notations for descriptions. However, most graphic notations cannot be mapped one-to-one to BPEL's complex language constructs. OWL-S suffers from similar problems. All formal methods discussed instead allow hierarchical service compositions and are thus scalable.

Exception Handling and Compensations BPEL defines a mechanism for catching and handling faults similar to common programming languages like Java. One may also define a compensation handler to enable compensation activities in the event that actions cannot be explicitly undone. OWL-S does not define recovery protocols. Neither BPEL nor OWL-S directly supports query mechanisms to expose the state of executing processes. BPEL lists this item as future work. Most Petri-net-based and process-algebraic models considered can handle compensations and exception handling, but this remains to be seen for the automata-based models.

Tool Support Vendors supporting or planning to support the BPEL specification include Collaxa, which offers a complete orchestration platform for BPEL; IBM, which provides a BPWS4J runtime/editor for BPEL from their alphaWorks Web site; and BindSystems, which provides a BPEL modelling/editing tool. The main problem with the industrial approaches, i.e. the lack of software tools for the verification of the correctness of service compositions, is at the same time the main advantage of the formal methods we considered in this paper.

5 Conclusions and Future Work

While there have been initiatives to compare composition languages [2, 58, 51] and the analysis of composition languages based on workflow patterns [68], these comparisons are conducted almost at the micro level, focusing on specific language structures and control patterns. In this paper, on the other hand, we provide an overview of service composition languages and models. Five approaches—namely BPEL, OWL-S, automata, Petri nets, and process algebras—were chosen

and compared against a total of six requirements that a service composition approach should support in order to facilitate the composition of web services.

OWL-S, like other service composition languages, provides a means of creating the description of web services that can be interpreted programmatically. The distinguishing characteristic of OWL-S is that, while current web service specification standards focus on service syntax, the goal of OWL-S is to facilitate the description of the semantics of services, their interfaces, and their behaviour. The problem of the composition of services is addressed by two orthogonal efforts. On the one hand, most major industrial players propose low-level process modelling and execution languages, like BPEL. These languages allow programmers to implement complex web services as distributed processes and to compose them in a general way. However, the definition of new processes that interact with existing ones needs to be done manually, which is a hard, time-consuming, and error-prone task. On the other hand, research within the semantic web community proposes an unambiguous top-down description of service capabilities, e.g. in OWL-S, thus enabling the possibility to reason about web services, and to automate web service tasks like discovery and composition. The main problem with all these industrial approaches is the verification of correctness. As we show in this paper, this is where formal methods can be of use.

Due to the solid theoretical basis of all formal methods considered in this paper, the tools that come with them allow one to simulate and verify the behaviour 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 reliability of web service applications.

This paper contains only an initial comparison. Many more details (e.g. quantitative information relating to the tool support and to the messaging models supported) are needed in order to understand which languages or models better suit web service composition. Another key point that we would like to deepen in future work is to determine the QoS characteristics that each of the languages and models is able to describe in order to define a QoS web service composition. All this with the objective to understand better which are the necessary elements of a framework for the automatic QoS composition of web services.

References

1. A. Ankolekar et al. DAML-S: Web Service Description for the Semantic Web. In *Proceedings of the 1st International Semantic Web Conference (ISWC'02), Sardinia, Italy*, volume 2342 of *Lecture Notes in Computer Science*, pages 348–363. Springer-Verlag, Berlin, 2002.
2. W.M.P. van der Aalst. Don't go with the flow: Web Services composition standards exposed. *IEEE Intelligent Systems*, 18(1):72–76, 2003.
3. R. Alur. Timed Automata. In N. Halbwachs and D. Peled, editors, *Proceedings of the 11th International Conference on Computer Aided Verification (CAV'99), Trento, Italy*, volume 1633 of *Lecture Notes in Computer Science*, pages 8–22. Springer-Verlag, Berlin, 1999.
4. R. Alur and D.L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

