Foundation for Research & Technology - Hellas

Institute of Computer Science

Information Systems Laboratory

Automated Web Service Composition: State of the Art and Research Challenges

George Baryannis and Dimitris Plexousakis

Technical Report ICS-FORTH/TR-409

October 2010

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Service-Oriented Architecture

In the last few years, Computer Science has observed a paradigm shift in the way software becomes available to end-users. The emergent paradigm is based on abstracting away from software and considering all resources as services that are available on-demand, forming what is called *Service-Oriented Computing (SOC)* [67]. SOC relies on services in order to facilitate the development of interoperable, dynamic, inexpensive and widely distributed applications. Services are entities that enable access to one or more capabilities, using a prescribed interface and in conformance to constraints and policies [26]. The interface as well as the constraints and policies are part of each service's description. Services are platform-independent and become available via a certain network, such as the Internet or the Universal Mobile Telecommunications System (UMTS). An important and distinguishing feature of services is the fact that they are loosely-coupled, allowing ad hoc and dynamic binding in contrast to a dynamically linked library, an assembly or the traditional linker that binds together functions to form an executable.

The service deployment model can be applied to any application component or any preexisting code so that they can be transformed into an on-demand, network-

available service. The general model of providers licensing applications to customers for a contractually-bound use as a service has been known as *Software as a Service (SaaS)* [10]. This practice is not limited to services, but can be applied to many other functions including communication, infrastructure and platforms, thereby leading to the concept of Everything as a Service which is the core of cloud computing [86].

Amidst this multitude of available services, the need for a set of design guidelines and principles in SOC endeavors is apparent. This need is satisfied through the *Service-Oriented Architecture (SOA)*. SOA is a style of design that guides all aspects of creating and using services throughout their lifecycle [63]. SOA enables an IT infrastructure to allow different applications to exchange data and participate in business processes, regardless of the underlying complexity of the applications, such as the exact implementation or the operating systems or the programming languages used to develop them.

1.2 Web Services

A SOA can be implemented using several different technologies such as SOAP [17], REST [29], CORBA [65] and Web services. Web services are considered the most promising service-oriented technology and are defined by the World Wide Web Consortium (W3C) as follows.

> A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. [15]

This definition gives a concise presentation of the advantageous characteristics of Web services, which we will briefly examine here.

The defining characteristic of a Web service is the use of the Internet and the World Wide Web as the communication medium for services to interact with each other and with service consumers. By using WWW, Web services exploit the existing URI infrastructure in order to be located by anyone having access to the Web. The URI scheme gives a name to each Web service which uniquely identifies it and allows one to use all existing operations on URIs in order to access it.

Web services make extensive use of the XML language [18]. From the definition of the messages exchanged between services to the service description, everything is based on XML, which is quite advantageous. XML is a simple language, both machine- and human-readable, with an intuitive hierarchical structure. It is also self-describing, as each XML data structure contains both a description of its structure and the content itself. Web services communicate by exchanging XML messages encoded using SOAP. SOAP defines a general pattern for Web service messages, some message exchange patterns, how XML information can be encoded in such messages and how these messages can be transported over the Internet using existing protocols such as HTTP or SMTP.

Web services have been the focus of several standardization efforts by global consortiums such as W3C and OASIS and have arguably been the most successful implementation of services, embraced by software corporations such as IBM and Microsoft which led to them being popular with traditional enterprize. Worldwide academia has also been a driving force behind Web service research.

1.3 Research on Web Services

The last decade has seen an exceptional wealth of research on topics related to Web services, especially by European researchers. The European Union 7th Framework Programme (FP7) has funded several service-related projects such as the Service Centric System Engineering Integrated Project (SeCSE), the Networked European

Software and Services Initiative (NESSI) projects and the Software Services and Systems Network of Excellence (S-Cube).

Research on Web services covers a multitude of issues that are involved throughout the life-cycle of a Web service or a Service-Based Application (SBA). This typically includes Web service description, discovery, composition, deployment, execution, monitoring and adaptation. Service discovery is the process of locating and gaining access to provided services that satisfy a set of requirements (whether it is capabilities or non-functional aspects). Service deployment involves concretely associating services to devices in the real world system that is used as the infrastructure for SBAs. Service execution follows deployment and is the result of invoking the deployed services or SBAs. Service monitoring comprises all the mechanisms that are required for capturing and processing several attributes of a service execution (related but not limited to quality aspects) to determine whether predefined agreements (such as Service-Level Agreements) are abided by and the execution goes as planned. Finally, service adaptation is required either to recover from a failure or to better satisfy the needs of a user in terms of functionality, application performance or resource management.

1.3.1 Web Service Description

Web service description deals with specifying all the information needed in order to access and use a service. The description should be rich, containing both functional and non-functional aspects of the service while it may also contain information on the internal processes of the service, depending on whether the service provider or owner decides to exposes such information or not. Service descriptions should also be written in a formal, well-defined language, allowing for automated processing and verification of the produced description documents. These characteristics are also extremely useful when attempting to compose Web services.

1.3.2 Web Service Composition

Web service composition involves combining and coordinating a set of services with the purpose of achieving functionality that cannot be realized through existing services. This process can be performed either manually or automatically (or semi-automatically in some cases), while it can occur when designing a composite service, hence producing a static composition schema or at run-time, when that particular service is being executed, leading to dynamic composition schemas.

1.3.2.1 Automated Web Service Composition

The process of creating a composition schema in order to satisfy a series of goals set by a requester is a really complex and multifaceted problem, since one has to deal with many different issues at once. First of all, it involves searching in an ever-growing global service repository in order to find matching services that may contribute to the complete satisfaction of the user's requirements. Assuming that these services have been found, one has to successfully combine them, resolving any conflicts and inconsistencies between them, since they most certainly will be created by different people using different implementation languages and systems. Since inconsistencies may occur at runtime, it may be necessary to predict such events so as to ensure that the system will run correctly. Finally, even after having overcome these issues, we have to be able to adapt to the dynamic characteristics of service-based systems, with services going offline, new services becoming online, and existing services changing their characteristics.

Attempting to overcome all these problems manually, will most certainly lead to a composition schema that is not fault-proof while the whole procedure will consume a lot of time and resources and cannot be considered scalable. Therefore, it is apparent that a certain degree of automation needs to be introduced to the composition procedure. Approaches that involve automation in the creation of the composition schema as well as during its execution constitute a major family known as automated

service composition. Automated service composition has been a "silver bullet" in Web service research and has attracted a great deal of researchers worldwide. The main focus of this document is to analyze and compare the most representative efforts in the area of automated Web service composition and identify possible gaps and deficiencies that can be the focus of future research.

1.4 Outline

The rest of this document is organized as follows. Chapter 2 presents a brief analysis of the major Web service description frameworks. This analysis is performed from the viewpoint of service composition, examining how these frameworks handle the description of composite services and what problems may arise.

Chapter 3 offers an extensive overview of the state of the art in automated Web service composition, presenting the most representative efforts in this area in recent years. The first part of the chapter deals with a set of requirements that should be satisfied by research efforts in automated Web service composition as well as several composition models that have been proposed and used by most researchers while the second provides a detailed description of specific research efforts. The final part of the chapter offers a detailed comparison of the works presented, based on several key aspects.

Finally, Chapter 4 explores the research challenges that were derived through the process of documenting the state of the art in automated Web service composition. The research gaps that are identified provide incentives and motivation that can drive future research in the area of automated Web service composition.

Chapter 2

Web Service Description

In this chapter, the most important efforts to handle the issue of describing Web services will be presented. This presentation will also focus on how these efforts handle the description of composite services. Then, we will briefly outline three major Artificial Intelligence problems that are directly related to Web service description, namely the Frame Problem and its Qualification and Ramification facets and see how they may affect Web service research in general.

2.1 Web Services Description Language (WSDL)

Web Services Description Language (WSDL) [22] is a W3C Recommendation, now in its second version, that has been established as the de facto description language for Web services. Figure 2.1 gives an overview of the elements that comprise a WSDL document.

The four fundamental elements of WSDL are the following:

- *types*: describes the kinds of messages that the service will send and receive.
- *interface*: describes what abstract functionality the Web service provides.
- *binding*: describes **how** to access the service.

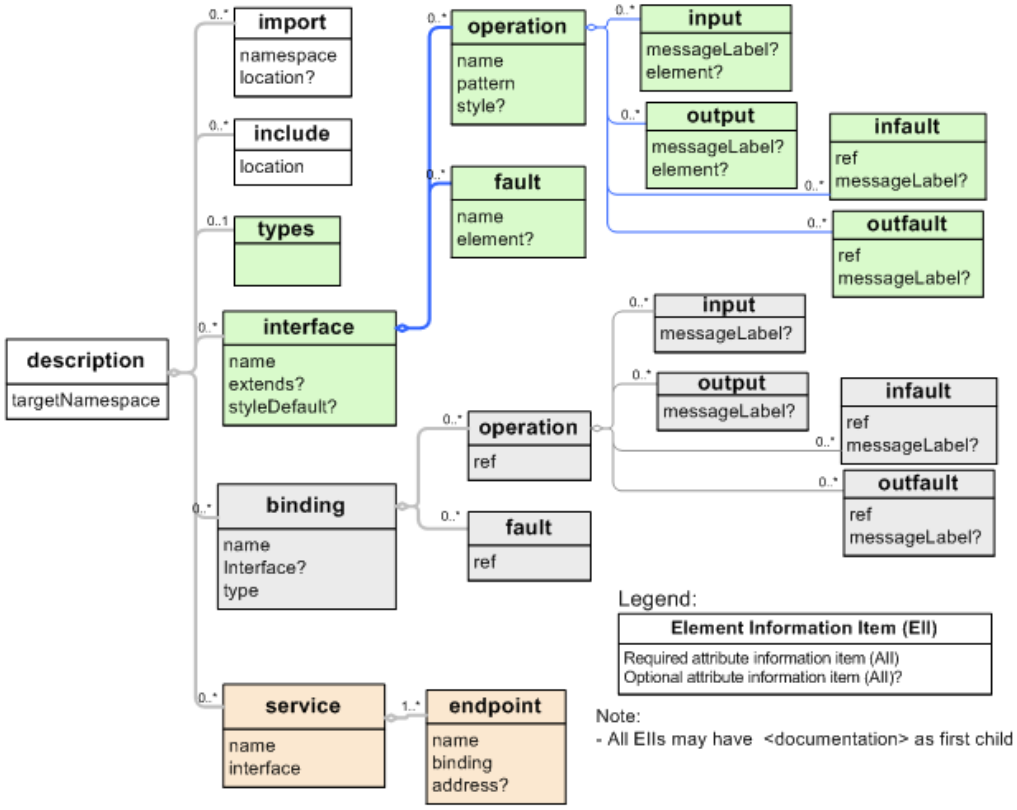

Figure 2.1: A representation of concepts defined by a WSDL document

- *service*: describes **where** to access the service.

2.1.1 WSDL Types

Message types can be defined in any schema definition language, with XML Schema being the one used in the WSDL 2.0 Primer. They can be explicitly defined inside the WSDL document or imported from an external schema document. In any case, all types that will be used in the description of the service must be defined prior to their inclusion in the description.

2.1.2 WSDL Interfaces

WDSL interfaces offer an abstract way of describing the service functionality, in contrast to the concrete representation offered by the binding and service elements that will be described later on. A WSDL interface consists of a set of operations, each one of them describing a simple interaction between the service and the client. The description is realized through a message exchange pattern that is followed when the particular operation is invoked. For each message contained in the pattern, a message type is specified, using the types that were defined previously in the WSDL document. WSDL 2.0 contains eight predefined message patterns but one can easily define new ones.

2.1.3 WSDL Bindings

A WSDL binding specifies concrete message format and transmission protocol details for an interface. Each operation in a WSDL description must be associated with a binding. Several typical binding extensions are defined in WSDL 2.0 such as SOAP binding or HTML binding, but the service provider is free to use others provided that they are supported by the client and the service toolkits. The syntax for bindings parallels the syntax of interfaces, since each interface construct has a binding counterpart. A binding can either be reusable, meaning that it is applicable to any interface, or not.

2.1.4 WSDL Services

A WSDL service element specifies a single interface that the service will support, and a list of endpoint locations where that service can be accessed. Each endpoint must also reference a previously defined binding to indicate what protocols and transmission formats are to be used at that endpoint. A service is only permitted to have one interface but there are workarounds for a service to be able to expose multiple

interfaces.

2.1.5 Critique on WSDL

WSDL, even in its second version, remains solely a language for the syntactic description of Web services. However, this poses severe limitations to Service-Oriented Architecture applications. To illustrate that fact, let's examine the following scenario. An application component in need for a specific service explicitly describes this need in a formal, concise and machine-understandable way and tasks a directory service with satisfying this need. The directory service then tries to match the description of the necessary service with the descriptions of the services it is aware of. A list of matches is compiled and a candidate service is selected and is automatically bound to the requesting application component. This scenario is not realizable because the service descriptions offered by WSDL are inadequate and incapable of handling such cases, due to their complete lack of semantics and composition support.

