Grant Agreement N° 215483

Title:	*Overview of the State of the Art in Composition and Coordination of Services*
Authors:	*George Baryannis (UOC), Manuel Carro (UPM), Olha Danylevych (USTUTT), Schahram Dustdar (TUW), Dimka Karastoyanova (USTUTT), Kyriakos Kritikos (UOC), Philipp Leitner (TUW), Florian Rosenberg (TUW), Branimir Wetzstein (USTUTT)*
Editor:	*Branimir Wetzstein (USTUTT)*
Reviewers:	*Ivona Brandic (TUW)*
	Dimka Karastoyanova (USTUTT)
	Raman Kazhamiakin (FBK)
Identifier:	*Deliverable # PO-JRA-2.2.1*
Type:	*Deliverable*
Version:	*1*
Date:	*15 July 2008*
Status:	*Final*
Class:	*External*

Management Summary

This deliverable presents a survey on the state-of-the-art in service composition. The report is structured into three main sections. First, different models and languages for service composition are described. These involve service orchestration, choreography, wiring, coordination, and semantic WS composition. The next part evaluates approaches to creation and development of service compositions, including model-driven, automated, and QoS-aware service composition. Finally, formal methods to verification of service compositions are presented. Each section starts with an introduction and classification of the approaches and then describes the main contributions in the corresponding area.

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement n° 215483 (S-Cube).

Members of the S-Cube consortium:

University of Duisburg-Essen (Coordinator)	Germany
Tilburg University	Netherlands
City University London	U.K.
Consiglio Nazionale delle Ricerche	Italy
Center for Scientific and Technological Research	Italy
The French National Institute for Research in Computer Science and Control	France
Lero - The Irish Software Engineering Research Centre	Ireland
Politecnico di Milano	Italy
MTA SZTAKI – Computer and Automation Research Institute	Hungary
Vienna University of Technology	Austria
Université Claude Bernard Lyon	France
University of Crete	Greece
Universidad Politécnica de Madrid	Spain
University of Stuttgart	Germany

Published S-Cube documents

These documents are all available from the project website located at http://www.s-cube-network.eu/

The S-Cube Deliverable Series

Vision and Objectives of S-Cube

The Software Services and Systems Network (S-Cube) will establish a unified, multidisciplinary, vibrant research community which will enable Europe to lead the software-services revolution, helping shape the software-service based Internet which is the backbone of our future interactive society.

By integrating diverse research communities, S-Cube intends to achieve world-wide scientific excellence in a field that is critical for European competitiveness. S-Cube will accomplish its aims by meeting the following objectives:

- Re-aligning, re-shaping and integrating research agendas of key European players from diverse research areas and by synthesizing and integrating diversified knowledge, thereby establishing a long-lasting foundation for steering research and for achieving innovation at the highest level.
- Inaugurating a Europe-wide common program of education and training for researchers and industry thereby creating a common culture that will have a profound impact on the future of the field.
- Establishing a pro-active mobility plan to enable cross-fertilisation and thereby fostering the integration of research communities and the establishment of a common software services research culture.
- Establishing trust relationships with industry via European Technology Platforms (specifically NESSI) to achieve a catalytic effect in shaping European research, strengthening industrial competitiveness and addressing main societal challenges.
- Defining a broader research vision and perspective that will shape the software-service based Internet of the future and will accelerate economic growth and improve the living conditions of European citizens.

S-Cube will produce an integrated research community of international reputation and acclaim that will help define the future shape of the field of software services which is of critical for European competitiveness. S-Cube will provide service engineering methodologies which facilitate the development, deployment and adjustment of sophisticated hybrid service-based systems that cannot be addressed with today's limited software engineering approaches. S-Cube will further introduce an advanced training program for researchers and practitioners. Finally, S-Cube intends to bring strategic added value to European industry by using industry best-practice models and by implementing research results into pilot business cases and prototype systems.

S-CUBE materials are available from URL: http://www.s-cube-network.eu/

Contents

Chapter 1

Introduction

Service-oriented computing (SOC) [1] promotes the idea of composing self-contained services into a loosely coupled agile software system. In that context, services are often described as autonomous software components that can be described, published, discovered and composed in a platform-neutral way. They perform functions ranging from simple atomic requests to executing complex business processes [2]. Software systems built on top of services utilize a service-oriented architecture (SOA) [3, 4]. SOA is a way of designing a system so that it can provide services to end users and/or other applications in the network. Currently, Web services [5, 6] are the most prominent realization technology for SOAs. They use well-known, tested and established standards, such as SOAP as message encoding format or WSDL for service definition.

One of the most interesting and relevant properties of services is the possibility to combine a number of existing services to create a more general composite service (aggregation [7]). Composing services allows for the definition of increasingly complex applications by progressively combining services at increasing levels of abstraction [8]. In literature, two different types of composition are often differentiated: on the one hand, the term *service orchestration* is used to describe interactions at message level, including the business logic and execution order under control of a single end point – today, WS-BPEL has succeeded earlier orchestration languages such as WSFL or XLANG and is the standard language for representing Web service orchestrations; on the other hand, the term *service choreography* denotes publicly visible message exchanges, i.e., interactions that are under the control of two or more end points [9]. Besides orchestration and choreography, *service coordination*, *service wiring*, and *semantic Web service composition* can also be considered as types of service compositions. All of these composition model types have different purposes, but can often be combined together. For example, one often starts modeling a choreography, and then refines the choreography to orchestrations of different partners.

Service Composition Models are involved in a lifecycle, which comprises modeling, verification, execution, and monitoring phases. When it comes to modeling of service compositions, there exist several approaches: *model-driven service composition* copes with generating service orchestration models from more abstract models, which are often abstract business process models created by business analysts; *automated service composition* selects services and orchestrates them automatically based on an abstract goal specification; *QoS-aware service composition* can be used to select those services which satisfy local and global QoS constraints of the service composition. After the composition is created, it can be verified against certain properties, such as whether it is deadlock-free. Therefore, often methods such as model checking are used. After deployment of the service composition to the corresponding IT infrastructure, which is typically a process engine in combination with a service bus, the composition is executed. At runtime, the composition can be adapted by for example rebinding other services, if the predefined service fails. Monitoring of service compositions is performed either for run-time verification or to measure certain metrics such as average response time of the service composition.

In the following survey we analyze the current state of the art in service composition in a structured way: we present and classify research directions and approaches and elaborate on the most important

papers in each category. Additionally, we present how the papers are related to each other, and what the main contributions of each work are. Often, research work exhibits characteristics from more than one category, which is then mentioned explicitly in the text. Please note that not all existing papers to every category can be discussed – often the authors had to choose between a number of similar research approaches; in these cases the more prominent approach has been chosen to be included in this survey.

The survey is structured into three main sections. First, in Chapter 2 different models and languages for service composition are described. Chapter 3 evaluates approaches to creation and development of service compositions, including model-driven, automated, and QoS-aware service composition. In Chapter 4 formal methods to verification of service compositions are presented. Finally, Chapter 5 summarizes the survey and identifies research challenges.

Several of the topics addressed in this survey are closely related to the topics addressed in other S-Cube surveys, or are addressed with a different focus in those surveys. In particular, those surveys deal with some aspects of service composition and its lifecycle, which could not been addressed in this survey. The relation of this deliverable to other S-Cube deliverables is as follows:

- *PO-JRA-1.1.1 "State of the art report on software engineering design knowledge and Survey of HCI and contextual Knowledge"* deals with engineering of service-based applications (SBAs) in general. It describes engineering methods for SBAs, requirements engineering, and mechanisms to support deployment, execution, and runtime evolution of SBAs. It however does not focus on service compositions in particular and resulting approaches to creation of these service compositions, which are described in Chapter 3 of this deliverable.

- *PO-JRA-1.2.1 "State-of-the-Art report, gap analysis of knowledge on principles, techniques and methodologies for monitoring and adaptation of SBAs"* describes the state-of-the-art in monitoring and adaptability of services. In particular, it also deals with monitoring and adaptability of service compositions, thus covering important aspects of the service composition lifecycle which are out of scope of this deliverable.

- *PO-JRA-1.3.1 "Survey of Quality Related Aspects Relevant for Service-based Applications"* deals with specification and negotiating of QoS and SLAs and quality assurance for Service-based Applications. Concerning the QoS aspect, in this survey, we focus on QoS-aware service composition which is based on guaranteeing local and global quality constraints in service compositions. For what concerns the verification of service compositions, PO-JRA-1.3.1 focuses on non-functional properties such as performance prediction, while approaches which lean more towards checking correctness of service compositions are examined here in Chapter 4.

- *PO-JRA-2.1.1 "State of the art survey on Business Process Modelling and Management"* deals with BPM which is situated in the layer above service compositions. PO-JRA-2.1.1 explains the relation between BPM and SOA in general when describing the BPM lifecycle. In this survey, in Section 3.1, we deal with model-driven service composition, and present works which transform business process models to service compositions.

- *PO-JRA-2.3.1 "Use case description and state-of-the-art(Self-* Service Infrastructure, Service Discovery)"* deals with service infrastructure which is situated in the layer below service compositions, and related issues such as service discovery, dynamic binding and invocation. Dynamic binding, in particular, enables runtime adaptability of service compositions by discovering and binding new services (which are part of those compositions).

Chapter 2

Service Composition Models

2.1 Introduction

For the focus of this survey, we define *service composition* as the combination of a set of services for achieving a certain purpose. Interchangeably, the term ”service aggregation” could be used in this context [7].

We consider four types of service composition models:

- **Service Orchestration:** Service orchestration is the most prominent approach to service composition, and often the term ”service composition” is actually used for service orchestration. Thereby, a new service is created by orchestrating several services in a process flow. Orchestrated services can be atomic services, i.e. self-contained programs which do not use other services, or again service orchestrations. The standard language for orchestrating Web services is BPEL.
- **Service Choreography:** While in service orchestration the perspective is on the implementation of a service for one partner, a service choreography defines the interaction protocol between several partners services. The goal is to define the so called public processes of the interacting partners and how they communicate together. For the specification of service choreographies, visual notations can be used such as BPMN and Let’s Dance. Other approaches define a completely new XML-based language such as WS-CDL, or adapt a language for service orchestrations such as BPEL4Chor.
- **Service Wiring:** Services specify provided and requested interfaces in form of operations with inputs and outputs. In order to create executable service assemblies, the requested interfaces of one service are *wired* to provided interfaces of other services. The services assembled in this way can again be recursively exposed as a service which can be wired and invoked. A service assembly is a deployable artifact, which is deployed to an enterprise service bus.
- **Service Coordination:** In agreement-based service coordination, a distributed activity is created by temporarily grouping a set of service instances following a coordination protocol. At the end of the activity a coordinator decides on the outcome of the protocol and disseminates the result to the participating services. One example is that of the bid auction. Buyers and sellers are participating Web services following a certain coordination protocol and being coordinated by an auctioneer. At the end of the bidding period, the auctioneer decides who is the winner of the bid and informs both buyer and seller, and the temporary collection of services ceases to exist.

As it is not a new composition model type, we look at *Semantic Web Service Composition* separately. The idea of Semantic Web Services (SWS) is to describe Web service interfaces semantically in a machine-readable manner, thus enabling automatic Web service discovery and composition. Semantic WS composition approaches either extend existing composition languages, e.g. BPEL4SWS, or create completely new languages, e.g. OWL-S.

In the following, we describe the state-of-the-art in service orchestration, choreography, wiring, coordination, and semantic WS composition. In each section, we introduce the main concepts, present the main approaches, and finally give a comparison of these approaches when applicable. In the last section, we deal with the combination of different service composition model types.

2.2 Service Orchestration

A service orchestration composes a new service by orchestrating several services in a process flow. Services which are part of an orchestration can be atomic services or again service orchestrations. Service orchestration is thus a recursive model. Atomic services are self-contained and do not use other services for implementing their business logic.

A service orchestration is a proactive composition that drives the interactions with the services it orchestrates. It defines execution semantics on a set of activities that perform these interactions. These execution semantics can be defined in different ways, e.g., calculus-based (XLANG), statechart-based [10], or graph-based (WSFL).

Service orchestrations can be programmed using a multi-purpose 3GL programming language, such as Java, but more typically a workflow-like language is used. In the following, we concentrate on languages which deal with services as first-class citizens.

There have been several efforts related to the creation of complex Web services out of simpler ones using workflow languages, e.g. WSFL [11], XLANG [12], BPML [13], BPEL [14]. In the context of Web services, these service orchestrations are called WS-flows. Besides WS-Flow languages, there are other approaches to service orchestration, such as JOpera [15] which is a visual service composition approach, not constrained to orchestration of (WSDL-based) Web services.

2.2.1 Orchestration Languages and Approaches

WS-Flow Languages

WS-Flows are similar to workflows, whereby the applications invoked by the activities of the workflow are Web services. There have been several efforts related to the creation of WS-Flow languages, in particular WSFL [11], XLANG [12], BPML [13], and WS-BPEL [14]. BPML was a proposed language by BPMI, but was dropped in favor of BPEL. BPEL can be seen as the successor of WSFL from IBM and XLANG from Microsoft and combines concepts and constructs from both languages. As BPEL has superseded these languages and has become a standard, in the following we will describe only BPEL.

BPEL is an XML-based language for specifying WS-Flows. WS-Flows can be thought of having two dimensions: the ”what?” dimension, and the ”what with?” dimension. The ”what?” dimension is represented by the control flow and the data flow, and is also referred as business logic. The control flow includes alternative paths of execution, defines how exceptional situations are handled (fault handling and compensation), how the process reacts to events signalled by users and the environment (event handling), and additional rules and constraints. Another aspect of the ”what?” dimension in traditional workflows is the so-called data flow which defines how data is exchanged among activities and between the workflow and its participants. The data flow is explicitly present in only some of the existing WS-Flow languages, such as WSFL, and XLANG. BPEL, however, does not explicitly model the data flow. Activities can access data variables, which can be defined either globally for the whole process or for specific (nested) scopes (which restrict their visibility).

The control flow of a BPEL process is specified using a combination of graph-based (WSFL) and calculus-based (XLANG) approaches. BPEL provides constructs for:

- *Interaction modeling* for invoking operations on other Web services, receiving and replying invocations on the process itself from the services which use the process. An executable BPEL process

is exposed as a WSDL Web service to external service requesters, and can thus be invoked as any other Web service.

- *Data manipulation* for reading and writing data variables associated with the process.
- *Control flow* for controlling the flow of the process using both (or either) directed control links and compound activities that impose, on the activities that are contained within them, semantics such as parallelism, sequencing, loops, and choice.
- *Fault handling, Event handling, and Compensation*

The second dimension of WS-flows, the ”what with?” dimension, is the one assigning to each interacting activity a participant, namely a Web service. Interacting activities are those tasks in the model of a WS-flow that stand for interaction with a partner Web service. The definitions of these activities specify the participating Web services only on the abstract level, namely only their portTypes and operations are to be supplied, and refer to no actual endpoints that implement these portTypes. This is reflected by the BPEL specification; this has been also one of the goals of XLANG, WSFL and BPML.

BPEL compositions are based purely on the interfaces of the composed services, that is, their abstract definitions in WSDL (portTypes, operations, and message types). From an architectural perspective, the resulting process is itself a Web service that can be exposed using WSDL definitions that define the portTypes it provides to clients. Additionally, the BPEL process also declares which portTypes it requires from each of the services it composes. Mechanisms for binding service instances to a process are intentionally left up to the runtime and therefore out of the process definition. This is where service wiring mechanisms come into play (Section 2.4).

BPEL process models can be defined as abstract or executable. An executable process model specifies a service orchestration which can be executed by a BPEL engine. In contrast, an abstract process model hides some activities (a part of the process model) by defining them as ”opaque”. Abstract process models are either used to define process templates, or to define behavioral interfaces. A behavioral interface contains mostly messaging activities which denote how the service requester should communicate with the process. A behavioral interface thus specifies a public process which can be seen by external requesters. Private information (process logic) which is not to be exposed to the outside is replaced by opaque activities.

BPEL supports only orchestration of Web services described in WSDL. There are, however, extensions to BPEL which relax this restriction. BPEL4People enables incorporating human tasks into a BPEL process. BPEL^{light} [16] removes the dependency on WSDL altogether, and describes just conversations as message exchanges with partner services. In a grounding step these message exchanges can be grounded to different technologies.

JOpera

JOpera [17] is a visual service composition tool built on the Eclipse platform. It supports an integrated visual service composition development, execution and monitoring.

For the definition of service compositions, JOpera provides the JOpera Visual Composition Language (JVCL) [15]. Services are composed using processes, which define their interactions using two seperate graphs for specifying the data flow and the control flow. In JOpera, the data flow has to be explicitly specifed. The data flow graph defines where the information required by each service comes from. The data flow is specified by connecting output parameters of one task to input parameters of others. The control flow is separetely visualized and is mostly inferred from data flow, which is seen as the primary representation in JOpera. Therefore, JOpera provides control flow extraction algorithms to ensure the automatic reconciliation between the two. Exceptions where explicit control flow constructs are needed are branching, synchronization, exception handling, loops, and more advanced workflow patterns.

When it comes to incorporating service invocations into the composition, JOpera is not constrained to WSDL Web services but supports also other service types, such as UNIX Commands, Grid Services (WSRF), Java snippets, SQL Queries (JDBC), XSLT scripts, which are imported into a service library and can be drag-and-dropped into the process flow. Additional service types can be added using an extension-point for custom service invocation plugins.

For executing the service composition model, it is transformed to Java byte code.

2.2.2 Comparison

As BPEL has superseded the other WS-Flow languages, we will not compare WS-Flow languages among themselves. In the following we thus compare BPEL and JOpera:

- *Modeling Approach:* JOpera provides a visual language which is compiled to Java byte code before execution. BPEL provides a standard XML syntax for its language. Process models can thus be exchanged and run on different BPEL process engines. Visual modeling is supported by BPEL visual editors; there is however no standard visual notation for BPEL language constructs.

- *Orchestration Model Properties:* JOpera supports explicit data flow and graph-based control flow. BPEL, on the other hand, supports only implicit data flow, but both graph-based and block-based control flow modeling. In JOpera the language is independent of services to be orchestrated. The description of the process, data flow and control flow, is separated from the description of how to invoke each of its tasks. Many constructs, such as timeouts and data transformations, are shifted from the language definition to the standard library of service types. In BPEL, interaction modeling and data transformations are part of the language.

- *Interaction with Services:* When it comes to service type support, BPEL supports only interaction with WSDL Web services. In order to support other types of services, language extensions are needed, e.g. BPEL4J or BPEL^{light}. In JOpera the language is independent of services to be orchestrated. Several service types, such as UNIX Commands, Grid Services (WSRF), Java snippets, SQL Queries (JDBC), XSLT scripts, which are part of a service library and can be drag-and-dropped into the process flow, are supported.

2.3 Service Choreography

A service choreography is a design-time artefact that describes a multiparty collaboration. That collaboration might have an inter-organisational as well as intra-organisational character and is performed in order to achieve certain (business) goals. Choreography focuses mainly on the conversational aspect - message exchanges between the participating processes - and doesn't aim to expose the internal structure of the processes. Being in full control of own internal business processes, the partners may specify on the level of choreography only the tasks which are essential for the description of the conversations. There is a set of business processes realised by orchestrations in lower abstraction level under the global interaction description provided by a choreography. The goal of service choreography is to give a notion on how these orchestrations are "tied" together and so to capture the complex conversations between business partners from a global view.

There are several usage scenarios for service choreographies:

- Specification (agreement) on how business partners interact with each other
- Standardization and thus support for the engagement of a wider set of users/providers
- Conformance checking and basis for adapting existing orchestrations
- Basis for development of new orchestrations

- Monitoring, e.g. for legal purposes
- Optimization of collaborations

There is a spectrum of modelling approaches and corresponding choreography languages, which differ in the way they capture the collaborations. In the following we will introduce well-known categorisations of choreography languages and then present the most important representative languages. Service Interaction Patterns proposed by [18] represent a set of most often occurring interaction scenarios and therefore can be used in assessment of choreography modelling languages. The main categories of Service Interaction Patterns are those outlining bilateral interactions, as well as multilateral, competing, atomic and causally related interactions.

As stated in [19], the languages for modelling choreographies can be distinguished in respect to two orthogonal criteria: (i) Interaction Models vs. Interconnected Interface Models, and (ii) Implementation-specific vs. Implementation-independent Choreography Languages.