5. M.H. ter Beek, C.A. Ellis, J. Kleijn, and G. Rozenberg. Synchronizations in Team Automata for Groupware Systems. *Computer Supported Cooperative Work—The Journal of Collaborative Computing*, 12(1):21–69, 2003.
6. J.A. Bergstra and J.W. Klop. Algebra of Communicating Processes with Abstractions. *Theoretical Computer Science*, 33:77–121, 1985.
7. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, May 2001.
8. T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks*, 14:25–59, 1987.
9. L. Bordeaux, G. Salaün, D. Berardi, and M. Mecella. When are Two Web Services Compatible? In M.-C. Shan, U. Dayal, and M. Hsu, editors, *Proceedings of the 5th International Workshop on Technologies for E-Services (TES'04), Toronto, Canada*, volume 3324 of *Lecture Notes in Computer Science*, pages 15–28. Springer-Verlag, Berlin, 2004.
10. D. Brickley and R.V. Guha. Resource Description Framework RDF Vocabulary Description Language (Version 1.0): RDF Schema, 2004. http://www.w3.org/TR/rdf-schema/.
11. Z. Chaochen, C.A.R. Hoare, and A.P. Ravn. A Calculus of Durations. *Information Processing Letters*, 40(5):269–276, 1991.
12. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language WSDL (Version 1.1), 2001. http://www.w3.org/TR/wsdl.
13. R. Cleaveland, T. Li, and S. Sims. The Concurrency Workbench of the New Century (Version 1.2), 2000. http://www.cs.sunysb.edu/~cwb/.
14. W.R. Cook and J. Misra. Orc—An Orchestration Language (Version 0.5), 2005. http://www.cs.utexas.edu/users/wcook/projects/orc/.
15. IBM Corporation. Websphere software. http://www.ibm.com/software/info1/websphere.
16. IBM Corporation. Business Process Execution Language for Web Services BPEL-4WS (Version 1.1), 2002. http://www.ibm.com/developerworks/library/ws-bpel.
17. IBM Corporation. BPEL for Java technology BPELJ, 2004. http://www.ibm.com/developerworks/library/ws-bpelj/.
18. IBM Corporation. Web Services Coordination, 2004. http://www.ibm.com/developerworks/library/ws-coor/.
19. IBM Corporation. Web Services Transactions specifications, 2004. http://www.ibm.com/developerworks/library/ws-coor/.
20. Oracle Corporation. Oracle bpel process manager. http://www.oracle.com/technology/products/ias/bpel/.
21. F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana. The next step in Web services. *Communications of the ACM*, 46(10):29–34, 2003.
22. G. Díaz, J.J. Pardo, M.-E. Cambronero, V. Valero, and F. Cuartero. Automatic Translation of WS-CDL Choreographies to Timed Automata. In M. Bravetti, L. Kloul, and G. Zavattaro, editors, *Formal Techniques for Computer Systems and Business Processes: Proceedings of the International Workshop on Web Services and Formal Methods (WS-FM'05), Versailles, France*, volume 3670 of *Lecture Notes in Computer Science*, pages 230–242. Springer-Verlag, Berlin, 2005.
23. J.S. Dong, Y. Liu, J. Sun, and X. Zhang. Verification of Computation Orchestration via Timed Automata, February 2006. Unpublished manuscript. http://nt-appn.comp.nus.edu.sg/fm/orc/.
24. C.A. Ellis. Team Automata for Groupware Systems. In S.C. Hayne and W. Prinz, editors, *Proceedings of the International ACM SIGGROUP Conference on Support-*