2.2 Semantic Annotations for WSDL (SAWSDL)

SAWSDL [28] is based on and shares the same design principles of previous work done by IBM and the LSDIS Lab of the University of Georgia and published as a W3C Member Submission with the title Web Service Semantics (WSDL-S) [2]. SAWSDL defines a way to semantically annotate WSDL interfaces and operations as well as XML Schema types, linking them to concepts in an ontology or a mapping document. The annotation mechanism is independent of ontology or mapping languages. The annotations are meant to help during Web service discovery, invocation and composition.

In Figure 2.2, the various elements of a WSDL description are shown and the ones that can be annotated in SAWSDL are marked clearly. As we can see, there are two basic semantic annotation constructs, through the extension attributes mod-

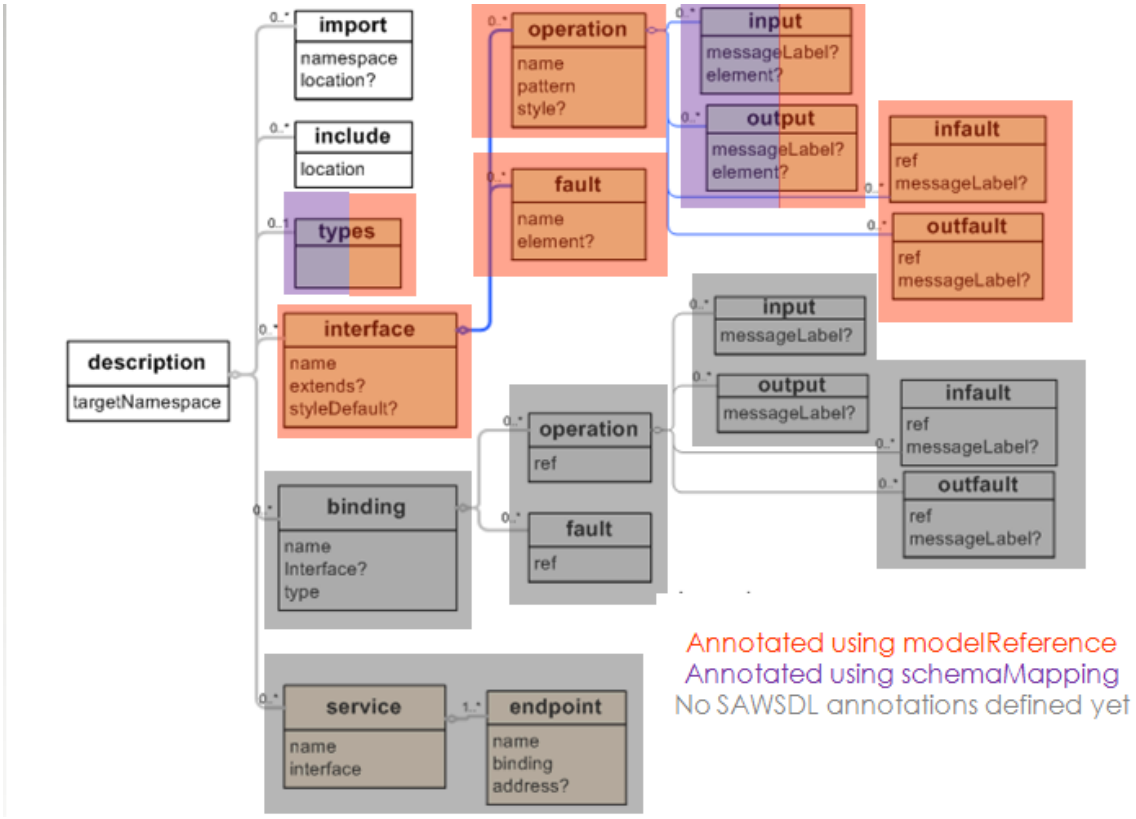

Figure 2.2: SAWSDL Annotations

elReference and schemaMapping. The **modelReference** attribute allows multiple annotations to be associated with a given WSDL or XML Schema component via a set of URIs. These URIs may identify concepts expressed in different semantic representation languages. SAWSDL does not define any particular way to obtain the documents linked by these URIs, it is simply recommended that the URIs resolve to a semantic concept definition document, which could be placed directly within the WSDL document, provided that it is expressed using XML. modelReference attributes can be used in WSDL interfaces, operations and faults, as well as XML Schema elements, types and attributes.

As far as the second annotation mechanism is concerned, two attributes are provided: **liftingSchemaMapping** and **loweringSchemaMapping**. These are used

to address post-discovery issues in using a Web service since there may still be mismatches between the semantic model and the structure of the inputs and outputs. Schema mapping relates the instance data defined by an XML Schema document with some semantic data defined by a semantic model (an ontology). Such mappings are useful in general when the structure of the instance data does not correspond trivially to the organization of the semantic data. While liftingSchemaMapping defines how an XML instance document is transformed to data that conforms to some semantic model, loweringSchemaMapping goes the opposite direction, defining how data in a semantic model is transformed to XML instance data.

2.2.1 Critique on SAWSDL

The SAWSDL extension can be applied to both WSDL 1.1 and WSDL 2.0 documents, although the latter case is more seamless than the first. Following from the ability of mapping WSDL 2.0 to RDF, SAWSDL-annotated WSDL documents can also be mapped to RDF. SAWSDL keeps the semantic model outside WSDL, making the approach independent from any ontology language. However, without describing how the use of annotations in different languages relate to one another, it is rather difficult, if not impossible to formally define requests, queries or matching between service requests and service descriptions. As a result, SAWSDL may be successful in annotating WSDL with semantic information, but does not offer any support for automated service discovery and composition. The efforts described in the following sections claim to do just that, among others.

2.3 OWL-S: Semantic Markup for Web Services

OWL-S [55] is the result of a collaborative effort by researchers at several universities and organizations, including University of Toronto, Yale University, University of Maryland at College Park and Carnegie Mellon University, as well as SRI Inter-

national and Nokia. The researchers' goal is to establish a framework within which these Web service descriptions are made and shared, employing a standard ontology, consisting of a set of basic classes and properties for declaring and describing services. The ontology structuring mechanisms of OWL provided them with an appropriate, Web-compatible representation language framework within which to realize their goal.

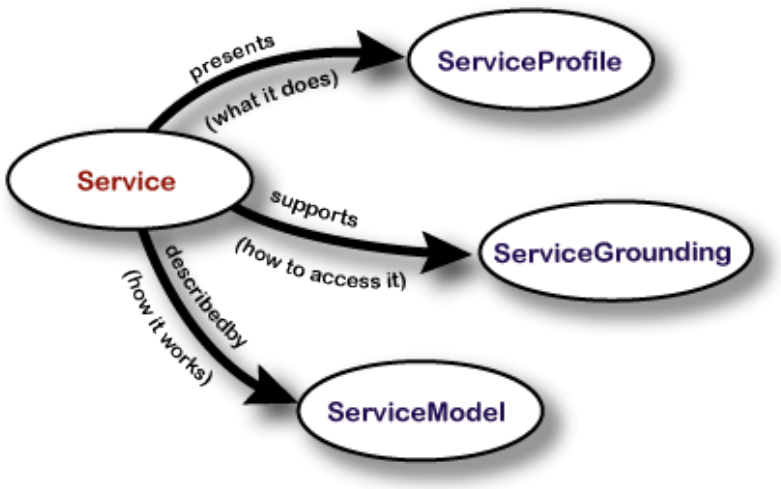

Figure 2.3: Top level of the OWL-S ontology

In Figure 2.3, the structuring of OWL-S in sub-ontologies is shown. This structuring of the ontology aims to provide three essential types of knowledge about a service, that are described in the rest of this section.

2.3.1 Service Profile

Service Profile allows service providers to advertise the services they offer, in such a way that the service requesters can easily find what they are looking for. Service providers can create their own specialized subclasses of Service Profile to suit their needs. The specification has a built-in subclass, Profile, which is just an example of that. The Profile subclass describes three dimensions of a web service: the service

provider, the service functionality and a set of service characteristics.

The first dimension of the Profile subclass deals with the entity that provides the service and contains contact information to anyone that may be associated with the service. The second dimension is essentially the functional description of the web service. It contains the inputs that are necessary for the service to be executed, the preconditions that must be met to ensure a valid execution, the information that is generated as the output of the service as well as any effects to the state of the world that result from the execution of the service. This set of information is referred to as IOPEs (Inputs, Outputs, Preconditions and Effects). Apart from these two dimensions, service providers are free to include any feature that they see fit to include in a service description.

A very important feature that should be part of a service description is the Quality of Service (QoS). QoS is a major factor in service discovery and selection, as searching only based on the functional description will yield services that may advertise the required operations, but may also be unreliable, slow or even malicious. OWL-S version 1.2 provides no properties relating to these features, leaving their specification entirely at the hands of service providers.

2.3.2 Service Model

The Service Model sub-ontology in OWL-S describes the service functionality and specifies the ways a client may interact with the service in order to achieve its functionality. This is done by expressing the data transformation with the inputs and the outputs and the state transformation with the preconditions and effects. Although the Profile and the Process Model play different roles during the Web service lifecycle they are two different representations of the same service, so it is natural to expect that the IOPEs of one are reflected in the IOPEs of the other. A Profile is usually a subset of a Service Model, containing only the information that is necessary in order to advertise the Web service.

Similarly to the Service Profile class, Service Model also has a built-in subclass, Process. Process contains all the IOPEs of a particular service and views a Web service similarly to a business process or a workflow. It is important to understand that a process is not a program to be executed, but a specification of the ways a client may interact with a service. A Process is further divided into three subclasses, atomic, simple and composite processes. An atomic process is a description of a service that expects one (possibly complex) message and returns one (possibly complex) message in response. It is the basic unit of implementation, is directly invocable by passing the appropriate parameters and is executed in a single step. A simple process provides, through an abstraction mechanism, specialized views of atomic processes or simplified representations of composite processes for purposes of planning and reasoning.

Composite processes are decomposable into other (non-composite or composite) processes; their decomposition can be specified by using control constructs such as Sequence, Split-Join or If-Then-Else, representing sequential, parallel and conditional composition schemas respectively. which are discussed below. A composite process is not a behavior a service will do, but a behavior (or set of behaviors) the client can perform by sending and receiving a series of messages. One crucial feature of a composite process is its specification of how its inputs are accepted by particular sub-processes, and how its various outputs are produced by particular sub-processes.

2.3.3 Service Grounding

While the Service Profile and Service Model describe what a service provide and its inner design and how it works, the grounding of a service specifies the details of how to access the service. Many different kinds of information are involved: protocol and message formats, serialization, transport, and addressing. The role of grounding is mainly to bridge the gap between semantic description of Web services and the existing service description models which are mainly syntactic, thus realizing process inputs and outputs as messages that are sent and received.

Following the trend set by the two other sub-ontologies, Service Grounding has a built-in subclass, Grounding, which offers a grounding of OWL-S to the most prominent syntactic service description language, WSDL. An OWL-S/WSDL grounding is based upon a series of correspondences between the two languages:

1. OWL-S atomic processes correspond to WSDL operations
2. OWL-S inputs and outputs correspond to WSDL input and output messages
3. OWL-S input and output types correspond to WSDL abstract types

The example of a grounding mechanism that involves OWL-S and WSDL that is presented in the OWL-S specification, explicitly states that both languages are required for the full specification of that mechanism. OWL-S alone or WSDL by itself cannot form a complete grounding specification. Both languages are indispensable in a grounding declaration and this enforces the notion of an OWL-S/WSDL grounding that uses OWL classes as the abstract types of message parts declared in WSDL, and then relies on WSDL binding constructs to specify the formatting of the messages.

2.3.4 OWL-Q: Extending OWL for QoS-based Web Service Description and Discovery

Although OWL-S provides a complete ontology for the description of Web services, it leaves the definition of QoS aspects to the service provider. In [46], the authors attempt to fill in that gap, by defining a semantic, rich and extensible QoS description model for Web services, called OWL-Q. OWL-Q is designed modularly, incorporating several independent facets, each one focusing on a particular part of QoS-based Web service description. There are facets regarding the connection of OWL-Q with OWL-S, the description of QoS offers and requests, the QoS metric model, the definition of constraints, to name a few. OWL-Q also incorporates reasoning by allowing the definition of rules to reason about class properties. OWL-Q is successfully used in

a Semantic Web service description and discovery framework implemented by the authors where several matchmaking algorithms based on OWL-Q descriptions are designed and evaluated.

2.4 Web Service Modeling Ontology (WSMO)

WSMO [78] is a conceptual model for describing various aspects related to Semantic Web services, produced by the ESSI WSMO working group which consists mainly of members of DERI and The Open University. The objective of WSMO and its accompanying efforts is to solve the application integration problem for Web services by defining a coherent technology for Semantic Web services. To formalize WSMO, the WSMO working group developed the Web service Modeling Language (WSML) and defined several variants of it, each one based on different formalisms. As illustrated in Figure 2.4, four main components are defined in WSMO and will be outlined in the rest of this section.