Interaction Models vs. Interconnected Interface Models

The Interaction Modeling approach describes a choreography from a truly global point of view by the means of combination of elementary interaction blocks, such as request-response and one-way into more complex interactions where dependencies between single interaction blocks are specified. One specialty of the interaction modeling style is that the data and control flow is not assigned to any of the roles but defined globally instead. Therefore the interaction choreography approaches allow to model locally unenforceable interactions: additional synchronization messages are needed to construct a locally enforceable collaboration model [19]. The strengths of interaction modeled choreographies unfolds in the early stages of service developing lifecycle where the stakeholders need an overall global perspective in order to recognize potential services and capture those interactions.

In contrast to the interaction modeling approach, the Interconnected Interface Choreography Model specifies the interface of each partaking process role by defining control flow dependencies between the send and receive activities inside of the participants. Local unenforceability is not an issue in the context of this modeling style, but one has to deal with the problem of possible incompatibility of the participants which can lead to e.g. deadlocks. This modeling approach is more convenient when creating new collaboration-enabled services as well as adapting existing ones.

Implementation-specific vs. Implementation-independent Choreography Languages

Based on the target user group, we can further classify choreography languages into implementation-independent and implementation-specific ones. The implementation-independent choreography languages address the Business Process Management Community, who observe the collaborations of the processes from the business perspective and make fundamental decisions about the collaboration. The implementation-specific choreography languages address the choreography modelling while looking forward to its later execution. Those address such issues as communication protocols, security issues as well as exception handling. The implementation-specific choreography languages refer to Web Services.

2.3.1 Choreography Languages

The positioning of the main choreography languages regarding the previous categorizations is depicted in the table 2.3.1. These languages are discussed in following.

BPMN

Business Process Modeling Notation (BPMN) [20] is an implementation-independent graphical modeling language. It allows description of both intra- and inter-organizational business processes. By means of a

	Implementation-independent	Implementation-specific
Interaction Model	Let's Dance, BPSS	WS-CDL
Interconnected Interface Model	BPMN, MSC	WSFL, BPEL4Chor

Table 2.1: Choreography Languages: Categorization

wide spectrum of constructs, BPMN supports modeling of complex control flow scenarios and supports modeling of both private (internal) processes and abstract (public) processes.

In order to model business entities and roles, BPMN uses "pools". It further allows combining the subsets of activities of a business process into "swim lanes" and so makes the mapping of activities to organizational units possible. Besides allowing to describe the control flow of a participant process, BPMN aims to support modeling of the collaboration of several processes and thus choreographies. While the control flow describes behavior of each participant and specifies the ordering of the interaction-relevant activities inside of each participant (BPMN pool), the message flow "ties" the participating processes into conversations.

There are several disadvantages in the usage of BPMN for modeling of choreographies. One of problematical cases arises when many participants of the same type are evolved in a conversation, since in BPMN each single participant of the conversation has to be represented by a separate pool and one cannot address a set of participants of a certain type. Another issue is that BPMN lacks formal execution semantics. Despite of described deficiencies, BPMN is popular in the industry. Since BPMN is not executable work has been investigated to allow mapping from BPMN to BPEL [21].

Let's Dance

Let's Dance [22] is an implementation-independent interaction modeling language, which was originally created for modeling of choreographies and was meant to cover all Service Interaction Patterns [18]. Being a visual modeling language with basic building blocks representing single atomic message exchanges, Let's Dance encompasses the properties and requirements of collaborations on a global level. Targeting business analysts, it does not support any technology-specific configurations. As it is an interaction modeling approach, Let's Dance does not specify explicitly the behavioral structure of participants. Graph reduction techniques have to be used in order to generate the corresponding participant behaviors models [22] out of the global interaction model.

Web Services Flow Language (WSFL)

WSFL [11], as described in the previous section, is a language for defining service orchestrations. It, however, in addition supports modeling of choreographies based on the interconnection interface model.

WSFL depicts both: a local perspective, which shows how the services are combined into one business process, and a global perspective, which illustrates the interaction of those business processes. The local perspective defines the whole execution sequence of the service orcchestration by the means of control flow as well as data flow constructs and is called "Flow Model". Orthogonal to the Flow Model, the Global Model shows merely the interactions omitting any detailed specification of internal execution ordering between the service orchestrations. The "plug links" describe the interaction of a single Web Service with an operation of another Web Service. Plug links are based on WSDL port types and operations. WSFL is thus an implementation-specific language.

Web Services Choreography Description Language (WS-CDL)

Web Services Choreography Description Language [23] belongs to the class of interaction modeling languages and was introduced as successor of WSCI and WSCL. The basic building blocks for defining process collaborations in WS-CDL are bi-lateral interactions with one or two messages (request-only,

response-only, request-response). A set of control flow constructs (sequence, parallel, choice and workunit) allows composing interactions into actions. WS-CDL doesn't support all workflow patterns and service interaction patterns, e.g. it doesn't directly support the patterns where exact number of partaking processes is unknown in runtime (e.g. "one-to-many-send/receive" pattern) [24]. The reusability of WS-CDL is limited because of its tight integration with WSDL: changes of WSDL configuration often imply the need to change the WS-CDL choreography. Another important point of critic on WS-CDL is that it's complicated to perform a proper transformation of WS-CDL choreographies into abstract BPEL orchestrations since its control flow constructs differ from those in BPEL.

BPEL4Chor

BPEL4Chor [25] is an implementation-specific choreography modeling language which belongs to the class of interconnected interface modeling approaches. With only a few new constructs, it introduces an additional layer on the top of Abstract BPEL supporting a seamless integration between choreography and orchestrations. BPEL4Chor consists of three "perspectives": participant topology, participant behavior description, and participant grounding. Participant behavior description focuses on the structure of an individual participant by adopting, with slight changes, the existing constructs of abstract BPEL. Important features of participant behavior description are mainly the possibility to address a single activity of a process and skipping technical configurations such as addressing concrete portTypes and operations. The participant topology shows a global perspective of collaborations. It introduces the construct "participant set" to allow the specification of an unknown number of participants of a certain type. A participant reference allows to point to a certain single participating process and so supports the so called link passing mobility. The message links wire the participant behavior descriptions together and thus define a collaboration. One of the main features of BPEL4Chor is that it avoids the tight coupling to WSDL and so improves the reusability of the choreography: all the web-service specific details are placed into the participant grounding which can be easily exchanged. BPEL4Chor, as well as Let's Dance, was designed to support of all Service Interaction Patterns, except Atomic Multicast Notification Pattern, which can be realized though a workaround. BPEL4Chor doesn't provide a graphical notation; [26], however, shows how BPMN can be extended towards modeling of BPEL4Chor choreographies.

Other Choreography Languages (Message Sequence Charts (MSC), BPSS, WSCI, WSCL)

Message sequence charts [27] describe sequences of simple process interactions. They do not provide more complex control flow constructs as parallel branching, iteration or conditional branching and therefore they are not well suited for modeling of more complex internal participant behaviors. Business Process Schema Specification (BPSS) [28], one of the early choreography languages, allows description of only bilateral conversations which limits the expressiveness of the collaboration description.

Web Services Choreography Interface WSCI [29] is an interface interconnection choreography modeling language. WSCI follows the bottom-up approach linking the existing web service interfaces (described in WSDL) into a conversation. WSCI doesn't support the top-down way of defining choreographies and focuses merely on application-to-application integration. The idea of WSCI is related to that of Web Services Conversation Language (WSCL) [30]. The disadvantage of WSCL is that is supports only bi-lateral interactions; compared to it WSCI provides more capabilities like support of contexts, correlations, exception handling and transactions.

2.3.2 Comparison

The criteria for categorization of the choreography modeling approaches introduced above are used in the following to compare the different languages:

- *Implementation-independent vs. Implementation-specific:* Lets Dance is the main representative of the implementation-independent interaction modeling approaches, the main strength of which

is providing a very high level global view on the collaboration which can be used by business analysts. Similarly, among the interconnected interface modeling approaches, BPMN is the leading implementation-independent approach. There is an extension to BPMN [26], which allows modeling of BPEL4Chor choreographies, thus bridging the gap between implementation-independent and implementation-specific views on the choreography. Implementation-specific languages are (at some point) grounded to Web services (WSDL). As the choreography is implementation-specific, the advantage of implementation-specific approached is that it is easier to derive orchestrations (implementations) for the involved roles which conform to the choreography.

- *Interaction Model vs. Interconnected Interface Model:* Main differences between the two approaches are the possibility to model locally unenforceable interactions in interaction models, and the possible incompatibility of the participants which can lead to deadlocks in interconnected interface models. Another difference is that it is easier to generate orchestrations from an interconnected interface model, as it already contains abstract processes for the different roles. These abstract processes have to be manually completed to executable processes. In case of an interaction model, however, these abstract processes have first to be generated.

2.4 Service Wiring

A service-based application typically comprises several services which communicate with each other. Thereby, each service provides an interface to other services, but also defines a requested interface of services it wants to use. In order to create an executable service-based application, one has to wire the requested and provided interfaces of server instances. The result is a service assembly, a deployable artefact, which is deployed on an enterprise service bus. Such an assembly is exposed to the outside as a service over a certain protocol, e.g. SOAP/HTTP. It can recursively be wired with other service assemblies.

Services specify provided and requested interfaces in form of operations with inputs and outputs, e.g. BPEL orchestrations use partnerlink types in order to specify provided and requested service interfaces. Inside the service assembly, the requested interfaces of one service is wired to provided interfaces of other services. The services assembled in this way can again be recursively exposed as a service which can be wired and invoked.

Service wiring enables also service chaining, whereby several service executions are executed one after each other by connecting the output of one service to the input of the next service and so on. A simple example is a service composed of a chain of filters, where each filter is a Web service. Wiring occurs by connecting the output of the first filter to the input of the next and so on. The input of the first filter and the output of the last one may then be exposed on the composition.

2.4.1 Service Component Architecture

Service Component Architecture (SCA) [31] is a set of specifications that provide a model for building composite applications based on a SOA. It is based on the idea that a service-based application comprises several services, which are assembled together to create solutions for implementing a particular business function.

SCA provides a model for both the creation of service components and composition of these components in composite applications. SCA supports a wide range of technologies for the implementation of service components and for the communication mechanisms which are used to connect them. SCA components may be implemented in programming languages such as Java, C++, or PHP, but also in BPEL. As communication mechanisms, SCA compositions support various communication and service access technologies that are in common use, including, for example, Web services, and Messaging systems such as JMS.

The SCA Assembly Model [32] is the main specification which defines the configuration of an SCA System in terms of Service Components which implement and/or use services and Ccomposites which describe the assembly of Components, including the connections between Components and describing how Services and References are exposed for use. The configuration of an SCA system has a standardized XML representation. That configuration can be seen as a deployment descriptor for the SCA System. The SCA System is deployed to an SCA Runtime.

The SCA Assembly Model consists of a series of artifacts. The main artifact is the Composite, which is the unit of deployment and which can be accessed remotely. A Composite contains one or more Service Components. Components offer their function as Services, which can either be used by other Service Components within the same Composite or which can be made available for use outside the module. Service Components may also depend on Services provided by other Components. These dependencies are called References. References can either be wired to services provided by other components in the same module, or References can be linked to Services provided outside the Composite, which can be provided by other Composites or arbitrary external services. All Service Components in the Composite are linked together, by connecting References with corresponding Services using Wires.

A Component consists of an implementation, where an implementation is the piece of program code implementing a business function (e.g., in Java or BPEL). In addition, Composites can again be used as Component implementations, in the same way as a Java classes or BPEL processes. SCA thus supports a recursive composite model, allowing a hierarchy of Composites that is arbitrarily deep. The Component can configure the implementation with specific values for settable properties declared by the implementation. The Component can also configure the implementation with wiring of References declared by the implementation to specific target Services.

Finally, for supporting non-functional properties such as security, SCA provides a Policy Framework [33] to support specification of constraints, capabilities and Quality of Service (QoS) expectations, from component design through to concrete deployment.

2.5 Service Coordination

Service coordination denotes a model in which a distributed activity is created by temporarily grouping a set of service instances following a coordination protocol. At the end of the activity, a coordinator decides on the outcome of the protocol (e.g., whether it is successful or aborted) and disseminates the result to the participating services.

Service coordination is much more loosely constrained than service choreography, as the participating services do not necessarily communicate with each other when performing their business logic. There is only a communication between the coordinator and the participant services, which takes place according to a coordination protocol.

The typical use case for the coordination model is the implementation of transaction processing. Thereby, coordinaton protocols are used for implementing conventional ACID transactions, but also long-running business transactions, where compensation is the preferred way in reaching a consistent outcome. However, also non-transactional use cases can leverage the coordination frameworks. [34] shows how coordination can be used for implementing auctions.

2.5.1 Web Service Coordination Approaches

WS-Coordination

WS-Coordination [35] defines an extensible framework for coordinating interactions between Web services. The framework enables participants to reach agreement on the outcome of activities using a coordinator and an extensible set of coordination protocols.

Coordinated interactions are called (coordinated) *activities*. An activity ties together several Web services into a (distributed) application. A participant is a web-service that contributes to an activity.

All participants of an activity must agree on an outcome of the activity. Therefore, a coordinator which mediates the agreement on the outcome is needed. Participants and coordinator exchange messages specified in a coordination protocol in order to agree on an outcome of the activity.

A coordination protocol consists of a set of messages and a specification of how these messages are to be exchanged. WS-Coordination does not specify concrete protocols, however allows the definition of arbitrary protocols. Two such protocol specifications already exist specifying transaction protocols, namely the WS-AtomicTransaction [36] for atomic (2PC) transactions and the WS-BusinessActivity [37] specification for long-running business transactions. Protocols in these specifications are defined as state charts. Each state change is communicated between coordinator and participants.

The WS-Coordination framework defines three services a coordinator has to provide: activation service, registration service, and protocol services. When an application, in the role of an initiator, wants to start a coordinated activity, it requests a coordination context from an activation service. The coordination context contains an activity identifier, the coordination type (e.g. atomic transaction) as requested by the initiator, and the endpoint reference of the registration service. When the initiator distributes work, it sends the coordination context with the application message to the participant Web service. Before starting work, the participant registers at the registration service of the coordinator. At some later point the protocol service is started which coordinates the outcome according to the specific protocol of the coordination type. While the logic of the coordination and registration service are fixed, the framework allows the definition of arbitrary coordination types, which specify different protocol services.

WS-CAF

The Web Services Composite Application Framework (WS-CAF) is another coordination framework for Web services. It is based on three specifications: (i) WS-Context (WS-CTX) [38], (ii) WS-Coordination Framework (WS-CF) [39], and (iii) WS-Transaction Management (WS-TXM) [40, 41, 42].

WS-CTX provides a mechanism for creating and managing activities and contexts belonging to a set of Web services. A context is needed for grouping several Web services in an activity. WS-CTX defines three components: (i) a Context Service: which creates and manages activities and corresponding contexts. (ii) a context which contains information on an activity. A context contains following attributes: a context-identifier as URI; activity-service, which idenifies the corresponding context service; activity-list which contains all services participating in the activity; child-contexts which contains all child-activities; timeout which defines how long the context is valid. (iii) Activity Lifecycle Service: Web services which want to participate in an activity register an activity lifecycle service at the context service. The context service then informs the lifecycle service on the state of the activity.

WS-Coordination Framework (WS-CF) supports coordination of an activity consisting of several Web services. Those Web services belong to a context as defined in WS-CTX. WS-CF defines the following components: (i) a coordinator which provides an interface for registration of participants. The coordinator coordinates the registered participants according to a protocol and disseminates the outcome of the protocol to the participants. (ii) A participant can be a single Web service, or an activity, i.e. a group of Web services. (iii) a coordination service which provides an Activity Lifecycle Service, which registers at the Context Service of WS-Context. A coordination service implements the behavior of a certain coordination protocol, as defined e.g. in WS-TXM. WS-TXM defines three transaction protocols, which can be integrated into WS-CF. Those are (i) WS-ACID for managing ACID-transactions, WS-LRA for long-running business transactions and WS-BP for mixed business transactions.

2.5.2 Comparison

WS-Coordination and WS-CF have many concepts in common. They both support the same coordination model, in which a coordinator manages the agreement protocol among several participants. Both approaches enable the definition of arbitrary coordination protocols, and already provide protocols for ACID transactions and long-running business transactions.

WS-CF is based on another specification, i.e. WS-CTX. WS-CTX defines mechanisms to share a context used for identifying an activity. WS-CTX aims at applications that simply need a shared context. It defines a Context Service, which is responsible for managing shared contexts and provides interfaces to access these contexts. Thus it is possible for a client to pass only a subset of the context to participants, which does not contain all information necessary for participants to get involved in the activity, but only points to a Context Service, where the information can be retrieved. This mechanism can be useful when a context contains a large amount of data. This feature is not possible in WS-Coordination, as there is no separate context service. WS-Coordination defines the management of contexts and the coordination mechanisms together.

In contrast to WS-Coordination, WS-CF specifies generic coordination interfaces which enable querying of the status of a coordination. If such interfaces are needed for a particular coordination protocol on top of WS-Coordination, the protocol has to define such interfaces separately.

In summary, while WS-Coordination and WS-CF tackle the same problem domain and have the same coordination models, WS-CTX and WS-CF introduce more generic definitions concerning the architecture of the coordination framework and the communication within the coordination framework.

2.6 Semantic WS Composition

Whilst promising to revolutionize e-Commerce and enterprise-wide integration, current standard technologies for Web Services (WSs) (e.g. WSDL) provide only syntactic-level descriptions of their functionalities, without any formal definition to what the syntactic definitions might mean. In many cases, WSs offer little more than a formally defined invocation interface, with some human oriented metadata that describes what the service does, and which organization developed it (e.g. through UDDI descriptions). Applications may invoke WSs using a common, extendable communication framework (e.g. SOAP). However, the lack of machine readable semantics necessitates human intervention for automated service discovery and composition within open systems, thus hampering their usage in complex business contexts.

Semantic Web Services (SWSs) relax this restriction by augmenting WSs with rich formal descriptions of their capabilities, such that they can be utilized by applications or other services without human assistance or highly constrained agreements on interfaces or protocols, thus facilitating automated composition, discovery, dynamic binding, and invocation of services within an open environment. A prerequisite to this, however, is the emergence and evolution of the Semantic Web (SW), which provides the infrastructure for the semantic interoperability of WSs.

Semantic annotation of WSs descriptions has been the issue of many initiatives, projects and languages introduced, the most significant among them are: the OWL-S Semantic Markup for Web Services [43], the Web Service Modeling Ontology (WSMO) [44] and the Semantic Web Services Ontology [45]. In this section, we will analyze these SWS languages in terms of how they enable the automatic composition of WSs. In addition, we will also analyze an effort to extend the BPEL standard WS composition language with ontological annotations. Finally, we will compare all of these languages based on their WS composition capabilities.

2.6.1 Main Semantic Web Service Composition Languages

OWL-S

OWL-S consists of a set of ontologies designed for describing and reasoning over service descriptions. It consists of three main upper ontologies: the *Profile*, *Process Model* and *Grounding*. The *Profile* is used to describe services for the purposes of discovery; service descriptions (and queries) are constructed from a description of functional properties (i.e. inputs, outputs, preconditions, and effects - IOPEs), and non-functional properties. In addition, the profile class can be subclassed and specialized, thus supporting the creation of profile taxonomies which subsequently describe different classes of services.

OWL-S process models describe the behavioral interface of a services in terms of a process model and are not executable, similarly to an abstract BPEL process. This is used both for reasoning about possible compositions (validation, verification, etc) and controlling the enactment/invocation of a service. Three process classes have been defined: the *composite*, *simple* and *atomic* process. The *atomic* process is a single, black-box process description with exposed IOPEs. *Simple* processes provide a means of describing service or process abstractions which can be either realized by an atomic process (e.g. through service discovery and dynamic binding at run-time), or expanded into a composite process. *Composite* processes are hierarchically defined workflows (consisting of atomic, simple and other composite processes) constructed using a number of different composition constructs.

The profile and process models provide semantic frameworks whereby services can be discovered and invoked, based upon conceptual descriptions defined within OWL ontologies. The *grounding* provides a pragmatic binding between this concept space and the physical data/machine/port space, thus facilitating service execution. The process model is mapped to a WSDL description of the service, through a thin grounding. Each atomic process is mapped to a WSDL operation, and the OWL-S properties used to represent inputs and outputs are grounded in terms of XML data types.

WSMO

WSMO (Web Service Modeling Ontology) is a conceptual model for describing semantically-enhanced WSs. The WSMO framework consists of four main modeling elements: *Ontologies*, *Web Services*, *Goals* and *Mediators*. *Ontologies* provide terminologies for defining the formal semantics of information resources. WSMO ontologies are classic ontologies with classes, relations, instances and axioms, plus functions (n-ary relations) and some additional information coded as non-functional properties (QoS and Dublic-Core-based).