ing Group Work: The Integration Challenge (GROUP'97), Phoenix, AZ, U.S.A., pages 415–424. ACM Press, New York, NY, 1997.
25. The Open Source BPEL Engine. Active BPEL (Version 2.0). http://www.activebpel.org.
26. J. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP: A Protocol Validation and Verification Toolbox. In R. Alur and T.A. Henzinger, editors, *Proceedings of the 8th International Conference on Computer-Aided Verification (CAV'96), New Brunswick, NJ*, volume 1102 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1996.
27. A. Ferrara. Web Services: a Process Algebra Approach. In M. Aiello, M. Aoyama, F. Curbera, and M.P. Papazoglou, editors, *Proceedings of the 2nd International Conference on Service-Oriented Computing (ICSOC'04), New York, NY*, pages 242–251. ACM Press, New York, NY, 2004.
28. Organization for the Advancement of Structured Information Standards OASIS. http://www.oasis-open.org.
29. X. Fu, T. Bultan, and J. Su. Analysis of Interacting BPEL Web Services. In S.I. Feldman, M. Uretsky, M. Najork, and C.E. Wills, editors, *Proceedings of the 13th International Conference on the World Wide Web (WWW'04), New York, NY*, pages 621–630. ACM Press, New york, NY, 2004.
30. R. Hamadi and B. Benatallah. A Petri Net-based Model for Web Service Composition. In K.-D. Schewe and X. Zhou, editors, *Proceedings of the 14th Australasian Database Conference (ADC'03), Adelaide, South Australia*, volume 17 of *CRPIT*, pages 191–200. Australian Computer Society, 2003.
31. S. Hinz, K. Schmidt, and Ch. Stahl. Transforming BPEL to Petri Nets. In W.M.P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Proceedings of the 3rd International Conference on Business Process Management (BPM'05), Nancy, France*, volume 3649 of *Lecture Notes in Computer Science*, pages 220–235. Springer-Verlag, Berlin, 2005.
32. C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, London, 1985.
33. G.J. Holzmann. *The SPIN Model Checker—Primer and Reference Manual*. Addison Wesley, Reading, MA, 2003.
34. Business Process Modeling Initiative. Business Process Modeling Language BPML, 2002. http://www.bpmi.org.
35. J. Clark et al. XML Path Language XPath (Version 1.0), 1999. http://www.w3.org/TR/xpath.
36. H. Jain, L. Liu, and L.-J. Zhang, editors. *Proceedings of the IEEE International Conference on Web Services (ICWS'04), San Diego, CA*. IEEE Computer Society Press, Los Alamitos, CA, 2004.
37. N. Kavantzas. Aggregating web services: Choreography and ws-cdl. Technical report, Orcale Corporation, 2004.
38. N. Kavantzas, D. Burdett, and G. Ritzinger. Web Services Choreography Description Language WSCDL (Version 1.0), 2004. http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/.
39. D.K. Kaynar, N.A. Lynch, R. Segala, and F.W. Vaandrager. *The Theory of Timed I/O Automata*. Synthesis Lectures on Computer Science. Morgan & Claypool, Ft. Collins, CO, 2006.
40. R. Kazmiakin, P.K. Pandya, and M. Pistore. Modelling and Analysis of Time-related Properties in Web Service Compositions. In *Proceedings of the 1st International Workshop on Engineering Service Compositions (WESC'05), Amsterdam, The Netherlands*, 2005. To appear.