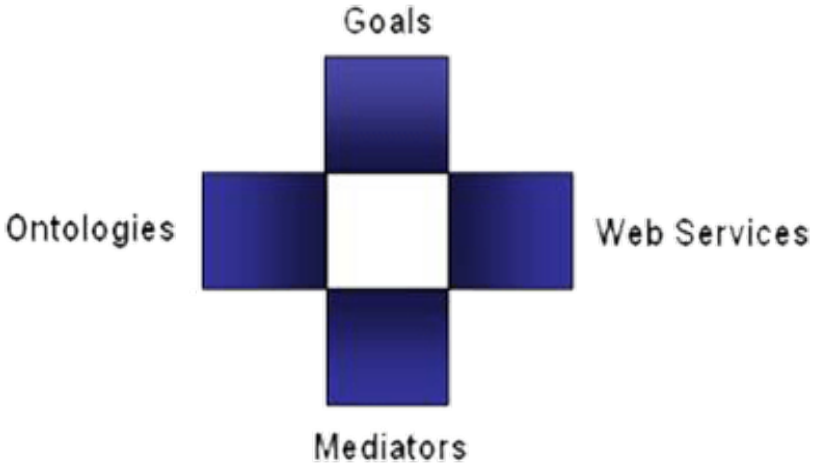

Figure 2.4: The four main WSMO components

2.4.1 WSMO Ontologies and Web Services

Ontologies provide domain specific terminologies for describing the other elements. The basic blocks of an ontology are concepts, relations, functions, instances, and axioms. Concepts, relations and instances play similar roles to the ones we described earlier in the general definition of an ontology. Axioms are specified as logic expressions and help to formalize domain specific knowledge. Functions are specialized relations that have a unary range.

Web services description in WSMO is realized using two different viewpoints: the interface and the capabilities. WSMO capabilities are similar to the IOPEs used in OWL-S. A capability should contain preconditions and assumptions that must be true before the execution of the service, postconditions that are true if the service has executed successfully, and effects that illustrate how the service execution changes the world.

A WSMO interface describes how the functionality of the Web service can be achieved. A twofold view of the operational competence of the Web service is offered: orchestration and choreography, Orchestration offers a description of how the overall functionality of the Web service is achieved by means of cooperation of different Web service providers, while choreography is essentially the description of the communication pattern that allows to one to consume the functionality of the Web service. WSMO allows more than one interface to be defined, so multiple choreographies can be defined for a Web service, hence allowing for multiple modes of interaction. Choreographies and orchestrations are based conceptually on Abstract State Machines, defining a set of states where each one is described by an algebra, and guarded transitions to express state changes by means of updates.

2.4.2 WSMO Goals and Mediators

WSMO Goals define exactly what a service requester demands from a Web service to offer. The requester (whether it is a human or an agent) defines the goals and

the corresponding system tries to find services or combinations of services that can realize them. In order to produce a complete goal description, we need to describe the interface and the capabilities of a Web service that, if it existed, it would completely satisfy our request. In this way, requests and services are strongly decoupled, since requests are based directly on goal descriptions.

Finally, Mediators connect heterogeneous components of a WSMO description when one encounters structural, semantic or conceptual incompatibilities. Mismatches can arise at the data or process level. Mediators are extremely useful in distributed and open environments such as the Internet, where vastly different objects must interoperate. Four types of mediators are defined. Ontology mediators help import elements of one ontology to another, goal mediators define at what level a goal is equivalent to another, Web service mediators assist in the interaction of different Web services and Web service-to-goal mediators can answer whether a Web service satisfies partly or completely a goal, or whether a Web service demands the satisfaction of a goal.

2.5 Semantic Web Services Framework (SWSF)

SWSF [8] is another effort to realize the Semantic Web service vision and is influenced by both OWL-S and WSMO. The Semantic Web Service Initiative is behind SWSF, with partners that also contributed in OWL-S, such as SRI International and University of Toronto and other important partners such as Hewlett Packard, Massachusetts Institute of Technology, Bell Labs Research and Stanford University. SWSF comprises of two major components, the Semantic Web Services Ontology (SWSO) and the Semantic Web Services Language (SWSL).

SWSL is used to specify formal characterizations of Web service concepts and descriptions of individual services. It includes two sub-languages. SWSL-Rules is based on logic-programming and rule-based reasoning and is used to support the use

of the service ontology in reasoning and execution environments, supporting a variety of tasks that range from service profile specification to service discovery, contracting, policy specification, and so on. The second sub-language of SWSL, SWSL-FOL, is based on first-order logic and is used primarily to express the formal characterization of Web service concepts. Both sub-languages are layers to facilitate learning and to be able to support varied specialized tasks that do not require the full expressive power of the language.

2.5.1 Semantic Web Services Ontology (SWSO)

SWSO presents a conceptual model by which Web services can be described. Similarly to OWL-S, SWSO is divided into three major components: Service Descriptors, Process Model and Grounding. **Service Descriptors** are the equivalent to OWL-S Service Profile and WSMO non-functional properties and contain basic identifying information for the service, such as the service name, author, contact information and so on. The set of properties can be expanded to include other properties, domain-specific or not.

The SWSO **Process Model** provides an abstract representation for Web services, their impact on the world and the transmission of messages between them. It is based on the Process Specification Language [38], a formally axiomatized ontology that was originally developed to enable sharing of descriptions of manufacturing processes. The fundamental building block of an SWSO Process Model is the atomic process, which is directly invocable, has no sub-processes and can be executed in a single step. An atomic process is associated with a set of IOPEs, which are expressed using fluents. However, inputs and outputs are treated as special kinds of preconditions and effects. An input is a knowledge precondition while an output is a knowledge effect.

Service composition is modeled by complex activities using a set of control constraints (similar to the OWL-S control constructs), ordering constraints (specifying just the order of execution) occurrence constraints (linking constraints with a spe-

cific activity occurrence) and state constraints (associating triggered activities with states). IOPEs are defined for atomic processes only. The IOPEs from complex activities are inferred by the atomic processes that constitute the activity. As far as messages are concerned, SWSO differentiates itself from OWL-S and WSMO, defining explicit objects for both messages and channels, the latter giving some structure to how messages are transmitted between services.

Finally, SWSO **Grounding** allows abstract SWSO service descriptions to be linked with one or more concrete realizations, such as WSDL. The central role of an SWSO/WSDL grounding is to provide an execution environment with the information it needs at runtime to send and receive messages associated with WSDL service descriptions. The following series of correspondences between SWSO and WSDL are defined:

1. SWSO complex activities correspond to WSDL operations
2. WSDL message patterns corresponds to the patterns of messages contained within the control structure of complex activities
3. SWSO messages are serialized as WSDL messages, using XML schema for message types

2.6 Comparison

Before completing the talk on Web services description, we offer a summary and comparison of the languages presented. We omit WSDL and SAWSDL from the comparison, because the former lacks any support for semantic description and the latter is not a fully fledged semantic framework as OWL-S, WSMO and SWSF are, and as a result, they can't be compared on the same basis.

From the languages presented in this section, OWL-S is the most mature one and the most commonly used in service discovery and composition research approaches.

OWL-S, however, doesn't come without its drawbacks, as it is pointed out in [6], which prevent it from being used in practical real-world scenarios. For example, its process model is neither an orchestration model nor a choreography model. It is essentially a behavioral description of the service. Moreover, OWL-S views Web service execution as a collection of remote procedure calls, which applies only to synchronous communication, leaving asynchronous communication out of the picture.

WSMO also has been used in many research approaches, even though it hasn't been around as long as OWL-S. The process model of WSMO offers both an orchestration and a choreography view. However, the orchestration view is rather primitive, and the WSMO Choreography model contains transition rules that only represent local constraints. In addition, WSMO Choreography needs to be formalized in order to allow reasoning and verification about it.

As far as SWSF is concerned, it is a very ambitious approach and has the most elaborate formalization of the three semantic framework efforts and relies on tried and proven foundations for both the included ontology and the language. However, most of its features are present in either OWL-S or WSMO, without necessarily keeping the best of both efforts. In Table 2.1 a comparison between OWL-S, WSMO and SWSF is provided, based on the following criteria:

- The support for Semantic Web services discovery
- The process modeling support (i.e. orchestration, choreography, behavioral interface)
- The formalisms to which the behavioral semantics are mapped
- The support of synchronous and/or asynchronous inter-process communication
- The existence of research approaches on composition using the languages

Semantic Web Service Frameworks	SWS Discovery Support	Process Model	Behavioral Semantics	Inter-Process Communication	Existing Composition Approaches
OWL-S	Yes	Limited Orchestration	Situation Calculus, Petri Nets	Synchronous	Yes
WSMO	Yes	Choreography	ASMs	Both	No
SWSO	Yes	Both	Same as OWL-S	Synchronous	No

Table 2.1: Comparison of Semantic Web Service Frameworks

2.7 The Frame, Ramification and Qualification Problems

The discussion so far in this chapter aimed to illustrate the fact that supporting the automation of tasks such as Web service discovery and composition requires the employment of rich, formal, semantic description languages and frameworks, such as OWL-S, WSMO and SWSF. One of the unifying characteristics of these three efforts is that they employ the precondition/postcondition notation in order to formally express a service specification. As it was first identified in [16], such formal specifications are prone to a family of problems related to the frame problem [42], a long-standing problem in the field of Artificial Intelligence. In a previous work [7], we identified the existence of the frame problem in Web service specification and proposed a solution inspired by the work on procedure specifications in [16]. In the next section, we will summarize the results of that work.

2.7.1 Addressing the Frame Problem

The frame problem stems from the fact that including clauses that state only what is changed when preparing formal specifications is inadequate since it may lead to inconsistencies and compromise the capacity of formally proving certain properties of specifications. One should also include clauses, called frame axioms, that explicitly state that apart from the changes declared in the rest of the specification, nothing else changes. Solving the frame problem means expressing frame axioms without resulting in extremely lengthy, complex, possibly inconsistent, obscure specifications and at the same time retaining the ability of proving formal properties of the specifications.

The proposed solution involves declaring, for each element of the service specifications we are creating, which services may result in changing them. Thus, we don't aim to write a set of frame axioms for each individual Web service specification, but we create assertions that explain the circumstances under which each predicate or function might be modified from one state to another. These assertions, called explanation closure axioms or change axioms, provide a state-oriented perspective to specifications. To be able to express the change axioms, a simple extension to the first-order predicate logic is proposed, that adds a special predicate symbol, named Occur and a special variable symbol named α. The semantics for these two additions are simple. Variable α is used to refer to services taking part in the specification. Occur(α) is a predicate of arity 1 that is true if and only if the service denoted by the variable α has executed successfully.

2.7.1.1 Expressing change axioms in Semantic Web Service languages

Change axioms can be expressed in all Semantic Web service frameworks by leveraging the rule languages included or supported by them. We will briefly outline the cases of OWL-S, WSMO and SWSO. In OWL-S, the most prominent Semantic Web service framework, rules are expressed using SWRL [40]. SWRL-FOL [69], a first-order logic extension to the SWRL syntax is more appropriate for our purposes. In SWRL-FOL,

the Occur predicate can be expressed as a unary predicate which has the meaning that its argument belongs to a certain OWL class. The variable α can then be expressed as an individual variable. For an example, see the attached paper in the Appendix.

As far as WSMO is concerned, the language used for expressing rules is WSML. WSML uses similar constructs to SWRL in order to express simple logical expressions such as the change axioms of our proposed solution. Thus, we again express Occur as a unary predicate and the variable α as an individual variable. Finally, SWSO includes SWSL as its de-facto rule language which contains two sub-languages: SWSL-FOL and SWSL-Rules. SWSL-FOL, a full first-order logic language is more than adequate for expressing change axioms.

2.7.1.2 An algorithm for producing change axioms

Having examined the ways of expressing a change axiom, we turn our focus on sketching an algorithm for automatically producing change axioms, given a service description using the precondition/postcondition notation. A complete set of change axioms should contain one axiom for every predicate contained in all the preconditions and postconditions stated in the description. Thus, we need to search the condition clauses to find every distinct predicate included. For each one of these predicates, our actions depend on whether a corresponding change axiom already exists. If it exists, we don't need to add a new one, since we already have expressed whether the corresponding service changes that particular predicate or not. If it doesn't exist, we need to add one that reflects the change in the value of the predicate.

For the case of composite service specifications, the algorithm needs to check each participating service specification separately to create change axioms for the predicates it contains. Also, if a change axiom already exists for a predicate, it doesn't mean that we don't need to modify it, since it may have been added in a previous step, when we were dealing with a different service. As a result, we need to do a separate search for predicates in each participating service specification, and

for each predicate we find, we should check if the corresponding change axiom has already been added.

2.7.2 The Ramification Problem

The frame problem is closely related to two other problems in logic-based knowledge representation, the ramification and qualification problems. In [59], the author defines the ramification problem as the problem of adequately representing and inferring information about the knock-on and indirect effects (also known as ramifications) that might accompany the direct effects of an action or event. The relation of the frame and ramification problems is somewhat contradicting: if one deals with the frame problem by expressing that no other effects are allowed except the ones explicitly stated, then any indirect effects are disallowed by definition, hence any solution to the ramification problem is precluded.