WSMO *Web service* descriptions consist of functional, non-functional and the behavioral aspects of a Web service. The functional aspect of a WSMO WS is described through one *capability* using similar concepts with OWL-S IOPEs. A capability is linked to several goals via mediators.

An interface (behavioral aspect) describes how the functionality of the WS can be achieved by providing a twofold view on the operational competence of the WS: a) *choreography* and b) *orchestration*. WSMO allows more than one interface to be defined, so multiple choreographies can be defined for a WS and hence multiple modes of interactions may exist for the same WS. The conceptual model for WSMO choreography is based on Abstract State Machines (ASMs) [46], defining a set of states where each one is described by an algebra, and guarded transitions to express state changes by means of updates. A WSMO choreography specifies a behavioral interface of the service, and not a complete choreography model. Similarly to WSMO choreography, WSMO Orchestration also uses ASMs. The difference with respect to choreography is that the WSML rules are extended to support the use of mediators to link the orchestration to other goals or WSs.

Goals are limited descriptions of WSs (a capability and a interface) that would potentially satisfy the user desires. WSMO model definition follows a goal-driven approach, which means that requests and services are strongly decoupled. Finally, *Mediators* connect heterogeneous components of a WSMO description which have structural, semantic or conceptual incompatibilities. Mismatches can arise at the data or process level. There exist four types of mediators: ontology, goals, WS-to-goal and WS-to-WS mediators.

SWSO

The Semantic Web Services Framework (SWSF) is the proposal from the Semantic Web Services Initiative (SWSI) for the development of SWSs. SWSF model includes two major components, the Semantic Web Services Language (SWSL) and the Semantic Web Services Ontology (SWSO). SWSL is a general-purpose logical language and its main objective is act as the basis of the SWSO ontology. SWSL defines two sub-languages, SWSL-FOL (First Order Logic SWSL), which is intended to be used mainly for

process-ontology definition, and SWSL-Rules, a rule-based sub-language to support reasoning over first order logic ontologies.

SWSO defines the conceptual model for defining SWSs in SWSL. SWSO ontology, expressed in SWSL-FOL, is called FLOW-S (First-order Logic Ontology for Web Services). FLOW-S includes mechanisms from PSL (Process Specification Language) [47] family of languages for describing process workflows. The structure of FLOW-S is very similar to OWL-S and also consists of three main components: *Service Descriptors*, *Process Model* and *Grounding*. Service Descriptors provide basic information about a WS to be used during service discovery with client preferences.

The Process Model adapts PSL and is created by extending PSL-Outer Core through: a) *Ordering Constraints*, which are simple constraints based on sequencing properties of atomic processes; b) *Occurrence Constraints*, which are constraints involving activities that must take place; c) *State Constraints*, used for associating activities which are fired when a condition is satisfied and d) *Exceptions*, a major component of the ontology for modeling exception handling through state constraints, messages, or other extended approaches.

Finally, the Grounding facilities allow to declare the low-level details of WSs definitions: message formats, transport protocols, etc. SWSO implements mechanisms to map high-level ontology descriptions with WSDL.

BPEL4SWS

The WS stack of standards currently relies on BPEL for executing orchestrations of WSs. BPEL is using WSDL descriptions to identify partner/component services in the process models. Unfortunately, effective dynamic service binding cannot be performed by solely matching WSDL messaging interfaces. Service partners should be selected based on functional, nonfunctional and behavioral descriptions of what a service does, and how it does it. Restricting service descriptions to the expressivity of strictly syntactic WSDL interfaces limits the integration of service partners that operate on messages that have different syntax but are semantically compatible.

The solution to the dynamic selection of WSs comes with SWSs. SWSs describe services not in terms of an interface but rather describe their functionality and capability semantically and in a machine processable manner. For this reason, SWSs increase the level of integration and can be considered an integration layer on top of WSs where services are discovered based on their functionality and not based on their signature.

Based on the above analysis, a new language called BPEL4SWS [48, 49] was introduced that extends BPEL. This language uses BPEL^{light} [16] as basis and allows to attach SWS descriptions to BPEL^{light} such that SWS frameworks like OWL-S and WSMO and corresponding implementations can be used to discover and select SWSs that implement the functionality required for an activity. Actually, SWS descriptions can be both, encoded in the process model as well as annotated using WS-PolicyAttachment [50]. As WSMO distinguishes between services provider (Web Service) and service consumer (Goal) and thus enables asynchronous communication between WSs, it is preferred to OWL-S. Both, the SWS description and the process itself are partly grounded to WSDL to facilitate WS-* based communication. BPEL4SWS enables the use of SWSs as well as traditional WSs. Hence, BPEL4SWS processes are able to use both semantic WSs and conventional WSs intermixed within a single process. BPEL4SWS processes themselves can be exposed both as SWSs and conventional WSs.

Current SWS frameworks use ontologies as data model to facilitate semantic discovery. For that reason, SAWSDL [51] is used to enable a seamless mapping of data between its XML representation and its ontological representation which is also needed because in BPEL4SWS WSs and SWSs can be used intermixed.

2.6.2 Comparison

WS composition addresses the situation when a client request cannot be satisfied by a single pre-existing service, but can be satisfied by suitably combining some available, pre-existing services. Two aspects of composition are identified: *composition synthesis* is concerned with synthesizing a specification of how to coordinate the component services to fulfill the client request; and *orchestration*, is concerned with how to actually achieve the coordination among services, by executing the specification produced by the composition synthesis and by suitably supervising and monitoring that execution. In this section, we will compare all presented approaches first on their discovery capability and second and most importantly on their composition (synthesis and orchestration) capability. However, we should state that we are not going to compare the SWSF approach, although it is quite promising and powerful, as it still in its infancy and no interesting research results accompany it.

From all SWSs languages, OWL-S is the most mature one and has been used in many WS discovery research approaches that include both functional and non-functional characteristics of WSs. However, it has been proven that OWL-S presents some drawbacks [52] that prevent it from being used in practical real-world scenarios. Concerning composition synthesis, OWL-S has been successfully used in AI planning [53] for automatically producing and executing WS composition plans. Moreover, it can be used for the analysis and validation of WS compositions as it can be mapped to (colored) Petri-nets [53, 54]. Unfortunately, as far as orchestration is concerned, OWL-S presents two major drawbacks. First of all, its process model is neither an orchestration model (although it resembles one) nor a choreography model. It just described the behavioral interface of the service. Secondly, OWL-S describes the execution of a Web service as a collection of remote procedure calls. Unfortunately, this is only correct for a small percentage of the cases in business processes [48] since typically the communication is asynchronous.

WSMO has the same discovery capabilities as OWL-S and has also been used in many WS discovery research approaches. It is starting to attract many SWS researchers as there exist a lot of SW tools that support it. Concerning WS composition, WSMO contains a behavioral interface model that can be easily mapped to ASMs. As the WSMO Choreography is based on ASMs, it inherits the core principles of ASMs which are: (1) it is state-based, (2) it represents a state by an algebra, and (3) it models state changes by guarded transition rules that change the values of functions and relations defined by the signature of the algebra. However, the current transition rules in WSMO Choreography can only represent local constraints. In addition, WSMO Choreography needs to be formalized in order to allow reasoning and verification about it. Moreover, WSMO contains an orchestration model which is quite primitive. So, WSMO can only be used in a limited way to orchestrate the interactions of WS-based processes. In addition, to the best of our knowledge, none research approach for automatic WS composition synthesis and analysis involving WSMO has been realized, yet. Thus, although WSMO is getting more researcher attraction than OWL-S, its lack of strong capabilities in WS composition confute its advantages.

BPEL's lack of dynamic semantic-based service binding led to the synergy with WSMO and SAWSDL. The result, BPEL4SWS, is not actually a SWS language but a XML-based language with semantic annotations, just like SAWSDL. However, the goal of dynamic service binding support has been achieved. Now, BPEL4SWS in comparison to OWL-S supports orchestration. Concerning the composition synthesis capability, there are many research approaches that have been applied to BPEL. So BPEL4SWS seems to be the right language for the WS composition process. It remains to be accompanied by a full-fledged composition engine that will automatically compose, analyze, execute and monitor WS-based business processes. In our opinion, another possibility is to extend BPEL with OWL-S and SAWSDL, especially if the synergy with WSMO does not go well and OWL-S specification is altered to allow asynchronous business process conversations.

2.7 Combined Service Composition Models

The composition models defined above constitute different approaches to service composition; however, they can be combined in different manner. In this section, we examine the different combinations.

Orchestration and Choreography

Service choreography specifies the interaction protocol between two or more partners. In case of an Interconnected Interface Model approach, for each partner only the public process is defined. A service orchestration on the other hand specifes the private process of each partner and is executable.

Considering a development approach, one can start with modeling a choreography, and then refine the choreography to orchestrations of different partners. In case of an Interconnected Interface Model, such as BPEl4Chor or WS-Global Model, the natural way to do this, is to refine the abstract public processes to private processes, i.e. service orchestrations for each partner. As the language is the same, BPEL and WSFL, this is relatively easy. In case of an interconnection model, such as WS-CDL, the orchestrations can be generated to some point and then further manually refined, e.g. in [55] an approach for mapping of WS-CDL to BPEL is shown.

Orchestration and Wiring

The service wiring model provides a mechanism for wiring service components together by linking their requested and provided interfaces and deploying the resulting service assembly as an executable artifact on a service bus. A service orchestration, defined in a WS-flow language, specifies an interaction with one or more partner Web services. The required interfaces of orchestrated Web services are specified in terms of WSDL port types and operations. The service orchestration also provides such an interface to service requesters.

A service orchestration is thus a service component which can be part of a service assembly. Typically, its provided interface would be exposed over one or more protocol bindings (e.g., SOAP/HTTP) to the requesters, and the requested interfaces would be wired to corresponding partner service instances.

SCA defines a BPEL implementation model for service components which are implemented as BPEL orchestrations [56]. It defines how BPEL concepts are mapped to SCA concepts and thus can be exposed as SCA components as part of an SCA composite. Thereby, BPEL partner links are mapped to SCA services and references, respectively. In case of partner links which consume other services, the SCA references are wired to those services. The partner link the BPEL process exposes to service requesters is exposed as an SCA service and can be wired to SCA references of service requesters or be exposed over a protocol binding to external requesters. SCA thus provides a mechanism to deploy BPEL processes on a service bus.

Orchestration and Coordination

The ability to coordinate multiple Web services can be extended to coordination between and within service orchestrations. Service orchestrations implementing business processes are often long-running. Therefore, BPEL employs a long-running transaction model to handle the internal transactions of a WS-BPEL process. WS-Business Activity (WS-BA) provides a set of mechanisms and protocols to coordinate a set of Web Services into a long-running compensation-based transaction. [57] shows how (an extended) WS-BA allow parts of a WS-BPEL process to participate in a WS-BA coordination, which is supervised by an external coordinator. In addition, the approach allows external partners to participate in these modified internal WS-BA transactions initiated by a WS-BPEL process and also allows for easy incorporation of BPEL sub-processes into the proposed coordination model.

Chapter 3

Service Composition Approaches

3.1 Introduction

In this chapter we describe how service composition models are created (developed). Thereby, we concentrate on the following three main approaches: (i) model-driven service composition, (ii) automated service composition, and (iii) QoS-aware service composition.

Model-Driven Service Composition copes with generating service orchestration models from more abstract models. The goal of model-driven development is to transform a platform-independent model (PIM) into a platform-specific model (PSM). In the context of service orchestration, one approach is to model the service orchestration by using a model such as UML activity diagram and then specify a transformation to executable workflow models, such as BPEL. The benefit of this approach is that the service orchestration is specified in a platform independent way, and thus could be transformed to different platforms and WS-Flow languages if needed. A related approach is business driven development. In that case one starts with abstract business process models, which are first modeled by business analysts, e.g. in EPC or BPMN. Those models can be both choreography and orchestration models. Business process models are then transformed to executable WS-Flow models, whereby typically both manual and automatic transformations are needed.

One can further distinguish between *manual* and *automated service composition*. Manual composition is created by a user, typically by an IT engineer who creates the control flow and selects the services. In automated service composition, the services are selected and orchestrated automatically based on an abstract goal specification.

The composition can futher happen at design time or at runtime. In the extreme case that only one service is composed, service selection is performed. At runtime, that boils down to selecting and dynamically binding the service to the service orchestration. Automated runtime composition enables runtime adaptability of service compositions.

When creating a service composition and selecting appropriate services, the first concern is to find services which implement the functionalities needed, and define a control flow which specifies in which order these services are to be executed. Besides functional composition, one is often also interested in QoS-properties of the composition, i.e. which QoS properties does the overall service composition posess, e.g. cost or duration. To determine these properties, the properties of the orchestrated services have to be aggregated. If for each needed functionality in the service composition, there are several alternative services with different QoS properties, then *QoS-aware service composition* can be used to select those services which satisfy local and global QoS constraints.

3.2 Model-Driven Service Composition

System theory suggests to use model-driven development (MDD) in order to raise the level of abstraction to cope with increasing complexity of systems [58]. In contrast to traditional engineering approaches, the primary entity of MDD are higher-level models. These models are then compiled into executable code (often using automated or semi-automated tool chains). The Model-Driven Architecture (MDA) is one possible approach to implement MDD. MDA has been put forward by the Object Management Group (OMG) in 2001, and has since then been seen as the de facto standard for model-driven software engineering. In MDA, all software functionality is specified using platform-independent models (PIMs). Then a platform definition model (PDM) is used to translate the PIM to various platform-specific models (PSMs), which can either be compiled into executable code or run directly.

Arguably, a similar approach can also be used to tackle service composition challenges and issues [59].

3.2.1 Model-Driven Development of Web Services

Model-driven development of Web services and Web applications has been around since about 2000, when Ceri et al. first published their work on WebML [60]. WebML is a modelling language that allows to specify data-intensive Web applications in an abstract high-level language, using the definition of a data schema (basically a standard entity relationship schema), and a number of hypertext schemas. Hypertext models (site views) express the user interfaces that allow accessing and modifying the data. Furthermore, site views are further partitioned into pages. Pages can be seen as collections of data fragments, which in turn contain pointers into the data model of an application. Pages in a site view are connected using links.

The WebML model has prominently been implemented in the WebRatio CASE tool [61].

Using current terminology, one can argue that the work of Ceri et al. provides a graphical and model-driven approach to the development of RESTful Web services (i.e., Web services that follow the Representation State Transfer architectural style defined by Fielding [62]).

Later on, Manolescu et al. have extended WebML to consider not only typical REST-style resources, but also WSDL-based Web services [63]. However, the scope of this work is not so much in the modelling of service compositions, but rather the integration of services in standard Web applications (even though they still ”compose” services using WebML site views).

3.2.2 Approaches to Model-Driven Composition

Early work in the area of Web service composition models has seen the adaption of traditional process algebras for Web service composition. One example of such research is the work of Hamadi and Benatallah [64], who used Petri Nets to construct Web service orchestrations with specific, formally verifiable properties.

Other researchers have analyzed languages and models on higher levels of abstraction (such as the UML, or BPMN) and found that these models, while being easier to write and interpret, can also be used to express the most important flow patterns [65]. Therefore, these languages are a natural choice for researchers aiming at bridging the gap between business workflows and models on the one hand, and technical service orchestrations on the other. Following the MDD naming, the approach of specifying compositions on business model level is called Business Driven Development (BDD) [66].

Skogan et al. have used UML as a language to model Web service orchestrations [67]. This paper proposes a method that utilizes UML Activity models to define compositions, and an MDA-based approach to generate executable code in varying composition languages. Their initial prototype supports the generation of both WS-BPEL and WorkSCo[1] executable code. Their approach endorsed a complete roundtrip model: existing WSDL specifications are refactored into UML models, which in turn are

[1] http://www.esw.inesc-id.pt/worksco/

arranged to create new service orchestrations; these orchestrations can then be transformed into executable code (either WS-BPEL or WorkSCo), and deployed on a workflow engine. That is, they use UML as an integration platform, which is independent of the actual execution environment. The actual transformation of high-level UML models to workflow code can be handled using XSLT transformations: this is possible since both WS-BPEL and WorkSCo are represented using XML.

The approach of Skogan et al. has later been extended to also consider semantic Web service composition and QoS attributes [68]. The main advantage of this extension is the increased support for run-time discovery of services and run-time service binding: in [67] the service bindings (the partner links in WS-BPEL notion) are defined at design-time, while [68] considers abstract bindings to specific semantically defined functionality; concrete services implementing this abstract functionality are discovered at run-time using semantic discovery and matchmaking technologies. Additionally, services are ranked based on QoS properties, i.e., if more than one service instance is discovered during run-time the one exhibiting the best QoS properties is selected. Unfortunately, the work is based on a set of assumptions (e.g., the existence of a semantic matchmaking and service discovery entity) which are not fulfilled today.

Already in 2003, Koehler et al. have presented an approach that is more clearly aligned to the OMG MDA [69] and BDD. Their work presents the transformation of business models (represented in ADF and UML) to technology-level WS-BPEL code. Unlike other approaches, their work considers not only the top-down transformation from high-level models to code, but also the other direction: their work also considers the bottom-up re-engineering of existing WS-BPEL compositions into business processes. Their transformation architecture consists of four levels: business models are firstly transformed to process graphs, which are in turn refined to flow graphs; these subflows can then be compiled to solution components using platform-specific transformation (synthesis). In the reverse direction (bottom-up), solution components are combined to create flow graphs, which are again merged to process graphs, and finally the business model can be restored. Later on, the same authors have refined their ideas [70]. In this paper they detail how graph-based process models can be transformed into executable code using graph transformation and compiler theory.

A conceptually similar model-driven and MDA-based methodology for service composition has been introduced by de Castro et. al [71]. Their paper presents MIDAS, a MDA framework that proposes multi-level modeling of service compositions: as defined by the Object Management Groups MDA, composite services are specified using Computation Independent Models (CIMs), which are mapped (transformed) into Platform Independent Models (PIMs), which in turn are again mapped into Platform Specific Models (PSMs). In MIDAS, PSMs are represented using WS-BPEL. PIMs contain User Service Models, Extended Use Case models (both are derivations of standard UML Use Case models), service process models and service composition models (both represented using UML Activity models).

Ouyang et al. use a different language, OMGs BPMN, to represent business models [72]. They argue that BPMN is (even though the language is relatively young) more common in the business world than the more software-centric UML. Additionally, BPMN is well supported by business analyst tools. However, the authors state that the transformation from BPMN is complex since BPMN is (as a graph-oriented language) fundamentally different than the block-oriented WS-BPEL. However, they argue that their approach still is complete, fully automated and produces WS-BPEL code that is comprehensible for humans.

Orriens et al. present a model driven service composition approach based on composition rules [73]. These rules are expressed in OCL (the Object Constraint Language), and are used to model constraints on possible compositions. Examples of such rules are `activity.function = "FlightBooking"` (i.e., the function of the activity always has to be ”FlightBooking”) or `activity.input -> notEmpty` (i.e., there always has to be an input message for this activity). Generally, they distinguish between structural rules (structuring activities with the composition), data rules (rules that relate messages to each other), behavioral rules (rules that guard the integrity of the composition, and preconditions, postconditions and invariants of activities), resource rules (rules that guide the usage of resources) and exception rules (rules that specify the behavior of the composition in exceptional cases). When all

necessary rules that apply for a composition are specified, an automated dynamic service composition engine is used to create an orchestration that satisfies the rules specified by the model. However, of course this approach implies that the composition has to be fully specified using business rules.

Another form of model-driven service composition is mapping WS-CDL choreographies to concrete partner WS-BPEL orchestrations. This approach is discussed e.g. by Mendling and Hafner [55]. WS-CDL defines the externally visible behavior of business entities, i.e., which messages are transferred to which business partner at which point in time, and what options are available. Additionally, WS-CDL relates operations to states in the business processes. Mendling and Hafner propose to use the inter-business WS-CDL model as a model to generate WS-BPEL code for each specific participating business partner. They show that many WS-CDL constructs can be mapped directly (`cdl:sequence`, `cdl:parallel`, `cdl:assign`, ...), e.g., using XSLT transformations, but others are more complex to map: for example, it is not easy to identically map the semantics of a `cdl:workunit` construct. In this case, the authors propose to map the construct to `bpel:scope`, but a human BPEL engineer is required to eventually add additional information that cannot be deduced from the choreography automatically. In general, this approach can be considered only semi-automated – a fully autonomous mapping from WS-CDL to WS-BPEL without human interaction is not possible.

The idea of generating WS-BPEL templates from WS-CDL choreographies has been further extended by Rosenberg et al. [74]. In their work, they have adapted the rules presented in [55] and proposed a so-called relevance mapping. Their idea is to map only those WS-CDL constructs which are directly relevant to the single BPEL processes. Another important property of their work is the consideration of Service Level Agreements (SLAs) and QoS attributes at WS-CDL level: choreographies are annotated with SLA references using IBM's WSLA, which are then mapped to WS-Policy QoS policies in the generated WS-BPEL files (WS-Policy QoS policies are a WS-Policy extension defined within this research work).