41. B. Kiepuszewski, A.H.M. ter Hofstede, and W.M.P. van der Aalst. Fundamentals of Control Flow in Workflows. *Acta Informatica*, 39(3):143–209, 2003.
42. L. Lamport. Proving the Correctness of Multiprocess Programs. *IEEE Transactions on Software Engineering*, 3(2):125–143, 1977.
43. K.G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, 1997.
44. F. Leymann. Web Services Flow Language WSFL (Version 1.0). Technical report, IBM Corporation, 2001.
45. N.A. Lynch. Input/Output Automata: Basic, Timed, Hybrid, Probabilistic, Dynamic, ... In R.M. Amadio and D. Lugiez, editors, *Proceedings of the 14th International Conference on Concurrency Theory (CONCUR'03), Marseille, France*, volume 2761 of *Lecture Notes in Computer Science*, pages 187–188. Springer-Verlag, Berlin, 2003.
46. N.A. Lynch and M.R. Tuttle. An Introduction to Input/Output Automata. *CWI Quarterly*, 2(3):219–246, 1989. Also published as Technical Memo MIT/LCS/TM-373, MIT, 1988.
47. Z.M. Mao, E.A. Brewer, and R.H. Katz. Fault-tolerant, Scalable, Wide-Area Internet Service Composition. Technical Report UCB/CSD-01-1129, University of California, Berkeley, 2001.
48. D.L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview, 2004. http://www.w3.org/TR/owl-features/.
49. S. McIlraith and T.C. Son. Adapting Golog for Composition of Semantic Web Services. In D. Fensel, F. Giunchiglia, D.L. McGuinness, and M.-A. Williams, editors, *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning (KR'02), Toulouse, France*, pages 482–496. Morgan Kaufmann, San Mateo, CA, 2002.
50. S.A. McIlraith, T.C. Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.
51. J. Mendling and M. Müller. A Comparison of BPML and BPEL4WS. In R. Tolksdorf and R. Eckstein, editors, *Proceedings of the 1st Conference Berliner XML-Tage, Berlin*, pages 305–316, 2003.
52. Microsoft Corporation. Microsoft biztalk server. http://www.microsoft.com/biztalk.
53. R. Milner. *Communication and Concurrency*. Prentice Hall, London, 1989.
54. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts I and II. *Information and Computation*, 100(1):1–77, 1992.
55. S. Narayanan and S.A. McIlraith. Simulation, Verification and Automated Composition of Web Services. In *Proceedings of the 11th International World Wide Web Conference (WWW'02), Honolulu, Hawaii*, pages 77–88. ACM Press, New York, NY, 2002.
56. OpenStorm. Web Service Orchestration Software. http://www.openstorm.com.
57. C. Ouyang, W.M.P. van der Aalst, S. Breutel, M. Dumas, A.H.M. ter Hofstede, and H.M.W. Verbeek. Formal Semantics and Analysis of Control Flow in WS-BPEL. Technical Report BPM-05-15, BPM Center, 2005.
58. C. Peltz. Web services orchestration, a review of emerging technologies, tools, and standards. Technical report, Hewlett-Packard Company, 2003.
59. C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, Rheinisch-Westfälisches Institut für Instrumentelle Mathematik an der Universität Bonn, 1962. In German.
60. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*. Number 1491 in Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1998.

61. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets II: Applications*, volume 1492 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.
62. R. Reiter. *Knowledge in Action—Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, Massachusetts, MA, 2001.
63. S. Rosario, A. Benveniste, S. Haar, and C. Jard. Net system semantics of Web Services Orchestrations modeled in ORC. Technical Report 1780, Istitut de Recherche en Informatique et Systèmes Aléatoires, January 2006.
64. G. Salaün, L. Bordeaux, and M. Schaerf. Describing and Reasoning on Web Services using Process Algebra. In *[36]*, pages 43–50, 2004.
65. Universal Description, Discovery and Integration UDDI. http://www.uddi.org/specification.html.
66. World Wide Web Consortium W3C. Semantic Web, 2001. http://www.w3.org/2001/sw/.
67. World Wide Web Consortium W3C. Latest SOAP versions, 2003. http://www.w3.org/TR/soap/.
68. P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Pattern-Based Analysis of BPEL4WS. Technical Report FIT-TR-2002-04, Queensland University of Technology, Brisbane, 2002.
69. P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Analysis of Web Services Composition Languages: The Case of BPEL4WS. In I.-Y. Song, S.W. Liddle, T.W. Ling, and P. Scheuermann, editors, *Proceedings of the 22nd International Conference on Conceptual Modeling (ER'03), Chicago, IL*, volume 2813 of *Lecture Notes in Computer Science*, pages 200–215. Springer-Verlag, Berlin, 2003.
70. X. Yi and K. Kochut. A CP-nets-based Design and Verification Framework for Web Services Composition. In *[36]*, pages 756–760, 2004.
71. S. Yovine. KRONOS: A Verification Tool for Real-Time Systems. *International Journal on Software Tools for Technology Transfer*, 1(1–2):123–133, 1997.
72. J. Zhang, J.-Y. Chung, C.K. Chang, and S. Kim. WS-Net: A Petri-net Based Specification Model for Web Services. In *[36]*, pages 420–427, 2004.