It would be interesting to explore the ramification problem and its consequences for the case of Web services. In that context, one may deal with effects that are caused by effects of a service execution, i.e. executing a service affects the world in a certain way, which then leads to secondary effects that are derived from the primary ones. Finding a way to include such effects in a formal Web service specification while at the same time avoiding any conflicts with the aforementioned solution to the frame problem poses an interesting challenge.

2.7.3 The Qualification Problem

While the ramification problem deals with the effects of an action, the qualification deals with the circumstances and the conditions that must be met prior to the execution of an action. In [59], the author separates two major aspects, the endogenous and the exogenous qualification problem. The endogenous qualification problem is the problem (and sometimes the impossibility) of listing all preconditions (also known as qualifications) that must be satisfied for an action to have the desired effect and

at the same time updating these qualifications as new statements become part of our knowledge, without resulting in inconsistencies. On the other hand, the exogenous qualification problem deals with qualifications that are outside the scope of our theory and result in contradicting some effects of an action due to inconsistencies.

In the context of Web services, it can be argued that even if one considers a Web service specification as complete, there might always be some conditions that have not been explicitly declared. Trying to declare more and more preconditions may then lead to a service that will never be able to be executed, or to specifications that are rendered inconsistent once a new statement becomes part of our knowledge. Moreover, in the case of service composition inconsistencies may arise when the qualifications of a new service added to a composition schema contradict the effects of the composite service that is being modified. This is an example of the effects of the exogenous (rather than the endogenous) aspect of the qualification problem, indicating that the field of Web services offers a interesting context in which the effects and possible solutions to the qualification problem can be explored.

We will continue the discussion on the ramification and qualification problems in the final chapter where we will formulate some research challenges through examples of the effects these problems have in formal Web service specifications.

Chapter 3

State of the Art

In this chapter, a comprehensive and comparative analysis of the most representative efforts in automated Web service composition will be presented. We will first identify some requirements that need to be met, followed by a brief analysis of the most popular composition models that have been proposed. Then, a complete state-of-the-art of automated Web service composition will be presented, organized in four major categories of approaches. At the end of the chapter, a comparison of the approaches will be attempted, based on how they satisfy the requirements we have identified.

3.1 Requirements for Automated Web Service Composition

A considerable number of surveys on Web service composition (automated or not) have been conducted and published by various researchers all over the world, over the last decade [45] [58] [77] [41] [25] [47] [1] [52]. Most of these surveys don't state clearly what requirements need to be met for an approach to successfully solve the problem of automated Web service composition and none of them contains a comprehensive comparison based on said requirements. In this section, we will present and briefly analyze a set of requirements that must be satisfied in order for a composition process

to be considered successful, accurate and complete.

3.1.1 Automation

Since the focus of our survey is automated Web service composition, the primary requirement is that the generation of the composition schema must be at least partially (if not fully) automated. The main purpose behind designing an approach to handle Web service composition is to minimize user intervention and accelerate the process of producing a composite service that satisfies preset user requirements. Automation reduces the time spent in order to create a composition schema compared to the time required in a manual composition approach, eliminates human errors and reduces the overall cost of the process. Thus, we should expect that a successful Web service composition approach provides the highest possible level of automation.

3.1.2 Dynamicity

A defining characteristic of a service composition approach is whether it produces a static or a dynamic composition schema. This distinction is closely related to the time in which the composition schema is created, at design-time or at run-time. A static composition approach involves selecting the component services to be used linking them together to form a composition process and finally deploying the composite process. This process cannot be modified in any way after the execution has begun, hence the term design-time composition that is often attributed to static composition approaches.

On the other hand, dynamic composition approaches produce an abstract composition schema, that is essentially a composition process without the actual service bindings. The abstract composition schema is concretized at run-time, when service endpoints are bound to each abstract task in the schema. Dynamicity ensures that a produced composition schema will be consistent and executable long after its initial design. A dynamic composition schema is able to overcome issues such as services no

longer being provided and services being replaced by new ones by providers, which would render a static composition schema invalid, inconsistent and impossible to execute. For that reason, dynamic (or run-time) composition approaches are preferable to static (or design-time) ones.

3.1.3 Semantic capabilities

As it was argued in Chapter 2, semantic capabilities are fundamental for the realization of Service-Oriented Architectures. Semantically rich descriptions of services and composition goals can be utilized by applications or other services without human assistance, facilitating automated composition. Moreover, they allow for more efficient service matchmaking based not only on service inputs and outputs and, as a result, for composite services that are more close to what the user requests. Thus, effective service composition approaches should exploit semantic descriptions to realize their goals.

3.1.4 QoS-Awareness

Another fundamental requirement for automated Web service composition is QoS-awareness. QoS-aware approaches take into account not only functional characteristics of services but also non-functional ones, dealing with quality aspects such as response time, price, availability and so on. Considering QoS aspects when deciding which services to include in a service composition schema is important when functional requirements are satisfied by more than one service. As a result, composite services produced by QoS-aware approaches not only offer the capabilities requested by the user but also guarantee the best possible quality.

3.1.5 Nondeterminism

Nondeterminism involves cases where an action may lead to more than one different states depending on the values of some parameters. For the case of service composition, nondeterminism is observed when the composition schema includes choice constructs (such as if-then-else) or loops (such as repeat-while). Nondeterminism may lead to an increase in the number of possible composition schemas for a given request and must be taken into account when designing a composition approach.

3.1.6 Partial observability

Real-world applications bring several requirements to the table that are not obvious when examining the composition problem from a theoretical point of view. One of these requirements is partial observability which involves dealing with incomplete information of the initial world state. This requirement is strongly linked to approaches that are based on AI planning techniques, since most of these techniques require a complete knowledge of the initial world state, which is not readily available on real-world application. Thus, an effective composition approach must deal with incomplete (or in some cases wrong) information and manage to produce composite services despite that fact.

3.1.7 Scalability

Another requirement brought by real-world applications is scalability. The fact that a composition approach works well given a set of services is no guarantee that will work as effectively with a different, larger or more complex set of services. Composition approaches must be tested against increasingly large service registries to examine how their performance is affected. A common trade-off often arises in this context, that of scalability versus performance. It is important to identify parts of the approach that may pose limitations and try to work them out in order to ensure the maximum

scalability for a given performance or vice-versa.

3.1.8 Correctness

Composition correctness is required when we want to check if certain properties of the produced composite service hold, such as the fact that it is guaranteed to produce a certain set of outputs given a certain set of inputs and under a certain set of conditions. Correctness is established through verification techniques, which have also been well researched by the Web service community.

3.1.9 Domain independence

A composition approach should not be limited to a specific domain (unless of course, the focus of the research was exclusive to that domain from the start). One should be able to apply the same approach to different domains, allowing for the solution of a broad range of problems. This requirement may be difficult to achieve while at the same time achieving the requirement of semantic capabilities, since, in some cases, semantic knowledge is domain-dependent.

3.1.10 Adaptivity

The last requirement that we will examine has recently been attracting interest in research communities and involves the ability of a service (atomic or composite) to adapt itself. Adaptation is the process of modifying Service-Based Applications in order to satisfy new requirements and to fit new situations dictated by the environment on the basis of predefined adaptation strategies. Adaptation goes one step further from dynamic composition approaches, in the sense that the former also deals with changes in the requirements set by the requester which the latter cannot handle.

Adaptation can be proactive, aiming to modify an application before a deviation will occur during the actual operation and before such a deviation can lead to

problems, or reactive, handling faults and recovering from problems reported during execution. While we include adaptivity as a requirement for automated Web service composition, that doesn't imply that it should be set as a research goal for such an approach. Instead, the composition researcher may employ research results from the adaptation domain, so that the composite services resulting from the approach designed will have the ability to adapt themselves.

3.2 Composition Models

Before we present the most important approaches that have been proposed for the realization of automated service composition, we will take a look at the most well-adopted composition models that have been defined. These models are used by most approaches that form the state-of-the-art of the next section and some of them are supported by the Semantic Web service frameworks presented in Chapter 2. It should be noted that these models are not used exclusively: one approach can implement more than one models at the same time.

3.2.1 Orchestration

As defined in [68], orchestration is a description of how the services that participate in a composition interact at the message level, including the business logic and the order in which the interactions should be executed. Thus, an orchestration should define which message is sent when and by which participating service. A service orchestration can be considered as being proactive, as it defines the interactions with the services it orchestrates, before any actual execution takes place.

Several researchers [41] [47] differentiate between composition synthesis and orchestration. Composition synthesis concerns synthesizing a specification of how to coordinate the component services to fulfill the client request, generating a plan that dictates how to achieve a desired behavior by combining the abilities of multiple ser-

vices. On the other hand, orchestration is about executing the result of composition synthesis by coordinating the control and data flow among the participating services and also about supervising and monitoring that execution.

Service orchestrations are typically described and executed using workflow languages. The most prominent and universally adopted language for describing service orchestration is Business Process Execution Language (WS-BPEL) [64]. BPEL is an XML-based language for representing the business logic, in other words the data flow and control flow of processes. The control flow may contain alternative execution paths, exception and fault handling, event handling and additional rules and constraints. Apart from that, BPEL processes contain participants (Web services) that are assigned to each activity contained in the process. In its official document, only WSDL descriptions may be associated with a BPEL process, which limits BPEL to static and syntactic (without semantics) service compositions. However, there have been research efforts that aim to address these deficiencies and provide support for dynamicity and semantics.

3.2.2 Choreography

Choreography is associated with the globally visible message exchanges, rules of interaction and agreements that occur between multiple business processes. This is their main defining characteristic, as they differ from orchestration which typically specify a business process executed by a single party. In choreographies, partners are in full control of their internal business processes and do not expose them to other partners, unless they are essential to the communication. Choreographies are conceptually related to conversations, which are defined as a message exchange between two partners that follows the rules of the overall choreography.

The principal language for defining choreographies is Web services Choreography Description Language (WS-CDL) [43]. WS-CDL uses bi-lateral interactions with one or two messages to incrementally model any interaction. The basic building blocks

are request-only, response-only, and request-response interactions. There is a limited set of control flow constructs (sequence, parallel, choice and work-unit) in order to represent the composition of interactions.

Although orchestration and choreography can be used separately to implement a service composition, their different viewpoints can be combined for a more complete representation. Orchestration can be used to describe participating services in the lower abstraction level and choreography can give a higher level description of how these orchestrations interact with each other and capture the complex conversations between them in order to realize the goal set by the requester. Of course, one can argue that the higher level description can also be realized using an orchestration model, but this would automatically imply that composition synthesis is involved for the creation of the process description of the orchestration, which may not be what the partners want, hence leaving choreography as the most suitable choice.

3.2.3 Coordination

Service coordination involves temporarily grouping a set of service instances following a coordination protocol. This protocol dictates the way participating services interact and also the outcome of the interaction, whether it was successful or not. The defining characteristic of coordination is the existence of a third party, called the coordinator, who acts as the nexus between participants and ultimately is responsible for the upholding of the protocol and the decision and dissemination of the outcome, once the activity ends. The participating services typically do not communicate directly with each other in a coordination model, all communication involves the coordinator.

OASIS has developed the Web Services Composite Application Framework (WS-CAF) [30] which includes, among others, Web Services Coordination Framework (WS-CF). In WS-CF, a coordination is defined based on three components, the coordinator, the participants and the coordination service. The coordinator provides an interface for the participants to register, coordinates the participants according to the protocol

and disseminates the protocol outcome. The participants can be single Web Services or composite ones. The coordination service is merely the implementation of the coordination protocol.

3.2.4 Component Model

The final model that we will examine is the Component model, which is sometimes referred to as service wiring. Service wiring involves the actual linking between inputs and outputs when services are being composed together. An indicative example of service wiring is Service Component Architecture (SCA) [23]. In this architecture, service components can be implemented in any programming language available and encapsulated so that all components share a similar description. Service components can be wired together to form a so-called Composite. Composites then can operate as service components themselves. The component model can be used together with an orchestration with the latter describing the control and data flow between participating services and the former implementing the actual input and output connections from one service to another.

3.3 Automated Web Service Composition Approaches

Approaches to the automated Web service composition problem have been exceptionally diverse and offer different interpretations of what should be addressed in a composition approach. They also differ on the degree of automation that is involved in the process ranging from semi-automated to fully automated approaches. Since our survey focuses on automated Web service composition, we will omit any purely manual approaches.

Due to the multitude and diversity of existing research approaches, it is imperative that we perform some kind of grouping in order to make the presentation of the approaches easier. Four distinct groups of approaches will be examined:

1. **Workflow-based:** approaches that exploit knowledge from workflow research

2. **Model-based:** approaches that use modeling languages (Petri-nets, UML, FSMs) to represent service compositions

3. **Mathematics-based:** approaches that use mathematics (logics, calculi, algebras)

4. **AI planning:** approaches that handle service composition as an AI planning problem

3.3.1 Workflow-based Approaches

Workflow organization and management have been a major research topic for more than twenty years. As a result, there has been a lot of effort on how to represent a sequence of actions. Drawing mainly from the fact that a composite service is conceptually similar to a workflow, it is possible to exploit the accumulated knowledge in the workflow community in order to facilitate Web service composition. Composition frameworks based on workflow techniques were one of the initial solutions proposed for automatic Web service composition. Initially, most works focused on static and manual compositions. More recent work, however, has attempted to realize automated Web service composition. Due to the popularity of BPEL, most approaches in this category employ BPEL in one way or another.