The presented approaches so far aim to generate service orchestration models, either from high-level business process models or service choreography models. In contrast to those works, in [75] an approach is shown how service coordination models can be created in a model-driven manner. Thereby, a so called coordination protocol graph which models the states and state changes of the coordination protocol is transformed to two BPEL process models, one for the coordinator role and one for the participant role, respectively. WS-Coordination is used as the coordination framework. Both generated process models contain in particular messaging activities which are needed to propagate state changes of the protocol between coordinator and participant. The generated process models are not yet executable; some of the protocol logic has to be manually added by the developer. Those places in the BPEL code are marked during transformation as BPEL "opaque" activities.

The presented approaches are summarized in Table 3.1. In the table, "n\a" means that the column is not applicable for a certain research approach (e.g., the work by Hamadi and Benatallah [64] can neither be considered top-down nor bottom-up, since the work does not consider transformations). "Not Explained" represents properties which are not mentioned in the concerning paper. In the table we differentiate between pure MDA approaches, and methodologies which are based on modified MDA.

Paper	**Methodology**	**Business Models**	**Composition Models**	**Top-Down Transf.**	**Bottom-Up Transf.**	**QoS**	**Semantics**
Hamadi and Benatallah [64]	n\a	n\a	Petri Nets	n\a	n\a	No	No
Skogan et al. [67]	MDA-Like	UML Activity Diagrams	WS-BPEL and WorkSCo	Yes	No	No	No
Gronmo et al. [68]	MDA-Like	UML Activity Diagrams	Not Explained	Yes	No	Yes	Yes (OWL-S, WSML)
Koehler et al. [69]	MDA	ADF and UML Activity Diagrams	WS-BPEL	Yes	Yes	No	No
MIDAS [71]	MDA	UML Class, Use Case and Activity Diagrams	None	Yes	No	No	No
Ouyang et al. [72]	Not Explained	BPMN	WS-BPEL	Yes	No	No	No
Orriens et al. [73]	Not Explained	Business Rules	None	n\a	n\a	No	No
Mendling and Hafner [55]	Not Explained	WS-CDL	WS-BPEL	Yes	No	No	No
Rosenberg et al. [74]	Not Explained	WS-CDL and WSLA	WS-BPEL and WS-Policy	Yes	No	Yes	No
Kopp et al. [75]	MDA-Like	Coordination Protocol Graph	WS-BPEL and WS-Coordination	Yes	No	No	No

Table 3.1: Comparison of Research Approaches

3.2.3 Products and Tools

IBM Alphaworks has released a UML-to-BPEL mapping tool as part of the Emerging Technologies Toolkit (ETTK) [76]. The tool uses a specific UML profile for automated business processes that has previously been published by Gardner [77]. In this profile processes are defined using the `<<Process>>` stereotype, while an activity model is used to represent the process itself. In this respect, this approach is very similar to the research work of [67]. ETTK includes an Eclipse plugin that can be used to create deployable WS-BPEL code from an XMI model annotated according to the UML profile for automated business processes. However, ETTK does not include an end-to-end development environment: the actual preparation of the annotated UML model has to be carried out in a standard UML tool such as Rational Rose (which lacks specific support for the necessary UML stereotypes and annotations). Furthermore, the ETTK has been discontinued by IBM Alphaworks in the meantime.

Finally, the IBM research regarding model-driven service composition has been further refined into a full production-quality tool chain [78]. The main components in the tool chain are the WebSphere Business Integration Modeler, which is used by business analysts to capture and simulate business processes, the WebSphere Business Integration Server, which stores and distributes standards-based artifacts which are generated from the business model, and the WebSphere Application Developer IE, which is used by the application developer to refine the generated artifacts into runnable and deployable code. During the transformation, business tasks are mapped to WSDL operations and control structures in the business process are transformed into respective BPEL structures. However, for the transformation to succeed a number or constraints need to be taken into account when modeling the business process (e.g., tasks are not allowed to be connected ”in loops”, tasks are not allowed to have more than one output criterion). Additionally, the mapping will usually only be semi-automatic – after the transformation the developer needs to refine the generated artifacts in order to produce the actual deployable BPEL process. The reason for this limitation is in the fact that not all information that is necessary on technical level is already contained in the business process description. Recently, the WebSphere Business Integration Modeler has been succeeded by the IBM Business Modeler, and WebSphere Integration Developer.

3.3 Automated Service Composition

A major family of approaches to Web service composition aim to fully or partially automate the process of composition in order to deal with its high level of complexity. Manual implementation of a composition schema is a time-consuming, error-prone and generally hard procedure and is most certainly not scalable. Therefore, the need for automation in Web service composition should be apparent.

In [79], the authors propose a general framework for automatic Web service composition. Through this framework, one can extract five distinct phases that constitute a complete automatic composition approach. These phases are: description of atomic services, description of internal processes of services, matching of requirements, composition evaluation and composition execution.

Atomic services description The first phase deals with the description and advertisement of atomic services that will be used as building blocks for the composition. Service advertisement can be achieved using UDDI or using the Service Profile class of OWL-S [43], which provides semantically annotated descriptions for Web services.

Internal processes description Having dealt with the issue of creating specifications to describe how services are represented externally, or in other words, what the service offers, a composition approach then must focus on service specifications that describe the internal processes of the service, essentially specifying how the service works. Formal and precise languages such as logic programming are required in this case in order to effectively capture the constraints describing the internals of a service. This kind of description is necessary in order to be able to decide how to compose different services and how their

operations will interact. Translators may need to be designed in order to bridge the gap between service specifications used in the previous phase and those defined in this phase.

Generation of composition process model The next phase is the heart of the composition process, as it involves generating the composition process based on the requester's requirements. The requester may express his requirements in the form of a service specification, effectively describing the one service that, if available, there would be no need to compose an equivalent from atomic services. In general, automated composition approaches vary in the way requirements are expressed, from desired effects to complex constraints and invariants. This phase should eventually produce a process model that describes which services participate in the composition as well as the control and data flow among the participants. If more than one composition schemas satisfy the requester's requirements, then this phase will produce multiple results.

Composition evaluation and execution The penultimate step in an automatic composition framework and the final before executing the composite service is to evaluate the different composite services from the previous phase in order to select one of them, according to a set of non-functional attributes such as QoS aspects. These non-functional requirements may be explicitly stated by the requester or they may be assumed by the framework. At the end of this phase, a single composite service is selected, which is then executed in the final phase. Service execution involves executing the services that constitute the composite service according to the process model produced in the third phase. The composite service can be executed only once, or exposed as a new service that can be discovered and reused, possibly as part of other composite services.

Automatic service composition approaches can be grouped into two distinct categories, workflow-related approaches and AI planning techniques. The first group draws mainly from the fact that a composite service is conceptually similar to a workflow, making it possible to exploit the accumulated knowledge in the workflow community in order to facilitate Web service composition. Workflow-related techniques may either provide a static composition scheme, meaning that only selection and binding of atomic Web services is performed automatically or a dynamic one, where the process model creation is also automatic.

On the other hand, AI planning techniques involve generating a plan containing the series of actions required to reach the goal state set by the service requester, beginning from an initial state. All approaches in this family rely on one of the many planning techniques that the AI community has proposed and incorporates it in the process model creation phase of the composition framework.

3.3.1 Workflow Techniques

Composition frameworks based on workflow techniques were one of the initial solutions proposed for automatic Web service composition. In [80], the authors present EFlow, a framework for specifying and managing service compositions. In EFlow, a service composition is represented by a graph containing three different kinds of nodes. Service nodes represent that a service is invoked, either an atomic or a composite one. Decision nodes are used to describe the execution flow, while event nodes represent the exchange of events between services. The graph is constructed manually, thus EFlow is considered a semi-automatic composition platform. However, the binding of concrete services to the abstract service nodes is done automatically and can be modified dynamically. Service nodes contain code that enables a search to be performed each time the flow leads to a particular node. As a result, the composition schema adapts dynamically to any unexpected changes in the availability of services.

A work contemporary to EFlow, introduces a Polymorphic Process Model (PPM) [81] for processes that span across multiple enterprises. Like EFlow, there are abstract processes that are binded to actual service implementations at runtime. Unlike EFlow, PPM doesn't use graphs to represent composite services, but rather employs state machines, specifying all possible states of a service and the transitions from state to state, which may be due to execution of service operations, service invocation or they may

be refer to internal transitions of a single service. State machines will be revisited in the next subsection, as they are also used in AI planning composition techniques.

In [82], the authors define Transactional WorkFlow Ontology (TWFO), an ontology used to describe workflows. Based on TWFO, they build an automatic composition framework, which aims to create a composite workflow based on component workflows described using TWFO. In this framework, a registry containing OWL-S service descriptions is used to find services corresponding to the tasks in the composite workflow. Then, the workflows for each service are read from a TWFO registry. Afterwards, ordering constraints are applied and the control and data flow is constructed, in order to create and execute the composite workflow.

In contrast to the previous work, where concrete workflows associated to services are combined to a master workflow, [83] and [84] deal with abstract workflows. [83] argues that business process flow specifications should be abstractly defined and only bound to concrete services at runtime. The authors present a BPEL engine that is able to take an abstract workflow specification and dynamically bind Web services in the flow taking into consideration any relations and dependencies between services as well as domain constraints. This work uses planning in order to generate concrete process flows from the abstract ones and, in that sense, is also related to the techniques that will be presented in the next section. [84] advances one step further by allowing for the automatic generation of an abstract workflow based on a high-level goal. An integrated Reasoning Service aims to create a combination of available services that provide the required functionality to satisfy the goal, based on inputs, outputs, preconditions and effects. Then, as in all previously mentioned frameworks, the abstract workflow tasks are matched with concrete deployed services. If no service is found for a task, the Reasoning Service along with a Matchmaking Service attempt to compose existing services in order to create a service that is match for that particular task. The matching process results to an executable graph for the composition.

Finally, [85] deals with the security dimension of workflow composition by constructing a workflow planning model that takes into consideration certain security policies. An algorithm for efficient workflow planning under such circumstances is devised. This work is a bridge between traditional workflow composition methods that were presented in this section and techniques based on AI planning that will be presented in the next sections.

3.3.2 AI Planning Techniques

AI research has extensively covered a complex problem known as planning. A classic definition of planning states that it is a kind of problem solving, where an agent uses its beliefs about available actions and their consequences, in order to identify a solution over an abstract set of possible plans [86]. One can deduce that if we make a mapping from actions to Web services, consequences to execution effects, and beliefs to user requirements, then the planning problem is very similar to the problem of service composition.

Based on the aforementioned definition of planning, an AI planning problem can be described as a quintuple $\{S, s_0, G, A, \Gamma\}$ [86], where S is the set of all possible states of the world, $s_0 \subset S$ denotes the initial state of the planner, $G \subset S$ denotes the set of goal states the planning system attempts to reach, A is the set of actions that can be performed in order to reach a goal state and $\Gamma \subseteq S \times A \times S$ is a transition relation that describe the resulting state or states when a particular action is executed in a given world state. If we consider A to be the available services, G to be the goal set by the requester, and a state model related to S, s_0 and Γ and applied to the available services, then we can use existing solutions to the AI planning problem in order to solve the Web service composition problem. It should be noted that this correlation is not followed by all approaches in this category.

Logic-based Planning

A large number of AI planning techniques involves expressing a domain theory in classical logic. A set of rules is defined and plans are derived based on these rules. An early work that uses rule-based planning

is SWORD [87]. SWORD is a toolkit that enables the composition of services that are described using an Entity-Relationship (ER) model along with a Horn rule where the preconditions of the service imply the postconditions. The developer specifies the desired inputs and outputs of the composite service and SWORD attempts to generate a plan based on rules defined by data and control flow among the existing services.

Medjahed et al. [88] presents another method using rule-based planning where a language called Composite Service Specification Language (CSSL) is used to describe the desired composite service, and composability rules are then used in order to select appropriate services for the composition. The composability rules are syntactic and semantic, including qualitative rules. The authors also introduce the notion of composition templates, which are stored templates of already created and tested composite services that can be reused, as-is or as components in other composite services.

In [89], the authors extend composition rules to include new constraints known as invariants, in order to capture the knowledge that a property must not change from a state to another. These constraints are also adapted to the existing service description architecture of OWL-S. The resulting set of composition rules is used in an backward planning approach enriching it with reasoning about service compositionality. The backward planner creates the composition plan by decomposing the given set of goals and trying to find services that satisfy a sub-goal while indulging the composition rules for that particular composition. A successful plan is returned upon reaching a point where all goals are satisfied.

Apart from rule-based planning, another AI planning in the field of logic is planning with situation calculus. The situation calculus represents the world and its change as a sequence of situations, where each situation is a term that represents a state and is obtained by executing an action. McIlraith and Son [53] adapt and extend Golog, a logic programming language built on top of the situation calculus in order to allow for customizable user constraints, nondeterministic choices and other necessary modifications to better satisfy the requirements of service composition. In [90], the authors present a set of methods written in the logic programming language Prolog, for translating OWL-S service descriptions into Golog programs.

Another approach for encoding operations into first-order predicate logics is the event calculus, where events initiate periods during which certain properties hold. These periods end when another event that terminates the associated property occurs. Similarly to the situation calculus approaches, Aydin et al. [91] translate not only OWL-S service descriptions to the event calculus but also the generic composition goal, which is expressed using the compound event construct. Afterwards, using goal decomposition the generic composition is split into smaller tasks which are also captured by compound events.

A separate category of logic-based AI planning techniques encompasses all frameworks that use theorem proving, expressing the goal as a theorem and trying to prove it follows from the axioms of the application domain. [92] is one of the initial works in the field of Web services that uses theorem proving in order to allow disparate web agents to cooperate, which essentially is a preliminary form of automatic Web service composition.

In [93], the author uses Structural Synthesis of Program (SSP) to automatically compose Web services. Using propositional variable as identifiers for the inputs and outputs of services, the composition problem is transformed to the problem of proof search in propositional logic. This effort however, ignores semantic information, including only structural properties in service specifications.

In his thesis [94], Rao presents a method for automatic Web service composition via Lineal Logic theorem proving. While the external description of services is in OWL-S, Linear Logic is used internally and is able to express both functional and non-functional attributes. The composite process model is derived from the theorem proof and expressed using π-calculus [95, 96]. The theorem proving process takes into consideration all aspects of a service description, the functional properties, the semantic information as well as the non-functional attributes to generate the composition plan. Inference rules are used to distinguish between functional and non-functional attributes. This effort is also outlined in [97].

Extra-logical Planning

Despite the advantages of the aforementioned pure logic-based approaches, such as the ability to prove certain properties of domain theories and precise semantics, the AI planning community developed and uses several different formalisms to express planning domains. Planning Domain Definition Language (PDDL) [98] is widely recognized as a standardized input for state-of-the-art planners and is based on earlier languages such as STRIPS and ADL.

McDermott [99] presents a Web service composition method based on PDDL that takes into account the state change produced by the execution of the service. To describe such type of knowledge, the author introduces value of an action, a persistent knowledge type that is not treated as a truth literal. The composition plan is generated by a backward graph-based planner, called regression-match planner. In [100], WSDL descriptions along with service-annotation elements are transformed to PDDL documents. To deal with the problem of partial observability of state that is often encountered in service compositions where lack of information is common, the framework allows for replanning when additional knowledge has been acquired.

Graph-based planners such as the one used by McDermott [99], are also used in other Web service composition approaches. In [101], a mixed initiative approach is presented, as the authors want to point out that it is unrealistic to pursue fully automated approaches. A planning component is included, which is based on the Graphplan algorithm. GraphPlan [102] uses a novel planning graph to find sequences of operations that lead to the goal state. The algorithm combines reachability analysis to determine whether a state can be reached from another and disjunctive refinement, to allow for constraints to be applied in order to resolve inconsistencies. In [103], the authors extend GraphPlan to take into account input and output data types and also support loop generation based on the frequent observation of patterns. The planner produces a BPEL flow that is combined with the results of a data mediator before being executed by a BPEL engine.

Another approach in the field of graph-based planning is proposed by Lecue et al. [104]. The framework includes an Automatic Composition Engine that can take as input both natural language and formalized requests. The natural language requests are transformed into formalized ones using a built-in Semantic Analyzer. The suggested framework uses the Causal Link Matrix (CLM) formalism [105] in order to facilitate the computation of the final service composition as a semantic graph. The CLM formalism is extended to support non-functional properties of Web services. The composition algorithm advances backwards from the goals and the service request outputs and possible effects, to the service request inputs and possible preconditions. Apart from exact semantic matches, the algorithm realistically searches for partial matches, i.e. outputs being subsets or supersets of the inputs.

Hierarchical Task Network Planning

Hierarchical Task Network (HTN) planning [106, 107] provides hierarchical abstraction to deal with the complexity of real-world planning domains, such as Web services. HTN planning includes not only traditional operations, called primitive tasks that achieve certain effects when preconditions are met but also a set of methods that allow for decomposition of tasks into subtasks. An approach of using HTN planning in the realm of Web Services is based on the SHOP2 system [108] which is an ordered task decomposition planner, meaning that it constructs the plan based on the execution order of the tasks. The authors present a transformation method from OWL-S service descriptions to an HTN, derived from the conceptual similarities of an OWL-S action description and an HTN task. The framework is effective when explicit descriptions of the task to be accomplished are available, however it is unable to adapt to information change that is common in dynamic environments.

In [109], an HTN planner is combined with Partial-Order Planning (POP). POP is used to generate an initial plan which is then refined through HTN action decomposition. The final plan is then fed into a type matching algorithm in order to match HTN actions to concrete Web service operations. The final service grounded plan is executed by a workflow engine. Klusch et al. [110] present OWLS-Xplan, a framework

that combines graph-based with HTN planning and uses PDDL descriptions. An OWLS2PDDL converter performs the matching between OWL-S services and PDDL actions. PDDXML, an XML dialect of PDDL is also developed, to facilitate SOAP communication of PDDL descriptions. As far as the planning component is concerned, Xplan is a hybrid planner that combines a Relaxed Graphplan heuristic [111], with an HTN component that aims to decompose any complex task that is encountered during the graph generation. The framework also supports replanning in order to adapt to dynamically changing environments. Paik and Maruyama [112] combine HTN planning with problem solving in the form of a Constraint Satisfaction Problem (CSP) to remove limitations of HTN planners in representing user-defined constraints and additional scheduling information.

Model Checking and FSM

Model checking is a formal verification technique, which allows determining whether a property holds in a certain system formalized as a finite state model, i.e. using a Finite State Machine (FSM). Planning by model checking is capable of dealing with complex issues such as nondeterminism, partial observability and extended goals and has good semantic foundations.

Berardi et al. [113] represent services using both an external schema that specifies the external behavior of a service and an internal schema that specifies how a service uses available information to execute an action. These schemas are represented using FSMs. Based on the FSM representations of the services, the authors attempt to prove whether a composition is possible or not, solving a satisfiability problem in Deterministic Propositional Dynamic Logic (DPDL).

Pistore et al. [114] present a composition framework based on planning as model checking that covers both cases of extended goals and cases of partial observability. For each of the services participating in a composition, a planning domain and a protocol is defined and the framework aims to combine the partner protocols so that the goals are achieved. In [115], abstract BPEL processes are transformed into state transition systems and then transform these models into planning domains as a basis to realize service composition based on model checking. ASTRO [116] is an implementation of the planning techniques explored in [114] and [115].

In [117], the authors present an automated composition framework based on planning at the knowledge level. BPEL processes are translated into formally defined knowledge level models. The planner presented in [115] is then used in order to achieve a goal that is also represented in the knowledge level. Marconi et al. [118] formally specify a data flow language as well as a graphical representation in the form of data nets in order to capture composition requirements. This work is integrated in the automated composition framework presented in [115] and [117] by representing data nets as state transition systems. Finally, [119] applies the aforementioned work to a real world case study that involves the Amazon e-commerce services and an e-payment bank service.

3.3.3 Comparison

The research approaches presented in this section are summarized and compared in Table 3.3.3. We take into consideration the following criteria:

- Domain independence: The method is not exclusive to a specific domain (e.g. travelling, automotive) but can be applied to any, allowing for the solution of a broad range of problems
- Partial observability: The method is able to reason on incomplete information
- Non-determinism: The method deals with actions that may lead to different states depending on the values of some parameters (e.g. an if-else construct)
- Scalability: The method is able to solve real-world composition problems which often deal with a large number of services. A low scalability means that this particular category of approaches

becomes less efficient as the number of associated services and/or their complexity rises. On the contrary, a high scalability ensures that there is support for large numbers of services and/or high levels of complexity

- Applicability: The method uses well-known and widely used standards (mentioned in the table)

The table compares different categories of approaches and not different approaches due to the large number of approaches presented in this section. It should be noted that if an approach supports one of the criteria examined in the table, it doesn't imply that all existing approaches in the category do so.