Majithia et al. [51] present a framework that automatically construct a Web service composition schema from a high-level objective. The input objective is fed to an abstract workflow generator that attempts to create an abstract workflow (written in BPEL) that satisfies the objective, either by using already generated workflows or subsets of them that are stored in a repository or by performing backtracking to find a chain of services that satisfy the objective. The abstract workflow is then concretized, either by finding existing services through a matchmaking algorithm

that matches inputs and outputs and binding them to the workflow, or by recursively calling the abstract workflow generator if no service can be found for an activity.

PAWS [4] is a framework developed by Politecnico di Milano focusing on the adaptation and flexibility of service compositions modeled as business processes. As illustrated in Figure 3.1 , designers create a BPEL process which is then annotated with global and local constraints that usually refer to QoS aspects. The constraints are expressed as Service-Level Agreements (SLAs). For each task in the created process, an advanced service retrieval module attempts to find services that have the required interface (expressed in WSDL or SAWSDL) and do not violate any constraints, by performing SLA negotiation. If no exact interface matches are found, a mediator is used to reconcile the interface discrepancies. For each task, more than one candidate services are selected. When the process is eventually executed by the BPEL engine, one candidate service is invoked for each task. PAWS also supports self-healing, allowing for faulty services to be substituted by other candidate services and at the same time enabling recovery actions to undo the results of the faulty services.

Fujii and Suda [31] [32] propose an architecture for semantics-based, context-aware, dynamic service composition inspired by the component model mentioned in a previous section. They introduce CoSMoS, a semantic abstract model for both service components and users which is the basis for all required representations of their framework. Their composition approach, named SeGSeC, involves receiving a natural language request from the user which is parsed using preexisting natural language analysis technologies into a CoSMoS model instance. This is fed to the workflow synthesis module which creates an executable workflow by discovering and interconnecting components based on the request and the components' functional description. The workflow synthesis module is apparently limited to sequential and parallel composition schemas. Then a semantic matching module ensures that the selected components are semantically equivalent to the user request. If more than one components satisfy both functional and semantic properties for a given task,

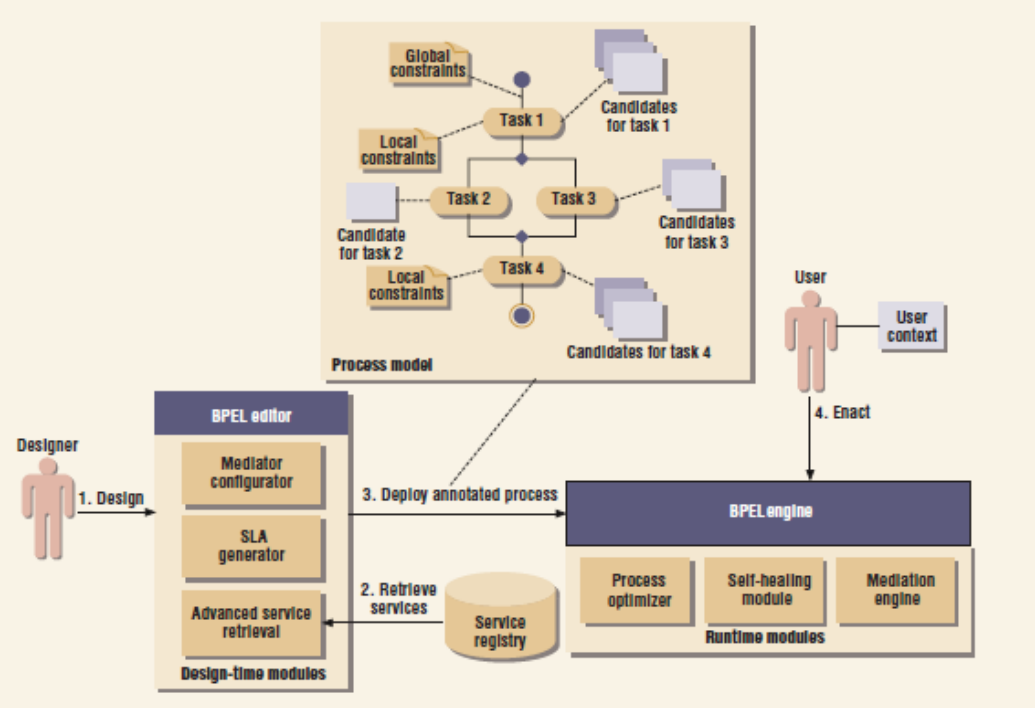

Figure 3.1: The PAWS framework

then context information is exploited, based on rules defined by the user or based on a history of previous decisions, in order to select the most suitable component. The final workflow is then executed and monitored. When a service failure is detected or a change in context is perceived, the workflow can be dynamically modified to adapt to these changes.

In general, we can conclude that while workflow-based composition approaches have evolved from offering only manual and static composition methods to supporting automation and dynamicity, the resulting workflows are limited to simple schemas such as sequential and parallel execution, or in other cases, such as PAWS, automation is only supported during the execution and adaptation of the workflow, while the workflow design process is manual. This deficiency has been addressed by combining workflow-based methods with AI planning techniques. We will examine such works when we analyze planning-based composition approaches.

3.3.2 Model-based Approaches

Model-based or model-driven composition follows the suggestion of system theory to raise the level of abstraction in order to deal with the increasing complexity of systems. Approaches in this category use already explored and well-established models to represent Web services and service composition, thus using a higher description level on top of the traditional service description in WSDL, OWL-S or a similar description framework.

Berardi et al. [11] presented E-Service Composition (ESC), a prototype tool that implements a model-based technique for automated service composition using Finite State Machines (FSMs). They argue that the behavior of a service can be modeled using two schemata, an external schema that specifies its exported (externally-visible) behavior and an internal schema that contains information on which service instance executes each given action that is part of the overall service process. Both schemata are expressed using FSM models. When attempting to synthesize a composition, the external FSM models of the available services and the target service are transformed to modal logic formulas in Deterministic Propositional Dynamic Logic (DPDL). If the resulting set of formulas is satisfiable, then a FSM for the target service exists and is produced. The automatically synthesized FSM can then be converted to a BPEL process and executed in a BPEL engine. In a more recent work [12], Berardi et al. replace FSMs with Transition Systems and use the notion of simulation to virtually compute all possible compositions at once. Neither work however, manages to address the issues of non-determinism and partial observability.

Skogan et al. [80] propose a method that uses UML Activity diagrams to model service compositions. The UML diagrams are then used to generate executable BPEL processes using XSLT transformations. While this work only uses WSDL service descriptions as input to the UML transformation, a follow-up work [37] lifts this limitation by also considering Semantic Web service descriptions in OWL-S and WSMO as well as supporting QoS attributes. This enables dynamicity, since the BPEL pro-

cesses that are generated are abstract and are only bound to concrete services at run-time. Additionally, services are ranked based on QoS properties if more than one service instance satisfies a given task. However, both works do not offer full automation, since the composition model is created manually. Also, the work by Gronmo et al. presents only the methodology behind their model-driven approach without examining whether and how such a methodology can be implemented.

Petri Nets have been used by many researchers in model-based Web service composition [62] [39]. Brogi and Corfini [19] presented Service Aggregation Matchmaking (SAM), a system that is able to discover service compositions, or create them when no suitable composition for a given request is found. The model used to represent Web services is Consume-Produce-Read (CPR) Nets, a simple variant of the standard Condition/Event Petri Nets defined by the authors to properly model the control flow and the data flow of a service. The authors provide a translator from OWL-S descriptions to CPR Nets. The SAM framework can handle both functional and behavioral requests (i.e. requests declaring the desired functional properties of the service or its desired behavior). The functional analysis returns a set of participating services based on a functional request. The behavioral analysis can generate a composite service based on a set of participating services and a behavioral request. The composition of services is based on the composition of CPR Nets which is formally defined by the authors.

Fan et al. [27] propose a notion of synthesized Web services (SWSs) as a uniform formalism to characterize Finite State Automata and transducer abstractions for the modeling of Web services. Using the SWS notion, they formalize several decision problems, such as finding if two services are equivalent or the validity of a service execution and then calculate the complexity of synthesizing a composition of SWSs. A follow-up work [24] also addresses the aggregation problem with regard to SWS, in other words the problem of synthesizing an SWS composition while maximizing or minimizing the output. Both works are fairly recent and in a preliminary stage and

only examine the problems on a theoretical basis with no examination of implementation issues so far.

3.3.3 Mathematics-based Approaches

This category of approaches is created in order to include all approaches that didn't fit in one of the other three categories and at the same time have the common characteristic to be based on mathematical foundations such as various logics, calculi or algebras (hence the term mathematics-based). Of course this in no way implies that approaches not included in this category cannot be based on mathematics, but they were eventually grouped in other categories due to having similarities stronger than a shared mathematical foundation.

The use of process algebraic languages, such as CCS [60] or the pi-calculus [61] has been advocated for Web service composition, since the process specifications included in Semantic Web frameworks such as WSMO or SWSF are formally grounded on process algebras. As illustrated in [58], the pi-calculus can be used to describe and compose Web services. Processes in the pi-calculus can be sequences of other processes, parallel compositions, recursive executions, or choices between operations, thus it is possible to express all basic composition schemas. Receiving and sending information between processes is formalized as input and output artifacts exchanged on channels. One can also introduce types to inputs and outputs. The abstract descriptions that are offered by pi-calculus can be exploited to verify correctness and other properties of compositions.

In [76], the authors propose the use of Linear Logic for automated Web service composition. Linear Logic is an extension of classical logic to model a notion of evolving state by keeping track of resources. The authors propose an automatic translation of OWL-S descriptions to sets of Linear Logic axioms. Then, they use theorem proving to produce the composite service. The target service is expressed as a sequent in Linear Logic and the prover generates all possible compositions which are

then translated to process models using a pi-calculus-inspired process language. It is possible to further translate such models to BPEL workflows. However, the authors themselves acknowledge that using a propositional subset of Linear Logic limits the presentation of Web service properties.

Lecue et al. [49] exploit the fact that OWL-S is based on Description Logics and attempt to use DL reasoning to realize service composition. In previous works, the authors have defined causal links as semantic links between inputs and outputs of services. There are five types of causal links: exact, plugin (when the output is a sub-concept of the input), subsumption (when the output is a super-concept of the input), intersection (if the intersection of the output and the input is satisfiable) and disjoint (when input and output are incompatible). Given a set of services, a causal link matrix (CLM) can be constructed, containing all possible causal links. The authors extend CLM to support non-functional properties. The extended matrix, CLM+ is used in their composition approach, which involves receiving a request, performing service discovery to find a set of candidate services, calculating the CLM+ for these services and attempting to incrementally achieve the output of the request, starting from the input. At each step, the decision is based on the matches inferred from the CLM+ between the output of the last service selected and the inputs of all candidate services. The flow diagram of the composition algorithm is shown in Figure 3.2.

3.3.4 AI Planning Approaches

The final category of automated Web service composition approaches comprises all research efforts that use AI planning techniques in order to generate a composition schema. 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.

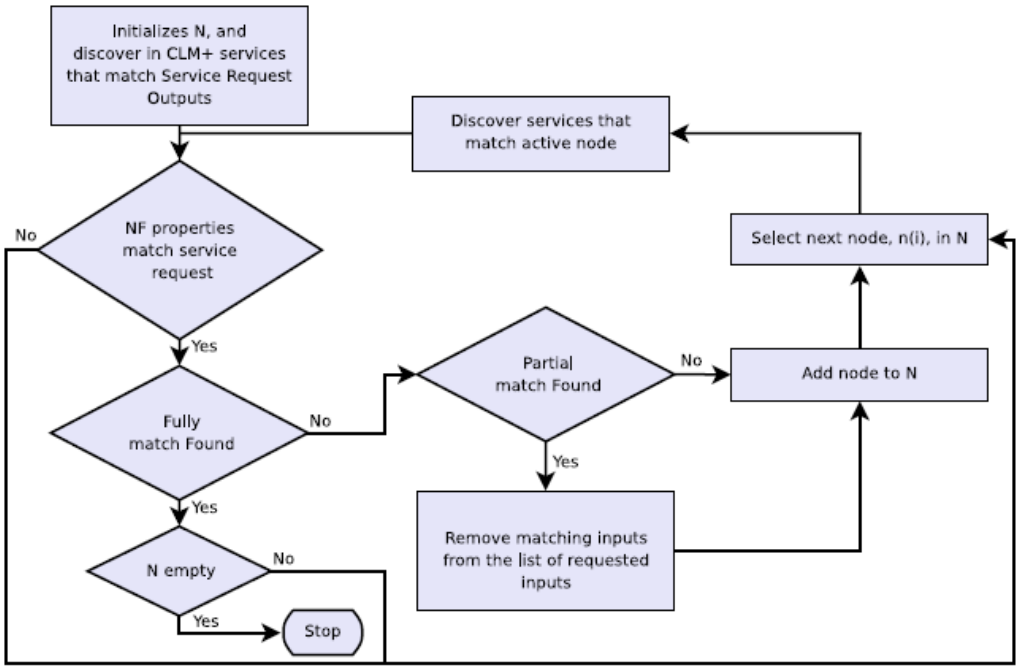

Figure 3.2: Lecue et al. composition algorithm

An AI planning problem (in the classical sense) can be described as a quintuple $\{S, s_0, G, A, \Gamma\}$ [20], 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 describes 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.