Automatic Composition Method	Domain Independence	Partial Observability	Non-Determinism	Scalability	Applicability
Workflow	Yes	No	Yes	Average	BPEL
Logic-based pl/ng Rule-based	Yes	No	No	Varies	OWL-S
Logic-based pl/ng Situation calculus	Yes	Yes	Yes	Average	OWL-S PDDL
Logic-based pl/ng Event calculus	Yes	No	No	Low	OWL-S
Logic-based pl/ng Theorem proving	Yes	No	No	Good	OWL-S
Extra-logical pl/ng PDDL	No	Yes	No	Good	PDDL
Extra-logical pl/ng Graph-based	No	Yes	Yes	Average	BPEL
HTN pl/ng	Varies	Yes	Yes	Good	OWLS PDDL
Model-checking and FSM	Yes	Yes	Yes	Varies	BPEL OWL-S

Table 3.2: Comparison of Categories of Research Approaches

3.4 QoS-Aware Service Composition

During the last years, many services have become available, either publicly on the Web or provided by business partners that implement a piece of functionality using specific programming models and exposing it as services by using common standards and models such as WSDL, SOAP or the notation of RESTful services (via XML or JSON).

Based on the fact that such services mostly use the Internet as a backbone, Quality of Service (QoS) aspects such as performance, reliability, etc are becoming increasingly important to distinguish "good from bad behaving services". Following [120], QoS can be classified in *deterministic* and *non-deterministic* QoS attributes. Deterministic QoS attributes, on the one hand, indicate that their value is known before a service is invoked, including price or penalty rates. On the other hand, their non-deterministic counterpart includes all attributes that are uncertain at service invocation time, for example service availability [121]. Dealing with non-deterministic attributes is more complex since it requires to perform calculations based on data gathered during runtime observations.

Within the last couple of years, the research community identified a number of different ways to define composition approaches taking QoS into account and trying to optimize the overall QoS of a composition by aggregating the QoS of all services involved in a composition. Although different strategies for calculating an optimal composition according to its QoS exist, the basic methodology that is shared among most approaches is summarized below:

1. All approaches require to define an abstract composition in terms of its control flow and the activities that perform the actual work. These activities are then implemented by a number of services that are stored in a service repository (e.g., UDDI, ebXML).

2. For each activity in the composition, a QoS-aware service selection strategy has to find all services that implement the functionality of a given activity dynamically at runtime. The goal is to find the best service in terms of its QoS. This step is often referred to as *local optimization*.

3. In the presence of global QoS constraints, finding locally optima does not necessarily imply that the QoS of the overall composition is optimal. Therefore, a global optimization step is necessary to find a global optimum that fulfills global QoS constraints. This step is generally considered to be NP-hard.

Various approaches for optimizing a composition exist (step 2 and 3) and will be discussed in detail in Section 3.4.2. Besides these methodological commonalities, two very important preliminaries are often not discussed in existing papers about composition: Firstly, the QoS model and the specification mechanism that is used to associate QoS to certain services. Secondly, an approach to bootstrap and monitor QoS to ensure that valid data about the service is available to achieve a composition. Both approaches are briefly discussed in the next section.

3.4.1 Preliminaries for QoS-Aware Composition

As stated before, non-deterministic QoS properties are usually uncertain at service invocation time, therefore, monitoring techniques have to be used to boostrap and define QoS attributes for atomic services after their initial deployment. In addition to monitoring QoS attributes, a more general issue is the specification and association of QoS attributes to concrete service implementations (besides whatever concrete QoS models and metrics are used). It is not the intended scope of the paper to discuss QoS model, specification and monitoring techniques as these mechanisms are out of scope of this deliverable. However, for the sake of completeness, we briefly discuss its relevance and need for QoS-aware composition.

QoS Model

Most QoS-aware composition approaches consider performance and dependability specific attributes as well as pricing as their QoS. In Table 3.3, we have summarized a few common attributes and formulas how they can be calculated. Please note that some approaches presented in this section have a slightly different semantics for these attributes. For a more detailed description of their semantics, please see the different papers.

QoS Attribute	Formula	Category
Response Time	$\frac{1}{\#requests} \sum rt_i$	Performance
Availability	1 - $\frac{downtime}{uptime}$	Dependability
Accuracy	1 - $\frac{\#failedrequests}{\#totalrequests}$	Dependability
Price	predefined	Cost and Payment
Penalty	predefined	Cost and Payment

Table 3.3: QoS Attributes

The QoS attributes from Table 3.3 are briefly discussed below, for a detailed specification of these attributes we refer the interested reader to [121, 122].

Response time: The response time of a service S is the time needed for sending a message M from a given client to S until the response R for message M returns back to the client. The response time is provider specific (due to the network latency), therefore, it is not possible to specify a globally valid value for each client. The response time can be calculated based on a per-request basis. The formula in Table 3.3 calculates the response time for n values by summing up the individual response times and dividing it by the number of requests. This helps to get a better approximation of the average response time.

Availability: The probability that a service S is running and producing correct results. The availability is calculated by dividing the downtime by uptime (downtime ratio) and subtracting the from the maximum availability (which is 1).

Accuracy: The accuracy of a service S is defined as the success rate produced by S . It can calculated by evaluating all invocations starting from a given point in time and examining their results. It can be calculated by subtracting the failure ratio (# failed requests divided by # total requests) from the maximal accuracy (which is 1).

Price: Price is a deterministic attribute that defines the price on a per-invocation basis. Other measures could include a fixed price for a number of request, e.g., 100 EUR for 10000 requests.

Penalty: The penalty is a deterministic measure that indicates the monetary value that has to be paid by the service provider if a specific service level objective of an SLA is not fulfilled. The concrete implementation of penalties can be combined with the pricing, e.g., each penalty reduces the price for the invocation, etc.

QoS Specification and Monitoring

The preliminaries for achieving QoS-aware service composition can be divided into two bigger block, namely *Specification* and *Monitoring*. At the end of this section we will also use these preliminaries to

compare the discussed approaches.

Specification: Specification of QoS is an important aspect when dealing with QoS-aware composition. Various QoS model have been proposed in different research papers, thus it is important that they are clearly defined and metrics for specifying the QoS is important. Additionally, a mechanism that describes how to associate QoS data with a given services is important as the QoS information is needed at runtime to query the QoS of a service.

Monitoring: After specifying QoS, it is necessary to describe a mechanism how non-deterministic QoS values can be monitored. We do not discuss monitoring approaches in detail as this is out of scope for this survey. Please consider references [121, 122, 123, 124, 125] for more information on monitoring approaches and languages for monitoring of atomic and composed services.

3.4.2 Composition and Optimization Approaches

This section describes different QoS-aware composition and optimization approaches that are grouped into different categories. The categories are chosen based on their QoS-composition technique.

Combinatorial Approaches

Zeng et al. [126, 127] present a multi-dimensional QoS model that consists of price, duration, reputation, success rate and availability. Each service is associated with a quality vector containing all QoS attributes for each operation of a service. Their basic idea is to split a composition into multiple execution paths based on their notation that a composition is specified using a state chart diagram. Each execution path is considered to be a directed acyclic graph (DAG) given the assumption that the state chart has no cycles. Additionally, the authors define an execution plan which is basically a set of pairs expressing that for every task in the composition, a service exists that implements the operations required for that task. For the local optimization, the system tries to find all *candidate Web services* that implement the given task. Each service is assigned a quality vector and user defined scores for the different quality constraints. These constraints are then used to compute a score for each candidate service. Based on Multiple Criteria Decision Making (MCDM), a service is chosen which fulfills all requirements and has the highest score. In order to find a global optimum, Zeng et al. initially came a with a naive global planning approach that tries to find an execution plan for each execution paths in the statechart that describes the composition. Since each planer only covers a subset of a statechart, these ”partial” plan need to be aggregated into an overall execution path. The major disadvantage of this naive approach is the time complexity. If there are n tasks and m candidate Web services for each task, the total number of m^n, which is impractical. Therefore, the authors propose an optimization approach based on Integer Programming (IP) that has variables, an objective function and a set of constraints as an input (both latter mentioned inputs must be linear). To achieve a QoS-aware optimization of the composition, global and local constraints can be specified. In addition, an objective function has to be fulfilled. The optimization problem is then solved using an IP solver. In addition to optimizing a composition, the authors also describe an approach to re-plan and re-optimize a composition based on the fact that QoS can change over time. To do so, a composition is split into regions according to the state of the tasks (’completed’, ’to be executed’, ’to be started’, ’no longer accessible’) that allow a re-planning by adding constraints of what has already been accomplished (where state is ’completed’) to optimize services that still have to be executed.

Brandic et al. [128] describe Amadeus, a QoS-aware Grid workflow system that supports a comprehensive set of QoS requirements. It provides in addition to performance and economical aspects also legal and security aspects. The Amadeus architecture includes: (1) *Visualization and Specification component*; (2) *Planning, Negotiation and Execution component* called QoS-aware Grid Workflow Engine (QWE); and (3) a set of *Grid Resources*. The specification and visualization component comprises Teuta, a tool for UML-based Grid workflow modeling and visualization. A user may specify the workflow with Teuta by composing predefined workflow elements. For each workflow element different properties (such as

execution time, price and location affinity) may be specified that indicate the user's QoS requirements. A set of QoS-aware service-oriented components is provided for workflow planning to support automatic constraint-based service negotiation and workflow optimization. Amadeus provides adequate static and dynamic planning strategies. For the *static planning approach* a mixed-integer linear programming is applied, suitable for solution of NP-complete global optimization problems. It can be applied to the planning of the whole workflow, or to the planning of complex activities. For improving the efficiency of workflow planning a novel QoS-aware workflow reduction technique is introduced.

Yu et al. [129] discuss algorithms for Web services selection with end-to-end QoS constraints. Their approach is based on different composition patterns similar to [130] and they group their algorithms according to flows that have a sequential structure and ones that solve the composition problem for general flows (i.e., flow that can have splits, loops etc). Based on this distinction, two models are devised to solve the service selection problem: a combinatorial model that defines the problem as multidimensional multi-choice knapsack problem (MMKP) and the graph model that defines the problem as a multi-constrained optimal path (MCOP) problem. These models allows the specification of user-defined utility functions that allow to optimize some application-specific parameters and the specification of multiple QoS criteria taking global QoS into account. In the case of the combinatorial model, the authors use an MMKP algorithm, that is known to be NP-complete. Therefore the authors apply different heuristics to solve it in polynomial time. For the general flow structure the authors use an IP programming solution (also NP complete), thus they again apply different heuristics to reduce the time complexity.

Jaeger et al. [130, 131] present a different approach of deriving QoS of a composite services by following an aggregation approach that is based on the well-known workflow patterns by Wil van der Aalst et al. [132]. The authors analyze all workflow patterns for their suitability and applicability to composition and then derive a set of seven abstractions that are well-suited for compositions, called *composition patterns*. Additionally, the authors define a simple QoS model consisting of execution time, cost, encryption, throughput, and uptime probability and QoS aggregation formulas for each pattern. The computation of the overall QoS of a composition is then realized by performing a stepwise graph transformation, that identifies a pattern in a graph, calculates its QoS according to the aggregation functions and the replaces the calculated pattern with a single node in the graph. The process is repeated until the graph is completely processes and only one single node remains (which is itself a sequence according to their composition pattern). For optimizing a composition, the authors analyze two class of algorithms, namely the 0/1-Knapsack problem and the Resource Constrained Project Scheduling Problem (RCSP). For both algorithms a number of heuristics are defined to solve the problems more efficiently.

Evolutionary Approaches

Canfora et al. [133] propose an approach that tries to solve the QoS-aware composition problem by applying genetic algorithms. Firstly, the authors describe an approach to compute the QoS of an aggregated service, similar to [134, 130, 135]. Secondly, the main issue of their approach, namely the encoding of QoS-aware service composition problem as genome, is presented. The genome is represented by an integer array with the number items equals to the number of distinct abstract services composing our service. Each item, in turn, contains an index to the array of the concrete services matching that abstract service. The crossover operator is a standard two-point crossover, while the mutation operator randomly selects an abstract service (position in the genome) and randomly replaces the corresponding concrete service with another one from the pool of available concrete service. The selection problem is modeled as a dynamic fitness function with the goal to maximize some QoS attributes (e.g., response time) while minimizing others (e.g., costs). Additionally, the fitness function must penalize individuals that do not meet the QoS constraints. The authors evaluate their approach by comparing it to well-known integer programming techniques. In another paper, the authors also describe an approach that allows re-planning of existing service compositions based on slicing [136].

Stochastic Approaches

Mukhija et al. [137] present the Dino framework that is targeted for the use in open dynamic environments. Their main idea is that no global view on a service composition is available and thus each service specifies which services it requires for its own execution. The service composition is formed at runtime by the infrastructure. A key aspect is the fact that service requirements can change dynamically (triggered for example by changing application context). Dino also supports QoS-aware service composition (similar attributes as depicted in Table 3.3) composition by describing it formally using an ontology. QoS is computed by using the actual and estimated QoS values that are monitored by the Dino broker. QoS computation is modeled by using a continuous-time Markov chain that enables the association of a probability value that expresses the confidence of the QoS specification.

Wiesemann et al. [138] present a QoS-aware composition model that is based on stochastic programming techniques. Their model takes QoS-related uncertainty into account that minimized the average-value-at-risk (AVaR) – a particular quantile-based risk measure – from the stochastic programming domain. The trade-off between different QoS criteria is modeled by a combination of goal weighting and satisficing two prominent techniques in multi-criteria optimization with the goal to minimize the AVaR.

3.4.3 Comparison of Approaches

In this section, Table 3.4 summarizes existing approaches according to the following key aspects of QoS-aware composition.

Specification: Expresses if the authors use a specific QoS model and how the QoS is specified an associated with the services, e.g., using UDDI or WS-Policy.

Monitoring: Specifies whether QoS attributes are monitored or just statically defined or not possible to determine from the paper.

Composition: As the composition aspect if the main focus of the paper it is further broken down to the following attributes:

- Composition Constructs: Specifies the constructs that are supported for creating a composition. We mainly distinguish between two aspect: simple processes (sequential ones with no loops and/or splits) or more complex structures of processes (e.g., loops etc).
- Constraint Types: This aspect specifies constraints that are supported; we distinguish between local and global constraints.
- Optimization Technique: It specifies the technique that is used for optimizing the QoS of a composition, e.g., Integer Programming, etc.
- Complexity: It states the time complexity of the approach.
- Heuristics: Specifies if heuristics are used to speed up the time for solving the QoS-aware optimization.

Re-optimization: Expresses whether re-optimization is performed.

Paper	Specification	Monitoring	Composition					Reoptimization
			Composition Constructs	Constraint Types	Optimization Technique	Complexity	Heuristics	
Zeng et al. (2004)	basic QoS model/ QoS specification in UDDI	n/a	complex structures	local & global	IP	NP-complete	no	Region-based
Brandic et al. (2008)	basic QoS model/WSLA	n/a	sequential processes (with splits)	local & global	workflow reduction & IP	NP-complete	no	n/a
Yu et al. (2007)	basic QoS model/QoS specification n/a	n/a	simple and complex structures	local and global	MMKP, MCSP-K, IP	NP-complete	WS_HEU, WFlow	n/a
Jaeger et al (2005)	basic QoS model/QoS specification n/a	n/a	complex structures	local and global	0/1 Knapsack & RCSP	NP-complete	Greedy, Discarding Subsets, Bottom-Up Approximation, Pattern-wise Selection	n/a
Canfora et al. (2005)	basic QoS model/QoS specification n/a	n/a	complex structures	local & global	Genetic Algorithm	n/a	no	Slicing
Mukhija et al. (2007)	QoS ontology/Registry	broker-based	simple structures	local	not specified	unknown	unknown	n/a
Wiesemann et al. (2007)	basic QoS model/QoS specification n/a	n/a	complex structures	probabilistic local and global constraints	Stochastic Programming	tbd	no	n/a

Table 3.4: Comparison of Approaches

Chapter 4

Verification of Service Compositions

The main aim of a verification procedure is to check that the constructed system meets some properties. These are sometimes generic, as termination, absence of deadlocks or starvation, temporal ordering of actions (e.g., making sure X is always done before Y), and sometimes these are related to an intended behavior which has been stated in a previous design stage. Some key points in the *formal verification* field are:

- The language with which the system is represented (for verification tools are seldom directly applicable to the wide variety of different languages systems are written in / designed with). Examples of system modeling language are Promela [139] (for the SPIN model checking system) or the NuMSV language [140] (for the NuMSV model checker). In this respect, Petri Nets and other formalisms which represent a system, are modeling languages.[1]

- The language with which the properties to be checked are expressed. Only properties that can be expressed within that language (and in its associated semantics) will, naturally, be amenable to be checked. The type of checkable properties, in turn, have an influence on the internal algorithms used by the verification tool. In all cases, the verification procedure involves *negating* the property to be checked, adding it to the system representation, and seeking for a *contradiction*: a reachable state where the negated property is not met. Absence of such a state means that the original property holds.

  When a property violation is detected, tools can usually give a trace of the actions which lead to the faulty state from the initial stated. This trace is expressed in terms of the model of the system, and the distance between this model and the modeled system is critical to be able to precisely map fault witnesses back to the original system.

- The actual procedure to verify that the system meets the properties. Model checking procedures are among the most widely used procedures: the system to be checked is a *Kripke Structure* on which temporal properties are tested for satisfiability. *Automata-theoretic approaches* [141] explicitly expand the system states, while *symbolic model checkers* (mainly, BDD-based and, more recently, SAT-based, represent the possible system states with a bounded propositional formula.

  Constraint-based verification procedures use constraints (equations and disequations, in this case usually over integers) both to represent system evolution and properties to be checked. Arithmetic methods (e.g., variants of the Fourier-Motzkin variable elimination procedure) are used to check for satisfiability, with very good results in many cases, with the added benefit that infinite-state system can in some cases be dealt with directly.

[1] It should be noted, however, that some of them have property verifiers of their own. For example, there exists a wide variety of techniques to verify Petri Nets which do not rely (at least directly) on Model Checking.

Some verification algorithms make use of abstraction procedures to be able to cope with systems with an infinite (or very large) number of states, and, orthogonally, to start with an abstract representation of the system and perform a stepwise concretization process in order to discard false negatives when a contradiction is detected.

Verification procedures based on proof-theoretic methods, which tried to symbolically prove / disprove a (usually temporal) property in a system were tested in the earlier days of verification, but did not become popular due to, e.g., scalability problems.

Due to the need to give correct answers[2] regarding which properties hold in a system, formal verification needs a formal base on which to build non-ambiguous reasoning. I.e., a usual prerequisite is to have a clearly defined semantics for the operations of the system being checked and for the way they are composed together. This is the first and necessary step for any verification attempt based on formal methods. If this formal semantic base does *not* exist beforehand, then the translation of a system into a format amenable to be formally verified provides in itself a semantics for, at least, a part of the language the system was written in.

Verification can in principle be applied at any stage where the conditions above are met, for example:

Design time: a design can be verified against some properties, being them *general* or pertaining to the application at hand.

Pre-deployment: after the system is built, it can be checked to make sure that relevant general properties hold, or that some design properties have not been lost (due, e.g., to mistakes in the development process). Sometimes, for efficiency / conciseness / finiteness only a part[3] of the system variables are taken into account.

Run-time: (also known as *run-time verification*) verification of properties takes place, additionally, while the system is executing, either actually instrumenting the system itself or by looking at system traces / logs. This approach can usually take into account all the variables in the system, but in turn it cannot guarantee that the system will operate as expected with untested inputs. This is obviously highly related to monitoring.

Verification approaches, unlike *correctness by construction* (CbC), work on an already built system or software, while CbC tries to construct / design software / systems which are correct from the ground up by defining the properties it has to finally met and making sure that these properties are not violated in any step, including composition of subsystems. These approaches should be seen as complementary rather than as competitors, as the properties they tackle are usually different and the moment in development in which they are applied is also different – not to mention the technology and tools themselves.

The difference between validation and verification is easy to grasp from the following lemmas:

Verification: *Did we built the system correctly?*

Validation: *Did we build the right system?*

Validation typically involves the final user / client and is *context dependent:* the environment in which the system is to be installed / executed has to be taken into account, as well as HCI issues. It is, therefore, difficult to formalize fully, and it partly is a subjective process.

In the software and services field, verification proposals have used predominantly automaton-based model checking with temporal logic properties. As for the level of the systems being modeled, the majority of the pieces of work examined geared towards modeling service compositions expressed using BPEL, sometimes extracting the model automatically from BPEL code. To the best of our knowledge,

[2] Or, at least, safe approximations thereof.

[3] Or an abstraction.

even in the cases where it is claimed that choreographies are verified, a initial step to convert them into orchestrations, which are then subject to verification, is taken. Of course this does not preclude that verification procedures can tackle choreograpies as the subject of study, in order to check, e.g., timing constraints or refinements of the choreography.

4.1 Verification Based on Model Checking

Temporal logic properties (expressed in a language like LTL or CTL, or with another language version which is later translated into them, like XPath), are the ones which are usually attached to model checking procedures. *State reachability* (e.g., *state S_a will never be reached if state S_b has been already reached*) is, in some sense, the basic procedure behind the scenes. This includes testing for *invariants*, which must be true in every state, such as *for every state, if the activity named* `init` *is running, the part global variable* `order_sent` *must have a false value* and *Goals* (properties on final states).

Also, some model checkers have specific constructs to express pre- and post-conditions on transitions. Given a transition between states, a pre-condition (post-condition) is an assertion that must be true always immediately before (after) the transition is taken. E.g., immediately before the activity named `init` completes its execution, the part `urgent` of the global variable `order_sent` must have a true value.

A number of approaches start with service compositions which are expressed using BPEL processes, while others use, as system to be verified compositions which are written at some other level. We will start with the former, and Section 4.1.2 will briefly deal with the former.

4.1.1 Approaches Starting with BPEL Processes

In this case the properties to be checked are either generic safety properties (such as termination, absence of deadlocks, etc.) or properties which have been explicitly stated by the designers. In this case there is, formally, no check that the BPEL composition preserves characteristics existing in some previous, maybe higher-level, design, since such alternative system formulation does not exist.

In general, the path followed is to compile the BPEL design into some formalism which can be input by some automatic verification tool together with a representation of the properties to check, and let the tool do the checking; particular approaches add different specific characteristics, like for example, checking not only behaviors, but also the compatibility of the interfaces of the different web services.

The need to check several types of characteristics (e.g., not only behavior) is acknowledged, for example, by [142, 143], which states that composition of services has to be verified at three levels: interface, behavior, and data. They, however, focus on the first two, and they pay attention to behavioral (e.g., temporal) properties, specifically safety and liveness ones. In this approach orchestrations are gathered and analyzed, and the activities are modeled using BPEL. At this point, the BPEL design is translated into a Finite State Process (FSP) model which is compiled into a Labelled Transition System (LTS) to be checked with the LTSA (LTS Analyzer) tool. The properties to be checked come with the reachability analysis in the LTSA, but more properties can be added by the user by directly coding it in FSP.

The complexity of BPEL has often been an issue. However, the work in [144] claims to take into account all the BPEL constructs, except for those related with time. It is based on bounded model checking, using the *Bogor* [145] extensible tool. As Bogor accepts formulæ written in LTL (through a plugin, part of its extensible framework), all properties expressible in LTL can be subject to verification. Additionally, expressions written in Ws-Col, an assertion language designed to state properties to monitor, can be checked by Bogor. This piece of work provides examples with translations of some basic BPEL constructions into BIR (the modelling language for Bogor) and offers a comparison of the number of states needed to check properties on some benchmarks in relation with SPIN, which is very favorable to Bogor.

Compiling BPEL into a modelling language for a model checker is also the approach of [146] where each web service is modeled using a finite state machine, and the different services are communicated using activities which synchronize the states of the different partners. The correctness properties to be verified are written using XPath predicates directly in the BPEL description of the business process. The BPEL process is translated into the language of the NuSMV [140] model checker, while XPath expressions are translated into CTL.

Timing aspects are one of the points to deal with when checking properties of systems in general, and in particular of service compositions — especially when Quality of Service is to be taken into account. This does not necessarily have to do with micro-scale time constraints, such as those frequently found in embedded systems: e-Government procedures do have deadlines and requirements on when some actions are taken, and the involved time span usually makes the computation time associated to the procedure completely negligible. This is the view taken in [147], where an automaton-based formalism, called *web services timed transition systems* (WSTTS), related to timed automata, is used to model the time behavior of BPEL process composition. Quantified Discrete-time Duration Calculus [148] was adapted to perform computations on WSTTS models. The processing time corresponding to BPEL itself is not taken into account because of the reasons aforementioned. Among the properties under check, time constraints can be tested against the modeled process, by translating time constraints into another WSTTS model and generating the cross-product with the model corresponding to the BPEL processes. This transition system is then translated into NuMSV which performs the actual model checking. Additionally the paper provides an algorithm to analyze compositions of services and extract the maximum and minimum time in which the composed service can finish.

Even if the model of composition under consideration is finite (because of the finite number of processes and states in each process), the model of the system itself may be infinite due to the asynchronous message passing adopted by BPEL. This means that an unbound number of messages may be in transit from one process to another process. These messages are of course part of the system, and they have to be somehow dealt with in order to faithfully represent the system behavior. This has been tackled by [149] and [150] using two different approaches.

The former translates BPEL processes into an intermediate language (guarded automata [151]) and messages between processes are represented as XPath expressions. This, additionally, makes it possible to prove properties about the data, too, thus having a fine-grained set of properties. The guards of the automata are also XPath expressions with local XML variables. The automaton is compiled into Promela (the SPIN [139] model checker language), handling local XML variables (i.e., keeping more data semantics), which is then fed to SPIN. Asynchronous communication, which needs unbound queues, can in some cases be replaced with synchronous communication one while preserving the conversation set, which makes it possible to have a bound number of states.

The approach in [150] tackles the problem of asynchronicity by substituting it with the possibility of *reordering* messages with several degrees on the allowed reorderings: no reordering, reordering between messages delivered to different partners, and reordering in messages delivered to the same partner (i.e., out-of-order delivery). Different validity checks are defined for each of these scenarios. This approach transforms BPEL designs into an state transition system (STS) extended with the notion of *trace compatibility*. The STS model is then subject to parallel composition to account for different processes and a NuMSV description is generated to be verified with respect to the validity defined for the communication model chosen.

Of particular interest to be modeled are the parts of the language which deal with abnormal situations, such as exceptions. As the existence of a fault must in practice not mean that the rest of the system has to collapse, some effort as been put in making sure that verification also takes into account the possibility that exceptions and exception handling can happen. The work in [152] claims precisely that, additionally providing facilities to check pre- and post-conditions of transitions.

An interesting side-effect of the analysis of reachability that most tools perform is the possibility to detect execution paths / activities which will not be needed in practice. This is one of the features of [153].

This piece of work extracts behavioral properties from BPEL code by constructing an extended finite automaton (EFA – essentially same as a guarded automaton in [149, 151]). Within it, temporal properties are described and the automaton is compiled into Promela and checked by SPIN. Attention is payed to *dead path elimination*, needed in order correctly deal with faulty situations where activities which can be logically executed do not need to be executed under a (simple) operational semantics.

Many approaches use the well-known model checkers SPIN a NuSMV; however others choose, for a variety of reasons, other model checkers, such as Bogor (seen before). The *Concurrency Workbench* (CWB) is another model checker which was used in [154] to verify orchestrations. CWB is a powerful and extensible tool, designed to prove properties on Milner's CCS, which can be extended using the *Process Algebra Compiler* (PAC). In this paper, BPEL is compiled into a *BPEL Calculus*, for which a module was written using PAC. CWB can then perform model checking, preorder checking,[4] and equivalence checking (bisimilarity and observational equivalence, in this case) to, for example, detect states which are operationally identical and which may lead to a simplification in the system.

Petri Nets are a very well studied formalism [155, 156] which has been applied to the modelling and verification of service compositions. In particular, in [157], Petri Nets have been used to give a semantics to BPEL, but also, based on that semantics, to model BPEL processes and verify properties on them, and also to pave the way to making safe and semantics-preserving transformation of BPEL processes. The authors show how BPEL constructs can be mapped into Petri Nets and, using the WofBPEL tool, the authors can detect unreachable activities, conflicting message activities, or realize that some message will not be received, in order to discard it as soon as this is detected.

Petri Nets are also used by [158] to express BPEL service models and to check their adherence to an abstract correctness criterion called *usability*, which may have several flavors (it can be local, distributed, etc.) and which is related to the termination of the processes while meeting some characteristics, such as absence of deadlock.

4.1.2 Approaches Dealing with Other Design Formalisms

Verification is amenable to be used at several stages in the development process. In principle, high-level designs with enough information can be verified to check that some properties are satisfied, and, at the same time, the (desirable) properties / requirements expressed in these designs can be checked for in the BPEL code and also at lower level.

In [159], the process model of OWL-S (the Web Ontology Language for Web Services) is used to represent the behavior of the web service. Since OWL-S gives some description of the services being composed, these services are no longer black boxes and the model checker has additional information on certain boundedness properties of the aggregation. This model is then automatically translated into a C-like language which is the input for the model checker BLAST, enhanced to check concurrency in OWL-S. Improvements aiming at not making the interleavings too costly were adopted. When the verification stage passes, positive test cases for the situations where the checked formulas hold can be generated; the same is done for the cases where they do not hold, generating *negative* test cases. These may be used to test and validate the final implementation.

Some approaches use ontology-based technology to express what a service composition should do. For example, [160] use the ontology Daml-S Daml+Oil to capture service semantics which is then translated to first order logic and to Petri Nets. In this point, composition of services can be performed, as well as verification thereof, performance analysis, and a series of other relevant operations, many of them casted in the terms of reachability and deadlock analysis.

An interesting approach which directly deals with the dynamic nature of web services is that of [161], where it is acknowledged that the changing environment of where web services are executed may invalidate the assumptions statically made at design / verification time. In this case, design-time verification is made through model checking where external entities are abstracted using their interfaces and whatever

[4] Verifying that the model of a system and a specification thereof (in μ-calculus, in this case) have the same behavior.

Approach	Modeled system	Modeling language / Checker	Types of properties tackled / Language to express them	Specific characteristics
Foster et al. [142, 143]	MSC / BPEL	FSP / LTSA	Safety (e.g., no deadlock) and liveness (e.g., progress)	User can define new properties in FSP
Bianculli et al. [144]	BPEL (stand-alone process or composition)	BIR / BOGOR	LTL / Ws-Col	Covers all BPEL constructs, except those related with time
Mongiello et al. [146]	BPEL (enriched with XPath expressions to state properties to be checked)	NuSMV	CTL	
Kazhamiakin et al. [147]	BPEL	WSTTS / NuSMV	Time-related constraints	Includes explicit algorithms to check for max./min. time
Fu et al. [149]	BPEL	Promela / SPIN	LTL	Properties on data also checked; XPath models message between processes; asynchronous communication sometimes replaced by synchronous comm.
Kazhamiakin et al. [150]	BPEL	extended STS / NuSMV	Validity defined according to asynchronicity model	Message asynchronicity dealt with by message reordering; different levels of reordering

Table 4.1: Model Checking on BPEL Approaches

Approach	Modeled system	Modeling language / Checker	Types of properties tackled / Language to express them	Specific characteristics
Arias et al. [152]	BPEL	Promela / SPIN	Invariants / predicates true at a final state / pre- and post-conditions of transitions / reachability of some states; faults and exceptions taken into account; VERBUS as framework	Multiple back-ends
Nakajima et al. [153]	BPEL	Promela / SPIN (through an EFA)	LTL	Also performs *dead path elimination*
Koshkina et al. [154]	BPEL	BPEL calculus / PAC / CWB	temporal properties, preorders, and behavioral equivalence	Takes advantage of CWB's extensibility
Ouyang et al. [157]	BPEL	Petri Nets / WofBPEL	Many, including unreachable activities, conflicting message consumers, detection of message to be garbage collected	Provides explicit Petri Net semantics for BPEL
Schlingloff et al. [158]	BPEL	Petri Nets / LoLA	Defines an abstract property, *usability*	Shows equivalence with model checking in alternating logic

Table 4.2: Model Checking on BPEL Approaches

properties can be derived out of them. Assertions are given to *represent* functional and non-functional properties. BPEL processes are mapped in the input language of the Bogor model checker. Some of the assertions are assumed to be true by the model checker (e.g., they represent knowledge about the external environment), and some are intended to be actually checked. Both types of assertions can be checked at by specifying monitoring rules and parameters. These state which has to be monitored, the ranges within which values are considered valid, and the frequency at which monitoring has to be made.

Verifying against specifications has already been tackled when the specification is done using common tools closer to Software Engineering. The work in [162] starts with a specification of the composite model done in UML using Message Sequence Charts (MCS). When the BPEL processes are ready, both the MSC and the BPEL processes are compiled into separate FSPs. By verifying trace equivalence of these two FSPs, using the LTSA tool [143], the compliance of the BPEL design with the MSC specification is ensured.

Another way to specify what a system should do is clarify how it should react to some situations written up by the system designer or system architect. One way to facilitate this is to design a specific high-level language to express, e.g., business requirements which is later on refined into a more language more closer to the business process. This is what the work in [163] does: the graphical modelling language *Tropos* language, together with its formal counterpart, *Formal Tropos*, makes it possible to express the requirements a business process has to meet, and to perform model checking (using, in this case, NuSMV) directly on the business requirement level. Formal Tropos specifications can be refined into BPEL process specifications, on which another model check procedure can be performed. it is worth to be noted that this proposal, in which correctness of the system under development was ensured by checking that the refinements still preserved the properties of the initial design, is not very far, in spirit, from what a *Correctness by Construction* approach would do.

The key idea of this [164] is to verify properties of interactions between peers with the help of a design pattern, the *PeerControllerPattern*. In this pattern, every participant provides a given interface to interact with other partners and a behavioral description thereof (similar in spirit to a contract in O.O. programming), from which BPEL specifications are generated for interoperability. The interface specifications are checked using Java's PathFinder, and the behavioral part is checked with SPIN, assuming bounded-length message queues. The authors also pave the way to check behavior assuming unbound queues by pointing out that the synchronizability analysis of [149] can be adapted for their case.

4.2 Other Verification Techniques

Computational logic has been used to verify service compositions in [166]. In their framework, $\mathcal{S}$CIFF, the composition of services is captured by describing the *events* which must happen, are expected to happen, have happened, etc. in the form of logical formulæ: choreography integrity constraints, as rules with the shape $Head \rightarrow Body$ express implications from expectations to events that must happen. They can be annotated with timing constraints. Additional knowledge on the choreography (such as number of participants) is also added to the description of the choreography, but it does not change its expected behavior. The specification is animated (e.g., run), and any errors (i.e., situations not abiding by the integrity rules) found are flagged.

Event Calculus has also been used to check consistency of business protocols [167]. BPEL processes are translated into event calculus, and the resulting *program* can be statically checked for potential flaws, or run-time verification can be performed (by, e.g., feeding the event calculus model with a trace of a real execution and checking that this trace really meets the properties expressed in the event calculus model).

Abstraction is pervasive in the analysis of software systems; for obvious reasons, linked to the spirit of verification, it is not so widely used when checking for program correctness. This is however not a reason not to use abstraction of programs or domains (and it is needed in some cases — for example, in the case of verifying infinite-state systems or even systems with an extremely large number of states). In [168], the authors deal with the very concrete case of verifying web services written in PHP5 through its translation

Approach	**Modeled system**	**Modeling language / Checker**	**Types of properties tackled / Language to express them**	**Specific characteristics**
Huang et al. [165]	OWL-S	C-Like lang. / enhanced BLAST	Reachability; undesired states / properties marked with the keyword ERROR for the checker to raise an error if the state is reached	Test case generation is facilitated
Narayanan et al. [160]	DAML-S + DAML-OIL	Petri Nets / KarmaSIM	Reachability and deadlocks	Gives semantics to DAML-S through Situation Calculus and Petri Nets
Baresi et al. [161]	BPEL + ALBERT assertions	BIR / BOGOR	Functional and non-functional (in the ALBERT language)	Monitoring rules are generated
Foster et al. [162]	UML / BPEL	FSP/LTSA	Verification against a MSC specification	It checks trace equivalence between the UML/MSC design and the BPEL design
Kazhamiakin et al. [163]	BPEL / *Formal Tropos*	NuSMV	Requirements stated by the designer	Checking along the refinement process; not very far away from CbC
Betin-Can et al. [164]	Java interface classes / BPEL	PathFinder (for interfaces) / SPIN (for behavior)	Interface conformance and behavioral properties	Heavily based on a design pattern (*PeerControllerPattern*)

Table 4.3: Model Checking Using a Previous Design

Approach	Modeled system	Modeling language / Checker	Types of properties tackled / Language to express them	Specific characteristics
Alberti et al. [166]	Choreography expressed in $\mathcal{S}$**CIFF** as CLP rules	CLP rules	Ad-Hoc Java tool	Verification linked to specification animation, using an *event source*
Rouached et al. [167]	BPEL	Event Calculus / Discrete Event Calculus Reasoner	Chiefly deviations of execution logs w.r.t. expected behavior	Focus on dynamic checking
Sharygina et al. [168]	PHP5	SAT model checker		Compositions taken care of with parallel process operator

Table 4.4: Approaches Not Directly Using Model Checking

into a labeled Kripke structure which CSP semantics for synchronization. In their approach, they initially abstract data at a coarse level. If a reported error trace is not realistic (because of the abstraction of the data), this is used to *refine* the abstraction until no more spurious errors are found. They use SATabs to model check the abstracted program and to refine it, using the unsatisfiable core of the SAT instance.

Chapter 5

Conclusions

5.1 Summary

This deliverable has presented the state of the art in the composition and coordination of services.

In the first chapter, we have identified four service composition types: service orchestration, service choreography, service wiring, and service coordination. For service orchestration, BPEL has been adopted as a standard language and is supported by most big middleware vendors today. In the service choreography space, the most prominent languages are BPEL4Chor, WS-CDL, and Let's Dance. They can be classified across two dimensions, implementation-independent vs. implementation-specific, and interaction model vs. interconnected interface model. For service choreographies there is not yet a standard language; BPEL4Chor is the most recent approach and because of the popularity of BPEL might have good chances in this respect. Service wiring is a composition model which allows creating service assemblies by wiring together requested and provided interfaces of services. The Service Component Architecture is a technology which supports service wiring and which is supported by several Enterprise Service Bus implementations. Finally, we have looked at service coordination, a model, in which services agree on the outcome of a business protocol or transaction based on a coordination protocol. So far this model is used mostly for implementing transaction protocols, however, also business protocols such as auctions could be supported. Finally, we have looked at semantic WS composition languages. These languages use ontologies and semantic Web services in order to reach a higher degree of automation in service discovery and service composition. The most promising SWS composition languages are BPEL4SWS and OWL-S. The first one is a mixture of BPEL, WSMO and SAWSDL while the second one is a pure semantic language. BPEL4SWS seems more promising as it combines the best of the composition and SWS discovery worlds but there is no actual implementation supporting it, yet, from planning until dynamic binding and execution. In addition, by mixing different languages, there is always the risk of non-uniformity and user confusion. On the other hand, OWL-S is a mature uniform language that is supported by concrete implementations from planning until execution. However, it presents some drawbacks that prevent it from becoming a de facto standard and restrict its use.

In the next chapter, we have described service composition approaches, which deal with creation of service composition models. Model-Driven Service Composition copes with generating service orchestration models from more abstract models. These abstract models are often created using platform-independent modeling notations such as UML Activity Diagrams or BPMN and represent abstract business process models modeled by business analysts. The approaches deal then with the transformation of abstract models to executable composition models, such as executable BPEL processes. That transformation is in general a process which involves both manual and automated tasks. Model-Driven service composition is relatively popular and implemented in several commercial products. The goal of automated composition is to select services and create a service composition based on an abstract goal without human intervention. In this context, we have presented and compared approaches based on workflow techniques and AI planning. Finally, we have dealt with QoS-based service composition which attempts to create a service

composition that adheres to local and global QoS constraints.

The last chapter has presented verification techniques for service composition. Most techniques are based on model checking. many of them focus on BPEL processes, checking properties such as safety and liveness.

5.2 Research Challenges

In the following we identify research challenges in respect to service composition approaches.

Model-Driven Service Composition

During the last years, significant research results have been achieved in the area of model-driven service composition: different high-level languages, and their mapping to executable representations (mostly WS-BPEL) have been proposed, and production-level tools are available on the market. However, some research challenges remain to be tackled in the future. First of all, integration of semantics or QoS into business models is not a fully solved issue today, even though some initial work has already been carried out. New high-level languages or models need to be defined, that can represent all metadata that will be important on a technical workflow level already in the business model. Additionally, current tools lack a full roundtrip modeling support: optimally, transformation should not only be possible top- down (i.e., from business models to executable processes), but also bottom-up (i.e., from executable processes to business models). So far, transformations in this direction has been researched less extensively than the top-down direction. Finally, the transformation between different types of models (i.e., from graph-based models to flow-based models, and vice versa) remains an open issue for future work.