Due to the large amount of approaches that fall into the AI Planning category, we will use a further categorization, based on the type of AI planning technique used. The classification is inspired by the works of Chan et al. [21] and Ghallab et al. [35].

Four categories of planning techniques will be examined:

1. **Classical planning:** approaches that involve state-space or plan-space planning

2. **Neoclassical planning:** approaches that employ graph-based planning, propositional satisfiability and constraint satisfaction

3. **Heuristics and Control Strategies:** approaches involving Partial Order Refinement, Hierarchical Task Network planning and planning based on the situation calculus

4. **Other planning techniques:** abductive planning in the event calculus and planning based on model checking

3.3.4.1 Classical planning

Classical planning approaches are based on the definition given at the beginning of this section and involve searching a state-space in order to find a series of state transitions from an initial state to a goal state. Classical planning sometimes involves decomposing a goal to sub-goals and generating a separate plan for each sub-goal, a technique known as plan-space planning.

In [3], the authors attempt to introduce planning in traditional workflow-based service compositions. Given an abstract BPEL flow (with no concrete services bound to the tasks), the goal is to use planning techniques to concretize the workflow. To that end, OWL-S service descriptions are translated to the Planning Domain Definition Language (PDDL) [34], an effort to standardize planning domain and problem description languages (the practice of translating service descriptions to PDDL is common in many approaches in the AI planning category). Then, the PDDL descriptions are fed to IBM's Planner4J planning framework, which contains many planning algorithms, including classical planning ones. At run-time, for each task in the abstract

BPEL flow, a concrete service is requested from the planner module. If a service is not found, the planner attempts to solve a planning problem with the available services as input. This approach is semi-automatic, since the creation of the abstract flow is manual.

Zeng et al. [88] propose the formulation of service composition as a goal-directed planning problem that takes three inputs: a set of domain specific service composition rules (which are manually defined), a description of a business objective or goal, and a description of business assumptions (organizational rules and structures). To that end, they devise an ontology to represent these concepts and a three-step composition schema generation process. In the first step, called Backward-Chaining, the composition rules are exploited trying to create a chain starting from the business objective and moving backwards, until there are no more rules or the initial state is reached. The second step, Forward-Chaining, follows the opposite direction, and attempts to complete the composition schema produced in the first phase by adding services that may be required by the results of some tasks. The final phase, Data-Flow-Inference, adds data flow to the composition schema, since the previous steps only contribute to the control flow aspects of the composition.

3.3.4.2 Neoclassical planning

Neoclassical planning comprises techniques that extend the classical notion of planning. These include graph-based planning, where a plan graph of all possible states and transitions (or a subset of those) is constructed and constraint satisfaction, where the planning problem is translated to a set of constraints and a solver attempts to find a model that satisfies all constraints.

In [75], the authors present a mixed initiative framework for semantic web service discovery and composition, choosing not to automate the complete procedure, but allowing user intervention in key decisions. Their composition engine combines rule-based reasoning on OWL ontologies with Jess and planning functionality based

on the GraphPlan algorithm. GraphPlan is chose because it provides reachability analysis to determine whether a given state can be reached from another state and disjunctive refinement, namely the addition of constraints between steps to resolve possible inconsistencies. Planning is used to propose composition schemas to the user, rather than enforce a decision, which is argued to be the most realistic approach by the authors.

Graph-based planning is also employed by the work of Wu et al. [87] in order to realize service composition. The authors propose their own abstract model for service description which is essentially an extension of SAWSDL to more resemble OWL-S and WSMO. In addition, they model service requests and service compositions with similar semantic artifacts. Then, they extend the GraphPlan algorithm to work with the models they defined. They also add limited support for determinism, by allowing loops only if they are identified beforehand. The final system takes a user request defined using the authors' models and extracts an executable BPEL flow, as shown in the architecture in Figure 3.3.

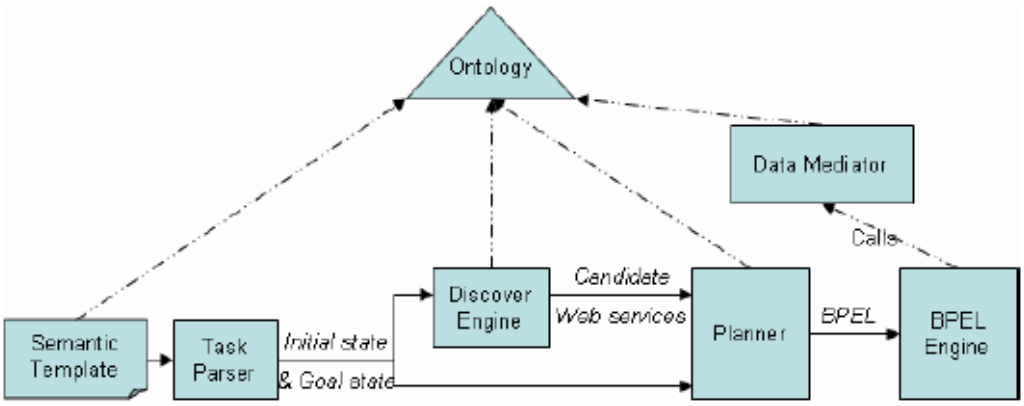

Figure 3.3: Wu et al. composition architecture

The next two works are a combination of neoclassical planning and Hierarchical Task Network planning which is a technique that belongs to the next category,

"Heuristics and Control Strategies". Beauche and Poizat [9] use GraphHTN as a planning technique, which extends the GraphPlan algorithm with task decomposition, which is the defining characteristic of HTN planning. The authors' primary goal is to extend GraphPlan with adaptation capabilities. They introduce original semantic structures to describe the capabilities and data that are involved in a service. These structure enable both vertical adaptation by exploiting semantic relations between capabilities and replacing one service with an equivalent one, but also horizontal adaptation, substituting missing data with semantically equivalent ones. The final generated plan is transformed to a YAWL workflow. YAWL is a workflow language that has been extended to support Web services and can also be transformed to BPEL, if needed.

In [66], the authors combine Constraint Satisfaction (a neoclassical technique) with HTN. The user expresses a service request, which is translated to valid input for an HTN planner (the authors use the most established HTN planner, SHOP2). The HTN planner produces a task flow and a set of constraints which are then fed to a Constraint Satisfaction solver which attempts to find a solution that satisfies all constraints by invoking candidate services with respect to the given task flow. The authors point out, however, that generating a solvable Constraint Satisfaction set may be very difficult for some domains and may require an intervention of an expert to create a broader set.

3.3.4.3 Heuristics and Control Strategies

Techniques in this category use heuristics to determine where to search next or control rules to prune the search space. We will first examine Partial Order Refinement, which is based on plan-space planning and produces incomplete plans where not all tasks are ordered, thus the produced plans may have more than one linearizations. Then we will examine more Hierarchical Task Network planning, which was first observed in the previous category as it was combined with neoclassical techniques.

HTN planning involves decomposing the desired task into sub-tasks and recursively apply decomposition until the resulting sub-tasks can be satisfied. Finally, we will examine works based on Golog, a logic programming language based on the situation calculus.

Peer [70] uses a Partial Order Planner, namely VHPOP, which uses PDDL as input language. The author has defined his own semantic representation of Web services, which is suitably translated to PDDL descriptions. These descriptions are fed to VHPOP, along with a set of links between tasks to avoid, which leads to the production of one or more plans, possibly partially defined. Because there is no guarantee that the plan or plans actually translate to a successful execution, the framework begins by executing the first step. If it fails, then a replanning is performed and a new plan is generated, given the conditions of the failure. If the execution of the first task in the plan is a success, we move on to the second task and so on.

In [44], the authors follow a similar approach, but they employ OWL-S descriptions (instead of creating their own services ontology), which are similarly translated to PDDL descriptions. They use a different planner, however, a hybrid heuristic search planner which combines graph-based planning and HTN. This combines the advantages of the two planning approaches, namely the fact that graph-based planning always find a plan if one exists and the decomposition offered by HTN planning. The framework also includes a replanning component which is able to re-adjust outdated plans during execution time. The architecture of the framework is shown in Figure 3.4

A more recent work by Sohrabi and McIlraith [81] introduces preferences and regulations to HTN planning. The authors extend PDDL to support preferences over how to decompose tasks as well as expressing preferences over the preferred parameterization of a task. They also include regulations (verification-geared constraints such as safety constraints) that should be followed by any generated plan. They introduce a modified HTN planning algorithm, HTNWSC, which takes into account

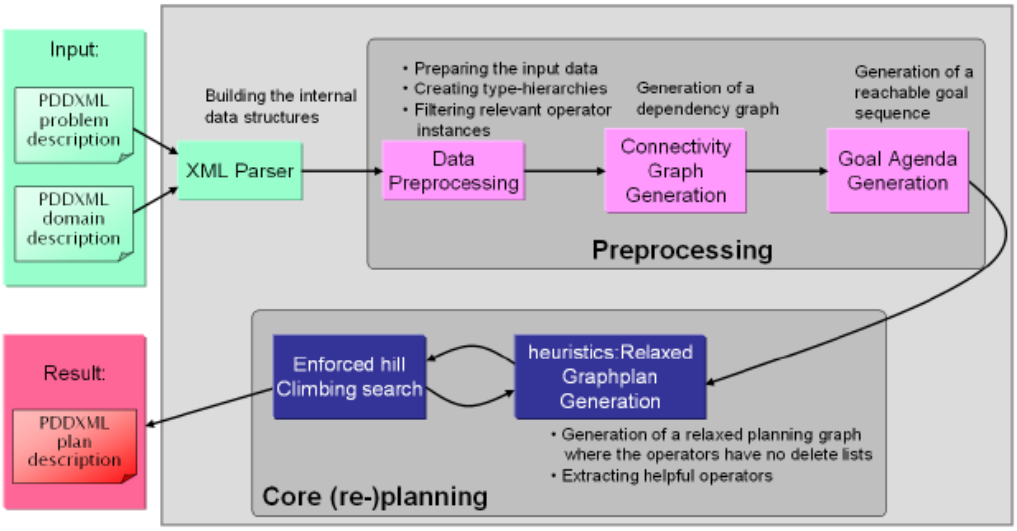

Figure 3.4: Klusch et al. composition architecture

both preferences and regulations in the plan generation procedure.

Before branching to HTN-based solutions, McIlraith and Son [57] first explored planning based on logic programming with Golog. They extended and customized Golog to support personalized constraints and nondeterminism in sequential executions and modified ConGolog, a Golog interpreter to realize these enhancements. This work was developed concurrently with the definition of OWL-S (then DAML-S) and was one of the first to consider Semantic Web services as an input to planners via translation to PDDL. Some years later Phan et al. [71] translate OWL-S directly to the situation calculus and use a ConGolog interpreter, extending it to support information gathering, but ignoring any of the extensions proposed by McIlraith and Son. Also, [50] advances the previous work by Lecue et al. [49] that use DL reasoning techniques and integrates them with an extended Golog interpreter that can compute conditional Web service compositions and can elaborate a strategy for automated branching by means of causal links and laws.

Recently, [82] has proposed the introduction of preferences to planning with Golog, in the spirit of the parallel work of the same authors [81]. Preferences are

expressed using a first-order language defined by the authors and used in a modified version of a Golog interpreter. Evaluation results illustrate the effectiveness of introducing preferences to find optimal compositions.

3.3.4.4 Other planning techniques

[5] uses planning as theorem proving in the event calculus as a means to create service compositions. The planning problem in the event calculus is formulated by the domain knowledge, the event calculus axioms and a goal state. An abductive theorem prover generates the plan, which is a series of events, as well as a set of temporal ordering predicates, giving the partial ordering of events. The authors consider an automatic translation from OWL-S to the event calculus but have not implemented it yet. Also, the fact that there isn't a logic programming language equivalent to Golog for the event calculus makes the expression of processes rather difficult.

The final planning paradigm that we will examine is planning as model checking. Fondazione Bruno Kessler in Trento, Italy has been conducting research in this area for several years. The original works by Traverso and Pistore [85] [72] attempt to exploit planning by model checking in order to deal effectively with nondeterminism, partial observability, and complex goals. OWL-S process models are translated to state transition systems in a similar way that Narayanan and McIlraith [62] translate them to Petri Nets. Goals are expressed using EAGLE, a requirements specification language. State transition systems and goal descriptions are fed to the MBP planner and evaluation showed that while correct plans were produced, the procedure did not scale well, mainly due to the way goals were expressed.

Further efforts by Pistore et al. [73] [74] [14] attempt to address the scalability issues by the definition of an appropriate model for providing a knowledge level description of the component services. In these works, BPEL workflows are used as input instead of OWL-S process models and are translated to their knowledge-level models (essentially a set of propositions). Given a composition goal, they automat-

ically generate its knowledge level representation that declares what the composite service must know and how goal variables and functions must be related with the variables and functions of the component services. This new representation of goals caused an increase in scalability.