Automated Service Composition

Automated service composition is still an open research problem since none of the approaches presented has been universally accepted as a standard method for automatically composing Web services. First and foremost, steps should be taken towards a standard way of describing user goals and user requirements. This description should be able to capture functional requirements as well as non-functional properties, including QoS aspects. At the same time, it should be easy for a user to express his requirements using such descriptions. Finally, these descriptions must be domain-independent, in order to cover all cases where service composition may be needed. Moreover, the problem of automatically matching the aforementioned requirements to compositions of existing Web services must be solved. If we settle on the AI planning solution instead of the workflow techniques, it remains an open issue to decide which of the available planners is most suited to the problem of Web service composition. It should also be examined whether some kind of transformation of the user requirements is necessary in order to meet the needs of a chosen planner or if a new customized planner should be constructed in order to meet the demands of service composition. Finally, it remains to be decided whether the highest possible level of automation should be our goal or if it is more effective to examine which level of automation yields the best results while at the same time being able to address real-world service composition problems. Allowing for some aspects of service composition to be conducted manually might result in a more realistic and satisfying solution.

QoS-aware Service Composition

QoS-aware service composition is an important area within the scope of available service composition methodologies as QoS allows to distinguish good from bad behaving services. This aspect is increasingly important when dealing with service-centric business applications. Current research in this area mainly focuses on algorithms and methods for optimization of QoS-aware compositions. Future research challenges will have to focus on models and languages for describing QoS-aware compositions. The

main goal is to ease the complex process of engineering service compositions by raising the level of abstraction. The focus should be laid on semi- automated techniques that allow users to specify what a composition should do - in a declarative manner - and what QoS guarantees are required and should be delivered at runtime. Such declarative specifications will consist of a number of constraints (functional and non-functional) that need to be matched and resolved against available services and their QoS properties by leveraging strong support from service runtimes (e.g., querying capabilities, metadata support). Facilitating such a declarative specification could be achieved using domain-specific languages (DSLs) that capture these specifications and will be integrated with service runtimes to iteratively engineer and refine a composition to the users requirements.

Verification of Service Compositions

BPEL is by far the most popular target for verification, yet it still poses challenges which have not been completely addressed. Among those, some characteristics of the language such as compensation and fault mechanisms, have been reported as being involved to model. Issues such as dynamic process instantiation are also to be properly addressed as an integral part of the model of execution of service-based applications. Indeed, in order to perform any formal work in a language, its formal semantics, in a formalism which is suitable for the technique to be applied, is needed. Every verification proposal tackling BPEL implicitly defines a semantics through the translation into a verification language,[1] but there is not a widely acknowledged formal semantics for BPEL as far as we know.

Being able to easily map a fault discovered by the verification process back to the initial specification of the system is a very interesting feature. It depends on the particular formalism which is used to verify the service composition, and not all proposals so far can do it to the same extent; it is indeed a direction for future research and development. Related to this, being as accurate as possible in the detection of faults is a needed characteristic. Some parts of the service composition are abstracted / overlooked when performing verification — e.g., some characteristics of the language being not taken into account has already been mentioned, and also concrete data values are sometimes not dealt with. Even when these are abstracted in a safe way (i.e., no faulty composition is determined to be right), reducing the number of false negatives is a very deasirable feature. Standard approaches include stepwise concretization (as in [168]), but it is still to be applied to service compositions proper, which are unaware of how the aggregated services are implemented. Introducing data aspects in the model to be verified may be a way to tackle this problem.

Verification time and memory usage is a general issue, and it can bite sufficiently complex service compositions. Among the possible workarounds, performing a pre-verification analysis which can gather information which discards some execution paths is a possible line of work. Indeed, analysis information should in general be useful to finally map the specification of the composition into an executable language, and not only to perform a swifter verification.

Finally, a verified model can be applied to several purposes besides generating (hopefully) correct executable code. It has to be noted that, although the model itself can be correct, it is possible that the translation to the executable code is faulty (due to e.g. an error in the software which performs the translation) or that the environment in which it is finally executed does not conform with the initial assumptions. However, a verified model can be used to generate *verified tests* which can be applied to the finally deployed service composition. If some of these tests fail, there is definitely a disagreement between the model and the deployed composition. Otherwise, while it cannot be ensured that the software is correct, there is additional evidence to trust it.

[1] Although some pieces of work [157] explicitly propose a semantics.

Bibliography

[1] M. N. Huhns and M. P. Singh, "Service-Oriented Computing: Key Concepts and Principles," *IEEE Internet Computing*, vol. 9, no. 1, pp. 75–81, 2005.

[2] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: State of the Art and Research Challenges," *Computer*, vol. 40, no. 11, pp. 38–45, 2007.

[3] T. Erl, "Service-Oriented Architecture. Concepts, Technology, and Design," *Prentice Hall*, 2005.

[4] M. P. Papazoglou and W.-J. Heuvel, "Service Oriented Architectures: Approaches, Technologies and Research Issues," *The VLDB Journal*, vol. 16, no. 3, pp. 389–415, July 2007.

[5] A. Tsalgatidou and T. Pilioura, "An Overview of Standards and Related Technology in Web Services," *Distrib. Parallel Databases*, vol. 12, no. 2-3, pp. 135–162, 2002.

[6] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI," *IEEE Internet Computing*, vol. 6, no. 2, pp. 86–93, 2002.

[7] R. Khalaf and F. Leymann, "On Web Services Aggregation," in *TES*, ser. Lecture Notes in Computer Science, B. Benatallah and M.-C. Shan, Eds., vol. 2819. Springer, 2003, pp. 1–13.

[8] S. Dustdar and W. Schreiner, "A Survey on Web Services Composition," *IJWGS*, vol. 1, no. 1, pp. 1–30, 2005.

[9] C. Peltz, "Web Services Orchestration and Choreography," *Computer*, vol. 36, no. 10, pp. 46–52, 2003.

[10] B. Benatallah, M. Dumas, and Z. Maamar, "Definition and Execution of Composite Web Services: The SELF-SERV Project," *IEEE Data Eng. Bull.*, vol. 25, no. 4, pp. 47–52, 2002.

[11] F. Leymann, "Web Services Flow Language (WSFL 1.0)," May 2001.

[12] S. Thatte, "XLANG - Web Services for Business Process Design. Microsoft Corporation," 2001.

[13] A. Arkin, "Business Process Modeling Language (BPML)," November 2002.

[14] "Web Services Business Process Execution Language Version 2.0 – OASIS Standard," Organization for the Advancement of Structured Information Standards (OASIS), Tech. Rep., Mar 2007.

[15] C. Pautasso and G. Alonso, "The jopera visual composition language," *Journal of Visual Languages and Computing (JVLC)*, vol. 16, pp. 119–152, 2005.

[16] J. Nitzsche, T. van Lessen, D. Karastoyanova, and F. Leymann, "BPELlight," in *BPM*, ser. Lecture Notes in Computer Science, G. Alonso, P. Dadam, and M. Rosemann, Eds., vol. 4714. Springer, 2007, pp. 214–229.

[17] "JOpera Project," http://www.iks.ethz.ch/jopera.

[18] A. Barros, M. Dumas, and A. ter Hofstede, "Service Interaction Patterns," in *BPM*, Vienna, Austria, Sept 2005.

[19] G. Decker, O. Kopp, and A. Barros, "An Introduction to Service Choreographies," *Information Technology*, vol. 50, no. 2, pp. 122–127, Februar 2008.

[20] "Business Process Modeling Notation (BPMN) Specification, Final Adopted Specification," OMG, Tech. Rep., Feb 2006, www.bpmn.org.

[21] C. Ouyang, M. Dumas, S. Breutel, and A. H. ter Hofstede, "Translating Standard Process Models to BPEL," in *CAiSE 2006*, Luxembourg, June 2006.

[22] J. M. Zaha, A. Barros, M. Dumas, and A. ter Hofstede, "A Language for Service Behavior Modeling," in *CoopIS*, Montpellier, France, Nov 2006.

[23] N. Kavantzas, D. Burdett, G. Ritzinger, and Y. Lafon, "Web Services Choreography Description Language Version 1.0, W3C Candidate Recommendation," Tech. Rep., November 2005, http://www.w3.org/TR/ws-cdl-10.

[24] A. Barros, M. Dumas, and P. Oaks, "A Critical Overview of WS-CDL," *BPTrends*, vol. 3, no. 3, 2005.

[25] G. Decker, O. Kopp, F. Leymann, and M. Weske, "BPEL4Chor: Extending BPEL for Modeling Choreographies," in *ICWS*, Salt Lake City, USA, July 2007.

[26] K. Pfitzner, G. Decker, O. Kopp, and F. Leymann, "Web Service Choreography Configurations for BPMN," in *Proceedings of the 3rd International Workshop on Engineering Service-Oriented Application (WESOA'2007)*, September 2007, Artikel in Tagungsband.

[27] ITU-T, "Message Sequence Chart," ITU-T," Recommendation Z.120, 2000.

[28] J. Clark, C. Casanave, K. Kanaskie, B. Harvey, N. Smith, J. Yunker, and K. Riemer, "ebXML Business Process Specification Schema Version 1.01," UN/CEFACT and OASIS, Tech. Rep., May 2001.

[29] A. Arkin, "Web Service Choreography Interface (WSCI) 1.0," Aug 2002.

[30] A. Banerji, C. Bartolini, D. Beringer, V. Chopella, K. Govindarajan, A. Karp, H. Kuno, M. Lemon, G. Pogossiants, S. Sharma, and S. Williams, "Web Services Conversation Language (WSCL) 1.0, W3C Note," March 2002.

[31] "Service Component Architecture," Open SOA Collaboration, March 2007. [Online]. Available: http://www.osoa.org/display/Main/Service+Component+Architecture+Home

[32] "SCA Assembly Model Specification 1.0," Open SOA Collaboration, March 2007. [Online]. Available: http://www.osoa.org/download/attachments/35/SCA_AssemblyModel_V100.pdf?version=1

[33] "SCA Policy Framework 1.0," Open SOA Collaboration, March 2007. [Online]. Available: http://www.osoa.org/download/attachments/35/SCA_Policy_Framework_V100.pdf?version=1

[34] F. Leymann and S. Pottinger, "Rethinking the Coordination Models of WS-Coordination and WS-CF," in *Third IEEE European Conference on Web Services (ECOWS 2005)*. IEEE Computer Society, November 2005, pp. 160–169.

[35] "Web Services Coordination (WS-Coordination) Version 1.1," OASIS, April 2007, http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.1-spec-os.pdf.

[36] "Web Services Atomic Transaction (WS-AtomicTransaction). Version 1.1," OASIS, April 2007, http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec-os.pdf.

[37] "Web Services Business Activity Framework (WS-BusinessActivity). Version 1.1," OASIS, April 2007, http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec-os.pdf.

[38] "Web Services Context Specification (WS-Context) Version 1.0," OASIS, April 2007, http://docs.oasis-open.org/ws-caf/ws-context/v1.0/wsctx.pdf.

[39] "Web Services Coordination Framework Specification (WS-CF) Version 1.0," OASIS.

[40] "Web Services ACID Specification (WS-ACID) Version 1.0," OASIS.

[41] "Web Services Business Process Specification (WS-BP) Version 1.0," OASIS.

[42] "Web Services Long Running Action Specification (WS-LRA) Version 1.0," OASIS.

[43] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, and T. Payne, "OWL–S: Semantic Markup for Web Services," World Wide Web Consortium, 2004, W3C Member Submission.

[44] H. Lausen, A. Polleres, and D. Roman, "Web Service Modelling Ontology (WSMO)," World Wide Web Consortium, 2005, W3C Member Submission.

[45] S. Battle, A. Bernstein, H. Boley, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, S. McIlraith, D. McGuinness, J. Su, and S. Tabet, "Semantic Web Services Language (SWSL)," World Wide Web Consortium, 2005, W3C Member Submission.

[46] E. Börger, "High Level System Design and Analysis Using Abstract State Machines," in *FM-Trends 98: Proceedings of the International Workshop on Current Trends in Applied Formal Method*. Boppard, Germany: Springer-Verlag, 1998, pp. 1–43.

[47] M. Grüninger and C. Menzel, "The Process Specification Language (PSL) Theory and Applications," *AI Magazine*, vol. 24, no. 3, pp. 63–74, 2003.

[48] J. Nitzsche, T. van Lessen, D. Karastoyanova, and F. Leymann, "BPEL for Semantic Web Services (BPEL4SWS)," in *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops*, ser. LNCS, Springer, Ed., vol. 4805/2007, 2007, pp. 179–188.

[49] D. Karastoyanova, T. V. Lessen, F. Leymann, J. Nitzsche, and D. Wutke, "WS-BPEL Extension for Semantic Web Services (BPEL4SWS), Version 1.0," Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, Universität Stuttgart, Institut für Architektur von Anwendungssystemen, Technischer Bericht Informatik 2008/03, April 2008. [Online]. Available: http://www.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=TR-2008-03&engl=0

[50] S. Bajaj, D. Box, D. Chappell, F. Curbera, G. Daniels, P. Hallam-Baker, M. Hondo, C. Kaler, A. Malhotra, and H. Maruyama, "Web Services Policy Attachment (WS-PolicyAttachment)," World Wide Web Consortium, 2006, W3C Member Submission.

[51] J. Farrell and H. Lausen, "Semantic Annotations for WSDL and XML Schema," World Wide Web Consortium, 2007, W3C Member Submission.

[52] S. Balzer, T. Liebig, and M. Wagner, "Pitfalls of OWL-S – A Practical Semantic Web Use Case," in *ICSOC' 04: Proceedings of the 2nd International Conference on Service Oriented Computing*, New York, NY, USA, November 2004, pp. 289–298.

[53] S. A. McIlraith and T. C. Son, "Adapting Golog for Composition of Semantic Web Services," in *KR*, D. Fensel, F. Giunchiglia, D. L. McGuinness, and M.-A. Williams, Eds. Morgan Kaufmann, 2002, pp. 482–496.

[54] N. Luo, J. Yan, and M. Liu, "Towards Efficient Verification for Process Composition of Semantic Web Services," in *IEEE SCC*. IEEE Computer Society, 2007, pp. 220–227.

[55] J. Mendling and M. Hafner, "From Inter-Organizational Workflows to Process Execution: Generating BPEL from WS-CDL," in *Proceedings of OTM 2005 Workshops. Lecture Notes in Computer Science 3762*, R. Meersman, Z. Tari, and P. H. et al., Eds. Springer Verlag, October 2005, pp. 506–515.

[56] "SCA Client and Implementation Model Specification for WS-BPEL," Open SOA Collaboration, March 2007. [Online]. Available: http://www.osoa.org/download/attachments/35/SCA_ClientAndImplementationModelforBPEL_V100.pdf?version=1

[57] S. Pottinger, R. Mietzner, and F. Leymann, "Coordinate BPEL Scopes and Processes by Extending the WS-Business Activity Framework," in *Proceedings of the 15th International Conference on Cooperative Information Systems (CoopIS 2007)*, ser. Lecture Notes in Computer Science, R. Meersman and Z. Tari, Eds., vol. 4803. Springer, November 2007, pp. 336–352.

[58] S. J. Mellor, A. N. Clark, and T. Futagami, "Guest Editors' Introduction: Model-Driven Development," *IEEE Software*, vol. 20, no. 5, pp. 14–18, 2003.

[59] K. Pfadenhauer, B. Kittl, and S. Dustdar, "Challenges and Solutions for Model Driven Web Service Composition," in *WETICE '05: Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 126–134.

[60] S. Ceri, P. Fraternali, and A. Bongio, "Web Modeling Language (WebML): A Modeling Language for Designing Web Sites," *Computer Networks (Amsterdam, Netherlands: 1999)*, vol. 33, no. 1–6, pp. 137–157, 2000.

[61] R. Acerbis, A. Bongio, M. Brambilla, and S. Butti, "WebRatio 5: An Eclipse-Based CASE Tool for Engineering Web Applications," in *ICWE*, 2007, pp. 501–505.

[62] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, IRVINE, 2000. [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[63] I. Manolescu, M. Brambilla, S. Ceri, S. Comai, and P. Fraternali, "Model-driven Design and Deployment of Service-enabled Web Applications," *ACM Trans. Interet Technol.*, vol. 5, no. 3, pp. 439–479, 2005.

[64] R. Hamadi and B. Benatallah, "A Petri net-based Model for Web Service Composition," in *ADC '03: Proceedings of the 14th Australasian database conference*. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2003, pp. 191–200.

[65] M. Dumas and A. H. M. ter Hofstede, "UML Activity Diagrams as a Workflow Specification Language," in *UML01: Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*. London, UK: Springer-Verlag, 2001, pp. 76–90.

[66] T. Mitra, "Business-driven Development," http://www-128.ibm.com/developerworks/webservices/library/ws-bdd/index.html, 2005.

[67] D. Skogan, R. Gronmo, and I. Solheim, "Web Service Composition in UML," in *EDOC '04: Proceedings of the Enterprise Distributed Object Computing Conference, Eighth IEEE International*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 47–57.

[68] R. Gronmo and M. C. Jaeger, "Model-Driven Semantic Web Service Composition," in *APSEC '05: Proceedings of the 12th Asia-Pacific Software Engineering Conference*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 79–86.

[69] J. Koehler, R. Hauser, R. Hauser, S. Kapoor, F. Y. Wu, and S. Kumaran, "A Model-Driven Transformation Method," in *EDOC '03: Proceedings of the 7th International Conference on Enterprise Distributed Object Computing*. Washington, DC, USA: IEEE Computer Society, 2003, p. 186.

[70] R. Hauser and J. Koehler, "Compiling Process Graphs into Executable Code," in *GPCE*, 2004, pp. 317–336.

[71] V. D. Castro, E. Marcos, and M. L. Sanz, "A Model Driven Method for Service Composition Modelling: A Case Study," *Int. J. Web Eng. Technol.*, vol. 2, no. 4, pp. 335–353, 2006.

[72] C. Ouyang, W. M. van der Aalst, M. Dumas, and A. H. ter Hofstede, "Translating BPMN to BPEL," BPM Center Report, Tech. Rep. BPM-06-02, 2006.

[73] B. Orriens, J. Yang, and M. P. Papazoglou, "Model Driven Service Composition," in *ICSOC: Proceedings of the International Conference on Service-Oriented Computing*, 2003.

[74] F. Rosenberg, C. Enzi, A. Michlmayr, C. Platzer, and S. Dustdar, "Integrating Quality of Service Aspects in Top-Down Business Process Development Using WS-CDL and WS-BPEL," in *EDOC '07: Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference*. Washington, DC, USA: IEEE Computer Society, 2007, p. 15.

[75] O. Kopp, B. Wetzstein, R. Mietzner, T. Unger, S. Pottinger, D. Karastoyanova, S. Michael, and F. Leymann, "A Model-Driven Approach to Implementing Coordination Protocols in BPEL," Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, Universität Stuttgart, Institut für Architektur von Anwendungssystemen, Technischer Bericht Informatik 2008/02, Mai 2008. [Online]. Available: http://www.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=TR-2008-02&engl=0

[76] K. Mantell, "From UML to BPEL," http://www.ibm.com/developerworks/webservices/library/wsuml2bpel/, 2005.

[77] T. Gardner, "UML Modelling of Automated Business Processes with a Mapping to BPEL4WS," in *Proceedings of the European Workshop on Web Services and Object Orientation*, 2003.

[78] R. Kong, "Transform WebSphere Business Integration Modeler Process Models to BPEL," http://www.ibm.com/developerworks/websphere/library/techarticles/0504_kong/0504_kong.html, 2005.

[79] J. Rao and X. Su, "A Survey of Automated Web Service Composition Methods," in *SWSWPC 2004: Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition*. San Diego, CA, USA: Springer, 2004, pp. 43–54.

[80] F. Casati, S. Ilnicki, L. jie Jin, V. Krishnamoorthy, and M.-C. Shan, "Adaptive and Dynamic Service Composition in eFlow," in *CAiSE '00: Proceedings of the 12th International Conference on Advanced Information Systems Engineering*. Stockholm, Sweden: Springer-Verlag, 2000, pp. 13–31.

[81] H. Schuster, D. Georgakopoulos, A. Cichocki, and D. Baker, "Modeling and Composing Service-Based nd Reference Process-Based Multi-enterprise Processes," in *CAiSE '00: Proceedings of the 12th International Conference on Advanced Information Systems Engineering*. Stockholm, Sweden: Springer-Verlag, 2000, pp. 247–263.

[82] J. Korhonen, L. Pajunen, and J. Puustjärvi, "Automatic Composition of Web Service Workflows Using a Semantic Agent," in *WI '03: Proceedings of the 2003 IEEE/WIC International Conference on Web Intelligence*. Halifax, Canada: IEEE Computer Society, 2003, p. 566.

[83] R. Akkiraju, K. Verma, R. Goodwin, P. Doshi, and J. Lee, "Executing Abstract Web Process Flows," in *Proceedings of the Workshop on Planning and Scheduling for Web and Grid Services of the ICAPS '04 Conference*, Whistler, British Columbia, Canada, 2004.

[84] S. Majithia, D. W. Walker, and W. A. Gray, "A Framework for Automated Service Composition in Service-Oriented Architectures," in *ESWS 2004: 1st European Semantic Web Symposium*, ser. Lecture Notes in Computer Science, vol. 3053. Heraklion, Greece: Springer, 2004, pp. 269–283.

[85] M. Lelarge, Z. Liu, and A. Riabov, "Automatic Composition of Secure Workflows," in *ATC*, ser. Lecture Notes in Computer Science, L. T. Yang, H. Jin, J. Ma, and T. Ungerer, Eds., vol. 4158. Springer, 2006, pp. 322–331.

[86] M. Carman, L. Serafini, and P. Traverso, "Web Service Composition as Planning," in *Workshop on Planning for Web Services in ICAPS'03*. Trento, Italy: AAAI, 2003.

[87] S. R. Ponnekanti and A. Fox, "SWORD: A Developer Toolkit for Web Service Composition," in *WWW 2002: The Eleventh International World Wide Web Conference*, Honolulu, Hawaii, USA, 2002.

[88] B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid, "Composing Web services on the Semantic Web," *The VLDB Journal*, vol. 12, no. 4, pp. 333–351, 2003.

[89] V. Alevizou and D. Plexousakis, "Enhanced Specifications for Web Service Composition," in *ECOWS '06: Proceedings of the European Conference on Web Services*. Zurich, Switzerland: IEEE Computer Society, 2006, pp. 223–232.

[90] M. Phan and F. Hattori, "Automatic Web Service Composition Using ConGolog," in *ICDCS Workshops*. IEEE Computer Society, 2006, p. 17.

[91] O. Aydin, N. K. Cicekli, and I. Cicekli, "Automated Web Services Composition with the Event Calculus," in *ESAW07: Proceedings of the 8th International Workshop in Engineering Societies in the Agents World*, Athens, Greece, 2007.

[92] R. J. Waldinger, "Web Agents Cooperating Deductively," in *FAABS '00: Proceedings of the First International Workshop on Formal Approaches to Agent-Based Systems-Revised Papers*. Greenbelt, MD, USA: Springer-Verlag, 2000, pp. 250–262.

[93] S. Lämmermann, "Runtime Service Composition via Logic-Based Program Synthesis," Doctoral thesis, KTH, Microelectronics and Information Technology, Stockholm, Sweden, 2002.

[94] J. Rao, "Semantic Web Service Composition via Logic-based Program Synthesis," Doctoral thesis, Norwegian University of Science and Technology, Faculty of Information Technology, Mathematics and Electrical Engineering, Trondheim, Norway, 2004.

[95] S. Abramsky, "Proofs as Processes," *Theoritical Computer Science*, vol. 135, no. 1, pp. 5–9, 1994.

[96] G. Bellin and P. J. Scott, "On the Pi-Calculus and Linear Logic," *Theoretical Computer Science*, vol. 135, no. 1, pp. 11–65, 1994.

[97] J. Rao, P. Küngas, and M. Matskin, "Logic-based Web Services Composition: From Service Description to Process Model," in *ICWS '04: Proceedings of the IEEE International Conference on Web Services*. San Diego, CA, USA: IEEE Computer Society, 2004, pp. 446–453.

[98] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL – The Planning Domain Definition Language," Yale Center for Computational Vision and Control, Technical Report CVC TR–98–003/DCS TR–1165, 1998, version 1.2.

[99] D. McDermott, "Estimated-Regression Planning for Interactions with Web Services," in *Sixth International Conference on AI Planning and Scheduling*. Toulouse, France: AAAI Press, 2002.

[100] J. Peer, "A PDDL Based Tool for Automatic Web Service Composition," in *PPSWR 2004: Second International Workshop on Principles and Practice of Semantic Web Reasoning*, ser. LNCS, vol. 3208/2004. St. Malo, France: Springer Berlin / Heidelberg, 2004, pp. 149–163.

[101] J. Rao, D. Dimitrov, P. Hofmann, and N. Sadeh, "A Mixed Initiative Approach to Semantic Web Service Discovery and Composition: SAP's Guided Procedures Framework," in *ICWS '06: Proceedings of the IEEE International Conference on Web Services*. Salt Lake City, Utah, USA: IEEE Computer Society, 2006, pp. 401–410.

[102] A. L. Blum and M. L. Furst, "Fast Planning Through Planning Graph Analysis," *Artificial Intelligence*, vol. 90, no. 1-2, pp. 281–300, 1997.

[103] Z. Wu, K. Gomadam, A. Ranabahu, A. P. Sheth, and J. A. Miller, "Automatic Composition of Semantic Web Services using Process and Data Mediation," in *ICEIS '07: Proceedings of the 9th International Conference on Enterprise Information Systems*, vol. 4. Funchal, Portugal: ICEIS Press, 2007, pp. 453–461.

[104] F. Lécué, E. Silva, and L. F. Pires, "A Framework for Dynamic Web Services Composition," in *WEWST07: Proceedings of the 2nd ECOWS Workshop on Emerging Web Services Technology*, C. Pautasso and T. Gschwind, Eds., vol. 313. Halle, Germany: CEUR, 2007.

[105] F. Lécué and A. Léger, "A Formal Model for Semantic Web Service Composition," in *International Semantic Web Conference*, ser. Lecture Notes in Computer Science, I. F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. Aroyo, Eds., vol. 4273. Springer, 2006, pp. 385–398.

[106] K. Erol, J. Hendler, and D. S. Nau, "Semantics for HTN Planning," UM Computer Science Department, Technical Report CS-TR-3239, 1994.

[107] K. Erol, J. A. Hendler, and D. S. Nau, "UMCP: A Sound and Complete Procedure for Hierarchical Task-network Planning," in *AIPS' 94: 2nd International Conference on AI Planning Systems*. Chicago, IL, USA: Morgan Kaufman, 1994, pp. 249–254.

[108] D. Wu, B. Parsia, E. Sirin, J. A. Hendler, and D. S. Nau, "Automating DAML-S Web Services Composition Using SHOP2," in *International Semantic Web Conference*, ser. Lecture Notes in Computer Science, D. Fensel, K. P. Sycara, and J. Mylopoulos, Eds., vol. 2870. Springer, 2003, pp. 195–210.

[109] Z. Jianhong, Z. Shensheng, C. Jian, and M. Yujie, "Improved HTN Planning Approach for Service Composition," in *SCC '04: Proceedings of the 2004 IEEE International Conference on Services Computing*. Shanghai, China: IEEE Computer Society, 2004, pp. 609–612.

[110] M. Klusch, A. Gerber, and M. Schmidt, "Semantic Web Service Composition Planning with OWLS-Xplan," in *Proceedings of the 1st Intl. AAAI Fall Symposium on Agents and the Semantic Web*. Arlington VA, USA: AAAI Press, 2005.

[111] J. Hoffmann, "A Heuristic for Domain Independent Planning and Its Use in an Enforced Hill-Climbing Algorithm," in *ISMIS '00: Proceedings of the 12th International Symposium on Foundations of Intelligent Systems*. Charlotte, North Carolina, USA: Springer-Verlag, 2000, pp. 216–227.

[112] I. Paik and D. Maruyama, "Automatic Web Services Composition Using Combining HTN and CSP," in *CIT '07: Proceedings of the 7th IEEE International Conference on Computer and Information Technology*. Aizu, Fukushima, Japan: IEEE Computer Society, 2007, pp. 206–211.

[113] D. Berardi, D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Mecella, "Automatic Composition of E-services That Export Their Behavior," in *ICSOC*, ser. Lecture Notes in Computer Science, M. E. Orlowska, S. Weerawarana, M. P. Papazoglou, and J. Yang, Eds., vol. 2910. Springer, 2003, pp. 43–58.

[114] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, and P. Traverso, "Planning and Monitoring Web Service Composition," in *AIMSA '04: Proceedings of the 11th International Conference on Artificial Intelligence, Methodology, Systems, Applications*, ser. LNCS, vol. 3192/2004. Varna, Bulgaria: Springer Berlin / Heidelberg, 2004, pp. 106–115.

[115] M. Pistore, P. Traverso, and P. Bertoli, "Automated Composition of Web Services by Planning in Asynchronous Domains," in *ICAPS '05: Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling*. Monterey, CA, USA: AAAI Press, 2005, pp. 2–11.

[116] M. Trainotti, M. Pistore, G. Calabrese, G. Zacco, G. Lucchese, F. Barbon, P. Bertoli, and P. Traverso, "ASTRO: Supporting Composition and Execution of Web Services," in *Demo papers of ICSOC '05: Proceedings of the Third International Conference on Service-Oriented Computing*, ser. LNCS, vol. 3826/2005. Amsterdam, The Netherlands: Springer Berlin / Heidelberg, 2005, pp. 495–501.

[117] M. Pistore, A. Marconi, P. Bertoli, and P. Traverso, "Automated Composition of Web Services by Planning at the Knowledge Level," in *IJCAI*, L. P. Kaelbling and A. Saffiotti, Eds. Professional Book Center, 2005, pp. 1252–1259.

[118] A. Marconi, M. Pistore, and P. Traverso, "Specifying Data-Flow Requirements for the Automated Composition of Web Services," in *SEFM*. IEEE Computer Society, 2006, pp. 147–156.

[119] A. Marconi, M. Pistore, P. Poccianti, and P. Traverso, "Automated Web Service Composition at Work: the Amazon/MPS Case Study," in *ICWS*. IEEE Computer Society, 2007, pp. 767–774.

[120] Y. Liu, A. H. Ngu, and L. Zeng, "QoS Computation and Policing in Dynamic Web Service Selection," in *Proceedings of the 13th International Conference on World Wide Web (WWW'04)*, 2004.

[121] F. Rosenberg, C. Platzer, and S. Dustdar, "Bootstrapping Performance and Dependability Attributes of Web Services," in *Proceedings of the IEEE International Conference on Web Services (ICWS'06), Chicago, USA*. IEEE Computer Society, 2006.

[122] O. Moser, F. Rosenberg, and S. Dustdar, "Non-Intrusive Monitoring and Adaption for WS-BPEL," in *Proceedings of the 17th International International World Wide Web Conference (WWW'08), Beijing, China*, Apr. 2008.

[123] L. Baresi and S. Guinea, "Dynamo: Dynamic Monitoring of WS-BPEL Processes," in *Proceedings of the International Conference on Service-Oriented Computing (ICSOC'05), Amsterdam, The Netherlands*. Springer, 2005, pp. 478–483.

[124] L. Baresi, S. Guinea, and M. Plebani, "Business Process Monitoring for Dependability," in *Proceedings of the Workshops on Software Architectures for Dependable Systems (WADS'06), Lecture Notes in Computer Science 4615*. Springer, 2006, pp. 337–361.

[125] L. Baresi, S. Guinea, and P. Plebani, "Policies and Aspects for the Supervision of BPEL Processes," in *Proceedings of the 19th International Conference on Advanced Information Systems Engineering (CAiSE'07), Trondheim, Norway*. Springer, 2007, pp. 340–354.

[126] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng, "Quality Driven Web Services Composition," in *Proceedings of the 12th International Conference on World Wide Web (WWW'03)*. New York, NY, USA: ACM Press, 2003, pp. 411–421.

[127] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-Aware Middleware for Web Services Composition," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 311–327, May 2004.

[128] I. Brandic, S. Pllana, and S. Benkner, "Specification, Planning, and Execution of QoS-aware Grid Workflows within the Amadeus Environment," pp. 331–345, Mar. 2008.

[129] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints," *ACM Trans. Web*, vol. 1, no. 1, p. 6, 2007.

[130] M. C. Jaeger, G. Rojec-Goldmann, and G. Mühl, "QoS Aggregation for Service Composition using Workflow Patterns," in *Proceedings of the 8th International Enterprise Distributed Object Computing Conference (EDOC'04)*. Monterey, California, USA: IEEE CS Press, September 2004, pp. 149–159.

[131] M. C. Jaeger, G. Mühl, and S. Golze, "QoS-aware Composition of Web Services: An Evaluation of Selection Algorithms," in *Proceedings of the Confederated International Conferences CoopIS, DOA, and ODBASE 2005 (OTM'05), Agia Napa, Cyprus*, ser. Lecture Notes in Computer Science (LNCS), vol. 3760. Springer, November 2005, pp. 646–661.

[132] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros, "Workflow Patterns," *Distributed and Parallel Databases*, vol. 14, no. 3, pp. 5–51, July 2003.

[133] G.Canfora, M. D. Penta, R. Esposito, and M. L. Villani, "An Approach for QoS-aware Service Composition based on Genetic Algorithms," in *Proceedings of the Genetic and Computation Conference (GECCO'05), Washington DC, USA*. ACM Press, 2005.

[134] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut, "Quality of Service for Workflows and Web Service Processes," *Journal of Web Semantics*, vol. 1, no. 3, pp. 281–308, 2004.

[135] M. C. Jaeger, G. Rojec-Goldmann, and G. Mühl, "QoS Aggregation in Web Service Compositions," in *Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05)*. Hong Kong, China: IEEE Press, March 2005.

[136] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani, "QoS-Aware Replanning of Composite Web Services," in *Proceedings of the IEEE International Conference on Web Services (ICWS'05), Orlando, FL, USA*, 2005, pp. 121–129.

[137] A. Mukhija, A. Dingwall-Smith, and D. S. Rosenblum, "QoS-Aware Service Composition in Dino," in *Proceedings of the Fifth European Conference on Web Services (ECOWS'05), Halle (Saale), Germany*, Nov. 2007, pp. 3–12.

[138] W. Wiesemann, R. Hochreiter, and D. Kuhn, "A Stochastic Programming Approach for QoS-Aware Service Composition," in *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'08), Lyon, France*, May 2008.

[139] G. J. Holzmann, *The SPIN Model Checker*. Addison-Wesley Pearson Education, 2003, http://spinroot.com.

[140] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri, "NUSMV: A New Symbolic Model Verifier," in *CAV*, 1999, pp. 495–499.

[141] M. Y. Vardi and P. Wolper, "An Automata-Theoretic Approach to Automatic Program Verification (Preliminary Report)," in *LICS*, 1986, pp. 332–344.

[142] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Compatibility Verification for Web service Choreography," *Web Services, 2004. Proceedings. IEEE International Conference on*, pp. 738–741, July 2004.

[143] J. Foster, S. Uchitel, J. Magee, and J. Kramer, "LTSA-WS: A Tool for Model-based Verification of Web Service Compositions and Choreography," in *ICSE '06: Proceedings of the 28th international conference on Software engineering*. New York, NY, USA: ACM, 2006, pp. 771–774.

[144] D. Bianculli, C. Ghezzi, and P. Spoletini, "A Model Checking Approach to Verify BPEL4WS Workflows," *Service-Oriented Computing and Applications, 2007. SOCA '07. IEEE International Conference on*, pp. 13–20, June 2007.

[145] Robby, M. B. Dwyer, and J. Hatcliff., "Bogor: A Flexible Framework for Creating Software Model Checkers," in *Testing: Academic & Industrial Conference - Practice And Research Techniques*, June 2006, technical Report SAnToS-TR2006-2.

[146] M. Mongiello and D. Castelluccia, "Modelling and verification of BPEL business processes," in *Fourth Workshop on Model-Based Development of Computer-Based Systems*, 2006.

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

[148] P. K. Pandya, "Specifying and Deciding Quantified Discrete-time Duration Calculus formulæusing DCVALID," in *Real-Time Tools, RTTOOLS*, 2001.

[149] X. Fu, T. Bultan, and J. Su, "Analysis of Interacting BPEL Web Services," in *WWW'04: Proceedings of the 13th International Conference on World Wide Web*. New York, NY, USA: ACM Press, 2004, pp. 621–630.

[150] R. Kazhamiakin, M. Pistore, and L. Santuari, "Analysis of Communication Models in Web Service Compositions," in *15th International World Wide Web Conference*. ACM Press, May 2006, pp. 267–276.

[151] X. Fu, T. Bultan, and J. Su, "Model Checking Interactions of Composite Web Services," UCSB Computer Science Department, Tech. Rep. 05, 2004.

[152] J. Arias-Fisteus, L. S. Fernandez, and C. D. Kloos, "Formal Verification of BPEL4WS Business Collaborations," in *E-Commerce and Web Technologies, 5th International Conference, EC-Web 2004, Proceedings*, ser. Lecture Notes in Computer Science, vol. 3182. Springer Verlag, 2004, pp. 76–85.

[153] S. Nakajima, "Model-Checking Behavioral Specification of BPEL Applications," in *Proceedings of the International Workshop on Web Languages and Formal Methods, WLFM 2005*, 2005.

[154] M. Koshkina and F. van Breugel, "Modelling and Verifying Web Service Orchestration by means of the Concurrency Workbench," *SIGSOFT Softw. Eng. Notes*, vol. 29, no. 5, pp. 1–10, 2004.

[155] J. Esparza, "Decidability and Complexity of Petri Net Problems – An Introduction," in *Lectures on Petri Nets I: Basic Models. Advances in Petri Nets*, ser. Lecture Notes in Computer Science, G. Rozenberg and W. Reisig, Eds., no. 1491, 1998, pp. 374–428.

[156] J. Esparza and C. Lakos, Eds., *Applications and Theory of Petri Nets 2002*, ser. Lecture Notes in Computer Science. Springer Verlag, 2002, vol. 2360.

[157] C. Ouyang, E. Verbeekb, W. M. van der Aalst, S. Breutel, M. Dumas, and A. H. ter Hofstede, "Formal Semantics and Analysis of Control Flow in WS-BPEL," *Science of Computer Programming*, vol. 67, no. 2-3, pp. 162–198, April 2007.

[158] B.-H. Schlingloff, A. Martens, and K. Schmidt, "Modeling and Model Checking Web Services," *Electronic Notes in Theoretical Computer Science: Issue on Logic and Communication in Multi-Agent Systems*, vol. 126, pp. 3–26, March 2005.

[159] H. Huang, W.-T. Tsai, R. Paul, and Y. Chen, "Automated Model Checking and Testing for Composite Web Services," in *8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005)*. IEEE Computer Society, 2005, pp. 300–307.

[160] S. Narayanan and S. A. McIlraith, "Simulation, Verification and Automated Composition of Web Services," in *WWW '02: Proceedings of the 11th international conference on World Wide Web*. New York, NY, USA: ACM, 2002, pp. 77–88.

[161] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini, "Validation of Web Service Compositions," *IET Software*, vol. 1, no. 6, pp. 219–232, December 2007.

[162] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Model-based Verification of Web Service Compositions," in *18th IEEE International Conference on Automated Software Engineering (ASE 2003)* . IEEE Computer Society, 2003, pp. 152–163.

[163] R. Kazhamiakin, M. Pistore, and M. Roveri, "A Framework for Integrating Business Processes and Business Requirements," in *Eighth IEEE International Enterprise Distributed Object Computing Conference*. IEEE Computer Society, 2004, pp. 9–20.

[164] A. Betin-Can, T. Bultan, and X. Fu, "Design for Verification for Asynchronously Communicating Web Services," in *WWW*, 2005, pp. 750–759.

[165] H. Huang, W.-T. Tsai, and R. Paul, "Automated Model Checking and Testing for Composite Web Services," *Object-Oriented Real-Time Distributed Computing, 2005. ISORC 2005. Eighth IEEE International Symposium on*, pp. 300–307, May 2005.

[166] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, M. Montali, S. Storari, and P. Torroni, "Computational Logic for Run-Time Verification of Web Services Choreographies: Exploiting the *OCS-SI* Tool," in *WS-FM*, 2006, pp. 58–72.

[167] M. Rouached, O. Perrin, and C. Godart, "Towards Formal Verification of Web Service Composition," in *Business Process Management*, 2006, pp. 257–273.

[168] N. Sharygina and D. Kröning, "Model Checking with Abstraction for Web Services," in *Test and Analysis of Web Services*, 2007, pp. 121–145.