These research results have been incorporated in the ASTRO framework [84], allowing it to realize automated service composition. ASTRO takes BPEL processes as input, feeds them into a planner that implements the planning as model checking technique and exports resulting plans in the form of BPEL processes. ASTRO also includes an execution and monitoring component, based on the ActiveBPEL engine.

More recent work from the team of FBK has focused on devising new models for the specification of data flow [53] and control flow [13] requirements. Data flow requirements specify how output messages (messages sent to component services) are obtained from input messages (messages received from component services). Control flow requirements involve termination conditions and transactional issues. Data flow requirements are translated to state transition systems while control flow requirements are translated to the EAGLE language, although previous work had indicated that this caused some scalability issues. [54] shows how this recent research is incorporated in the ASTRO framework.

3.4 Comparison

In this section, we will present comparison tables for all approaches presented in the previous section. The comparison will be based on the requirements that were outlined in the beginning of this Chapter.

Tables 3.1 and 3.2 present a comprehensive comparison that details how each approach manages to meet the automated composition requirements that we defined. A ✓ denotes that the corresponding approach satisfies that particular requirement. A $\sim$ denotes that the corresponding approach only partially satisfies that particular

requirement. No symbol denotes that the corresponding approach does not satisfy that particular requirement (or the authors do not deal with the requirement at all).

Table 3.3 contains a condensed comparison which lists only categories of approaches instead of individual approaches. A ✓ denotes that the particular requirement has been addressed by multiple approaches in the category. A $\sim$ denotes that the particular requirement has been addressed by some approaches in the category. No symbol denotes that no approaches in this category have addressed that particular requirement.

By studying these tables, one can infer a series of research goals that still pose a challenge and could be explored. These will be analyzed in the next Chapter.

	Automation	Dynamicity	Semantic Capabilities	QoS Awareness	Nonde-terminism	Partial Ob-servability	Scalability	Correctness	Domain Inde-pendence	Adaptivity
Majithia et al. [51]	✓	✓	✓							
Ardagna et al. [4]	∼	✓	∼	✓	✓				✓	✓
Fujii et al. [31] [32]	✓	✓	✓				✓		✓	✓
Berardi et al. [11]	✓							✓	✓	
Berardi et al. [12]	✓				✓			✓	✓	
Skogan et al. [80]	∼	✓	✓							
Gronmo et al. [37]	∼	✓	✓	✓						
Brogi et al. [19]	✓		∼				✓	✓	✓	
Fan et al. [27]	✓							✓		
Milanovic et al. [58]	∼						✓	✓	✓	
Rao et al. [76]	✓		✓					✓	✓	
Lecue et al. [49]	✓	✓	✓	✓					✓	

Table 3.1: Comparison of Automated Web Service Composition approaches - part 1

	Automation	Dynamicity	Semantic Capabilities	QoS Awareness	Nonde-terminism	Partial Ob-servability	Scalability	Correctness	Domain Inde-pendence	Adaptivity
Akkiraju et al. [3]	∼	✓	✓		✓			✓	✓	
Zeng et al. [88]	∼		✓	✓				✓		
Rao et al. [75]	∼		✓					✓		
Wu et al. [87]	✓		✓		∼			✓	✓	
Beauche et al. [9]	✓		✓					✓	✓	✓
Paik et al. [66]	✓									
Peer [70]	✓	✓	✓		✓			✓	✓	
Klusch et al. [44]	✓	✓	✓						✓	
Sohrabi et al. [81]	✓		✓				✓	✓	✓	
McIlraith et al. [57]	✓		✓		∼				✓	
Lecue et al. [50]	✓		✓						✓	
Sohrabi et al. [82]	✓		✓				✓		✓	
Aydin et al. [5]	✓		✓						✓	
Traverso and Pistore [85] [72]	✓		✓		✓	✓		✓	✓	
Pistore et al. [73] [74] [14]	✓				✓	✓	∼	✓	✓	

Table 3.2: Comparison of Automated Web Service Composition approaches - part 2

	Automation	Dynamicity	Semantic Capabilities	QoS Awareness	Nonde-terminism	Partial Ob-servability	Scalability	Correctness	Domain Inde-pendence	Adaptivity
Workflow-based	✓	✓	✓	≈					✓	✓
Model-based	✓	≈						✓	✓	
Mathematics-based	✓	≈	✓	≈				✓	✓	
Classical planning	≈	≈	✓	≈				✓	≈	
Neoclassical	✓		✓					✓	≈	
Heuristics	✓		✓				≈	≈	✓	
Model checking	✓		≈		✓	✓	≈	✓	✓	

Table 3.3: Comparison of Automated Web Service Composition approaches - by category

Chapter 4

Research Challenges

In this chapter, we will present and analyze a set of research challenges that were identified while preparing this document. The first section contains research challenges that deal with Web service description, especially with the formal specification of Web services and service compositions and is based on the analysis carried out in Chapter 2. The second section includes research challenges on the topic of automated Web service composition, based on the comprehensive state of the art that was presented in Chapter 3.

4.1 Formal Specification of Web services and service compositions

From the analysis in Chapter 2 it was gathered that there are still open issues concerning the description and formal specification of Web services and service compositions. These issues concern the ramification problem, the qualification problem and the problem of automatically deriving specifications for Web service compositions. The rest of this section is devoted to describing the research challenges related to these issues.

All challenges presented in this section contribute to the same high-level goal: to

provide complete, formal, rich, semantic-aware specifications for Web services. The existence of such specifications is of crucial importance for both service providers and service consumers. Service providers will be able to provide complete specifications of what they are offering, which would enable them to more effectively advertise their service products to potential clients. Service consumers, on the other hand, will be able to be informed of the exact way in which a service is expected to performed and the produced results, which will allow them to make informed choices and select the service that is the most suitable match for their requirements. Given the fact that formal specifications can be automatically processed in an effective way, such matchmaking between what is offered by providers and what is required by consumers can be performed in an automated way, speeding up the process and at the same time lowering the costs. Thus, formal specifications are one step towards the embracing of Service-Oriented Architectures by the industry.

4.1.1 Addressing the ramification problem

In Chapter 2 the ramification problem was defined as the problem of adequately representing and inferring information about the knock-on and indirect effects (also known as ramifications) that might accompany the direct effects of an action or event. The ramification problem has not yet been examined in correlation with Web services and the effects it may have on their description and formal specification.

Let's take a look at a simple example. Suppose that we want to create a formal specification for a service that withdraws a specific amount of money from a bank account associated with a credit card. This specification should include the required inputs and preconditions for the service to execute correctly, as well as the expected outputs from a successful execution and the postconditions that must be held. Let's focus on the preconditions and postconditions of the money withdrawal service. A precondition that should hold before the service begins execution is that the balance of the client's account should at least be equal to the amount of money about to

be withdrawn. Moving on, let's say that the bank our service works with imposes a limit for the amount of money an account holder can withdraw in a day. If the limit has been reached, the account is banned for the rest of the day. Based on that, we should add another precondition that makes sure that the account is not banned when the service is executed. As far as postconditions are concerned, a successful execution means that the new balance is equal to the old one minus the amount of money withdrawn. Also, the total money withdrawn for the day should be increased by the same amount. Table 4.1 shows a possible first-order logic encoding of the preconditions and postconditions we expressed here in natural language. The unprimed/primed notation is used to refer to predicates and functions calculated immediately before and after the execution of the service, respectively. We use the variable A to denote the amount of money that will be withdrawn.

Preconditions
$balance(account) \geq A \wedge$
$Valid(creditCard, account)$
Postconditions
$balance'(account) = balance(account) - A \wedge$
$withdrawalTotal'(account) = withdrawalTotal(account) + A$

Table 4.1: Pre/Postconditions for the Money Withdrawal Service

At first glance, one would assume that this is a complete specification that can be used for all intents and purposes. However, the definitions of the frame and ramification problems tell us the opposite. If we employ the solution to the frame problem presented in [7], then the specification would be transformed as shown in 4.2. Essentially, we're expressing that the *balance* and *withdrawalTotal* functions change only when the MoneyWithdrawal service has executed given the particular account as input as well as the amount of money to withdraw. The predicate *Valid* doesn't

change in any case.

Preconditions
$balance(account) \geq A \wedge$
$Valid(creditCard, account)$
Postconditions
$balance'(account) = balance(account) - A \wedge$
$withdrawalTotal'(account) = withdrawalTotal(account) + A$
$\forall \alpha, \forall x[(balance(x) \neq balance'(x)) \wedge Occur(\alpha) \Rightarrow \alpha = MoneyWithdrawal(x, A)]$
$\forall \alpha, \forall x[(withdrawalTotal(x) \neq withdrawalTotal'(x)) \wedge Occur(\alpha) \Rightarrow \alpha = MoneyWithdrawal(x, A)]$
$\forall \alpha, \forall x, y[Valid(x, y) \not\equiv Valid'(x, y) \wedge Occur(\alpha) \Rightarrow false$

Table 4.2: Pre/Postconditions w/ Change Axioms for the Money Withdrawal Service

However, this specification, while solving the frame problem has precluded any indirect effects that may result because of the effects of the service execution. For example, suppose that there is a limit to the amount of money that the client can withdraw during a day and if it is surpassed the credit card associated with the account is temporarily made invalid. Thus, a ramification of the increase in the total amount withdrawn from the account due to the execution of the service is that if the daily limit is surpassed, the credit card must be temporarily made invalid. However, this goes against the change axiom that was included to address the frame problem, which states that the *Valid* predicate remains unchanged after the successful execution of the service. One can think of many other ramifications, so it is imperative to find a solution of expressing such indirect effects while at the same time being consistent with a solution to the frame problem.

Ramifications must be included in a service specification since the service consumer must be aware of all (direct and indirect) effects of the service execution. This is particularly important in service compositions, as the lack of knowledge of an indi-

rect effect may lead to the assumption that a composition is valid and correct while that particular effect may contradict a precondition of another participating service, leading to an inconsistent composite service. For instance, in our running example, if we are unaware of the ramifications of the service, then we can assume that we can sequentially compose two executions of the Money Withdrawal service and result in a composition that is always correct and executable. This is not the case, however, if the first execution leads to the credit card being rendered invalid because the daily limit has been reached.

A first step towards the direction of addressing the ramification problem in Web services is to identify the cases in which the problem makes its appearance, to examine the effects it may have and to assess the severity of them. Then, it would be interesting to examine whether previous solutions for the ramification problem that have been presented in other research areas could be adapted to work in the case of Web services. Such solutions include but are not limited to McIlraith's solution [56] in the situation calculus and Shanahan's solution [79] in the event calculus.

4.1.2 Addressing the qualification problem

The second challenge is similar in spirit to the first one. It involves addressing the qualification problem in Web service specifications. From its definition in Chapter 2 one can derive many correlations to the case of Web services. First of all, updating qualifications when new statements become part of our knowledge is something very common in the ever-changing world of Web services. This facet of the qualification problem is closely related to the adaptation qualities of a Web service framework. Hence, a solution to the qualification problem for Web service specifications would involve adapting such specifications whenever new knowledge is acquired, maybe due to new services becoming part of the repository at hand or due to the service provider modifying a specification.

Moreover, the qualification problem becomes very interesting in the case of com-

posite service specifications. It is interesting to explore how the introduction of a new service in a composition schema (either as a replacement to an existing one that failed or by modifying the workflow of the schema itself) affects the qualifications of the services already participating in the composition. It is even more challenging to address such questions not at design-time, but at run-time, when the composite service is being executed and monitored.

Let's return to our running example to see how this applies to the case of Web service specifications. Given the initial specification in Table 4.1, suppose that now we are told that there is an initial period after the initial activation of a credit card when it cannot yet be used for purchases (i.e. no money can be withdrawn), in other words a card cannot be valid and at the same time be in a provisional state. This new knowledge (that can, for example, be expressed in first order logic as $\neg(Valid(creditCard, account) \wedge Provisional(creditCard, account)))$ must be assimilated in the existing service specifications, leading to the derivation of new qualifications, if necessary, while at the same time taking care so that the resulting is not logically inconsistent. If the new knowledge is derived from a new participating service in a composition schema and leads to inconsistencies, this means that this addition can't be approved and another service must be found to be included in the composition.

A similar strategy to the one mentioned in the previous subsection could be followed while attempting to address the qualification problem. McIlraith's work [56] also involves the axiomatic formalization of a solution to the qualification problem. Kvarnstorm and Doherty [48] offer a solution based on their so-called fluent dependency constraints, while Thielscher [83] examines the problem from the viewpoint of intelligent agents in open environments.

4.1.3 Automatic derivation of composite specifications

The final challenge of this section serves as a bridge between service description and service composition. While service description frameworks attempt to describe service compositions using a variety of composition models ranging from orchestrations to choreographies to Finite State Machines, no framework (to the best of our knowledge) attempts to handle the problem of automatically producing specifications for a composite service, based on the specifications of the participating services) The same is true for automated Web service composition approaches: while each and every one of them offers a way of automatically or semi-automatically producing the composition schema, the control flow and data flow of the composite service, none attempt to derive a complete specification of the inputs, outputs, preconditions and effects (IOPEs) that should be provided to a service consumer.

The composite specification is directly linked to the composition schema and the workflow patterns that describe the control flow and data flow between participating services. Automatically deriving the IOPEs of a composite service should be based on the way the services are orchestrated together. The naive way would be to expose the preconditions of the first services executed as the preconditions of the composite service, and the postconditions of the last services executed as the postconditions of the composite service, leaving everything else as internal knowledge that shouldn't be delivered to the requester. This may be true in some cases, but it can't be used as a general practice. The following simple example illustrates that.

We want to compose service I and service A. Service I is the identity service, which outputs whatever is passed to it as input. Both the precondition and the postcondition of the identity service are true. Service A is a generic service, with precondition PreA and postcondition PostA. If we compose I ; A, according to the naive solution, the precondition of the composition is always true and PreA is not exposed in the composite specification. Similarly, if we compose A ; I, the postcondition of the composition is always true and PostA is not exposed in the composite

specification. However, common sense dictates that in the first case the precondition of the composite service should be PreA and in the second case the postcondition should be PostA. Hence, the naive solution doesn't work in the general case.

On the other hand, a trivial solution would be to expose the full set of preconditions and postconditions of all participating services. This, however, wouldn't scale well to large compositions consisting of many complex services. Thus, a middle-ground solution should be explored: composite specifications have to reveal some part of the specifications of some of the internal services. The challenge is to decide what should be included and what should be left out. Similar issues have been raised in programming language specifications and as evidenced by the work of Gries [36], Hoare logic, weakest precondition and the Craig-Lyndon interpolation theorem should give insight to the issue of the automatic derivation of composite specification.

The derivation of composite specifications can be proven very useful since such specifications can be used as complete formal descriptions of what the composite services are offering to consumers in terms of functionality and under what circumstances such functionality can be achieved. This will promote and facilitate the reusability of composite services. Such specifications would also be of great assistance, when one attempts to deduce whether a set of services can actually be composed in a meaningful way. If inconsistencies are detected during the process of automatically creating the composite specifications, then the set of services is not composable and a replacement should be found for the service or services that cause the inconsistencies.

4.2 Automated Web Service Composition

Chapter 3 ended with a comprehensive comparison of the most important approaches that attempt to handle the problem of automated Web service composition. The comparison was presented in a series of tables that prove very useful when it comes to identifying research challenges and open issues to the general problem at hand. In

this section, we will briefly summarize the research challenges that were derived from the state of the art survey and comparison we conducted. Notice that we exclude any discussion about the requirement of adaptivity. We consider service adaptation to be a research area with many interesting directions that are nevertheless out of the scope of this survey.

4.2.1 QoS-Awareness in AI Planning techniques

When examining Table 3.3 with regard to QoS-Awareness, one can deduce that while it has been explored (adequately or not) in several categories of approaches, it has been largely ignored in the case of AI planning. The only AI planning effort in the state of the art survey that deals with QoS is the work of Zeng et al. [88]. This work however, as it can be seen in Table 3.2 fails to meet very important requirements such as support for non-determinism and partial observability as well as scalability and domain independence.

Apart from this discussion, introducing QoS-awareness in AI planning techniques involves making at least three fundamental decisions. The first decision is to choose a planning technique which will then be adapted to be QoS-aware. The comparison tables can again be useful to making that decision as we can choose a technique that has shown promising results in meeting the rest of the requirements that we have set for automated Web service composition. Planning as model checking is one such technique, as it is the only one that addresses the issues of nondeterminism and partial observability.

A second decision involves the QoS model that will be employed. Rather than creating one from scratch, a preexisting one could be used, as long as it satisfies some important requirements such as being expressive, formal and fine-grained, as well as being based on Semantic Web concepts. A choice that could be explored is integrating the work of Kritikos and Plexousakis [46] which extends OWL for QoS-based Web Service Description and Discovery. It would be interesting to examine how OWL-Q

can be used to make a composition framework based on AI planning QoS-aware.

Another important aspect is to decide on which phases of the composition process will the QoS characteristics be applied and the effects of that application. For example, introducing non-functional characteristics in AI planning techniques may assist in speeding up the plan generation by limiting the search space when excluding services that don't meet preset QoS thresholds. However, it may also do the exact opposite if we include QoS criteria in the plan goal, since it will make the goal more complex and harder to satisfy. It is apparent that introducing QoS-awareness is a rather challenging task.

4.2.2 Dynamicity in AI Planning techniques

As is the case with QoS-awareness, dynamicity is another requirement that hasn't been adequately explored in AI planning techniques. The vast majority of them produce a static composition schema (the generated plan) without exploring the case of run-time composition where composition schemas are abstract and are only made concrete at run-time. The only exceptions are the works of Peer [70] and Klusch et al. [44] which have some notion of dynamicity through the use of replanning. Peer uses it to recursively generate plans that are incrementally closer to the required goal. This introduces some level of dynamicity but only at design time.

On the other hand, Klusch et al. use replanning whenever an agent detects that a service's preconditions have been violated during the execution of a plan. The replanning module is informed of the position of the error at the plan and tries to fix the problem by searching for an alternative path in the connectivity graph from that position onwards. To reduce the search space, unnecessary services can be blocked to avoid a complete preprocessing phase.

Klusch et al.'s effort is a step in the right direction, however, as is usually the case, it lacks many other important features, such as support for nondeterminism and partial observability, QoS-awareness, scalability and any proof of correctness. It

should be challenging to explore how dynamicity at run-time via replanning can be applied to planning techniques that support most of these features, such as planning as model checking. Thus, the advantages of both approaches could be combined to result in a composition framework that achieves most of the composition requirements we have set.

4.2.3 Scalability

A major drawback in most (if not all) automated service composition techniques is the lack of scalability. This has been one of the reasons (maybe the main one) behind the fact that the industry has yet to adopt any of the approaches presented by research communities, choosing to adopt manual composition techniques based on BPEL which has proven to be effective and scalable. It is a fact that most of the techniques presented in this survey work well in theory but there is no substantial evidence that they work in real-life composition problems. Although many approaches explore scalability issues and present examples that are borrowed from real-life scenarios, true scalability still remains an open issue, as it is usually artificially obtained by severely limiting the scope of the composition problem.

Choosing a subset of the problem in order to achieve scalability is not considered by definition wrong. In fact, it is reasonable to explore what is the cause of the lack of scalability and try to remove that cause. However, this should be done only after evaluating the significance of limiting the approach in that way. Achieving scalability but at the same time crippling the overall applicability of the approach is obviously unreasonable since the advantage offered by the former is canceled by the effects of the latter. The tradeoff between scalability and applicability is as fundamental as it is challenging.

Realizing scalability, dynamicity and QoS-awareness in automated Web service composition (the three challenges that were presented in this section) may be the necessary incentive for the industry to consider adopting such techniques for the

composition of Web services. The advantages offered by an approach that satisfies all the requirements that we defined in the beginning of Chapter 3 far outweigh the disadvantages that one can think of such as the lack of expertise on SOA technologies in the industry and the extra training that may be required for traditional service experts. Of course, the eventual adoption of such an approach by the industry also depends on the availability of semantically-enabled Web services and the existence of effective discovery techniques for such services which are the main concerns that were identified in a recent industry survey performed by the S-Cube consortium [33].

Bibliography

[1] V. Agarwal, G. Chafle, S. Mittal, and B. Srivastava. Understanding approaches for web service composition and execution. In R. K. Shyamasundar, editor, *Bangalore Compute Conf.*, page 1. ACM, 2008.

[2] R. Akkiraju, J. Farrell, J.Miller, M. Nagarajan, M. Schmidt, A. Sheth, and K. Verma. Web Service Semantics - WSDL-S, A joint UGA-IBM Technical Note, version 1.0. Technical report, UGA-IBM, April 2005.

[3] R. Akkiraju, K. Verma, R. Goodwin, P. Doshi, and J. Lee. Executing Abstract Web Process Flows. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2004.

[4] D. Ardagna, M. Comuzzi, E. Mussi, B. Pernici, and P. Plebani. Paws: A framework for executing adaptive web-service processes. *IEEE Software*, 24:39–46, 2007.

[5] O. Aydin, N. K. Cicekli, and I. Cicekli. Automated web services composition with the event calculus. In A. Artikis, G. M. P. O'Hare, K. Stathis, and G. A. Vouros, editors, *ESAW*, volume 4995 of *Lecture Notes in Computer Science*, pages 142–157. Springer, 2007.

[6] 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, pages 289–298, New York, NY, USA, November 2004.

[7] G. Baryannis and D. Plexousakis. The frame problem in web service specifications. In *PESOS '09: Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems*, pages 9–12, Washington, DC, USA, 2009. IEEE Computer Society.

[8] 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 Framework (SWSF) Overview. World Wide Web Consortium, Member Submission SUBM-SWSF-20050909, September 2005.

[9] S. Beauche and P. Poizat. Automated service composition with adaptive planning. In A. Bouguettaya, I. Kruğer, and T. Margaria, editors, *ICSOC*, volume 5364 of *Lecture Notes in Computer Science*, pages 530–537, 2008.

[10] K. Bennett, P. Layzell, D. Budgen, P. Brereton, L. Macaulay, and M. Munro. Service-based software: the future for flexible software. In *Seventh Asia-Pacific Software Engineering Coference (APSEC2000)*, pages 214–221, 2000.

[11] D. Berardi, D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Mecella. Automatic service composition based on behavioral descriptions. *Int. J. Cooperative Inf. Syst.*, 14(4):333–376, 2005.

[12] D. Berardi, F. Cheikh, G. D. Giacomo, and F. Patrizi. Automatic service composition via simulation. *Int. J. Found. Comput. Sci.*, 19(2):429–451, 2008.

[13] P. Bertoli, R. Kazhamiakin, M. Paolucci, M. Pistore, H. Raik, and M. Wagner. Control flow requirements for automated service composition. In *ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services*, pages 17–24, Washington, DC, USA, 2009. IEEE Computer Society.

[14] P. Bertoli, M. Pistore, and P. Traverso. Automated composition of web services via planning in asynchronous domains. *Artif. Intell.*, 174(3-4):316–361, 2010.

[15] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web Services Architecture. W3C Note NOTE-ws-arch-20040211, World Wide Web Consortium, February 2004.

[16] A. Borgida, J. Mylopoulos, and R. Reiter. On the Frame Problem in Procedure Specifications. *Software Engineering, IEEE Transactions on*, 21(10):785–798, 1995.

[17] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (SOAP) 1.1. W3c note, World Wide Web Consortium, May 2000.

[18] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, and J. Cowan. Extensible Markup Language (XML) 1.1 (Second Edition). World Wide Web Consortium, Recommendation REC-xml11-20060816, August 2006.

[19] A. Brogi and S. Corfini. Ontology- and behavior-aware discovery of web service compositions. *Int. J. Cooperative Inf. Syst.*, 17(3):319–347, 2008.

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

[21] K. S. M. Chan, J. Bishop, and L. Baresi. Survey and comparison of planning techniques for web services composition. Technical report, Department of Computer Science, Unviersity of Pretoria, 0002 Pretoria, South Africa, 2006.

[22] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. World Wide Web Consortium, Recommendation REC-wsdl20-20070626, June 2007.

[23] O. S. Collaboration. Service Component Architecture. Internet (http://www.osoa.org/display/Main/Service+Component+Architecture+Home), 2007.

[24] T. Deng, W. Fan, and L. Libkin. On the aggregation problem for synthesized web services. In *AICP*, pages 242–251. ACM, 2010.

[25] S. Dustdar and W. Schreiner. A survey on web services composition. *IJWGS*, 1(1):1–30, 2005.

[26] J. A. Estefan, K. Laskey, F. G. McCabe, and D. Thornton. Reference Architecture for Service Oriented Architecture Version 1.0. Online PDF, 2008.

[27] W. Fan, F. Geerts, W. Gelade, F. Neven, and A. Poggi. Complexity and composition of synthesized web services. In M. Lenzerini and D. Lembo, editors, *PODS*, pages 231–240. ACM, 2008.

[28] J. Farrell and H. Lausen. Semantic Annotations for WSDL and XML Schema. World Wide Web Consortium, Recommendation REC-sawsdl-20070828, August 2007.

[29] R. T. Fielding and R. N. Taylor. Principled design of the modern Web architecture. *ACM Transactions on Internet Technologies*, 2(2):115–150, 2002.

[30] O. for the Advancement of Structured Information Standards (OASIS). Web Services Composite Application Framework. Internet (http://www.oasis-open.org/committees/), 2006.

[31] K. Fujii and T. Suda. Semantics-based dynamic web service composition. *Int. J. Cooperative Inf. Syst.*, 15(3):293–324, 2006.

[32] K. Fujii and T. Suda. Semantics-based context-aware dynamic service composition. *TAAS*, 4(2), 2009.

[33] A. Gehlert and E. D. Nitto. Ia-2.2.1: Identification of potential industrial collaborators. Technical report, S-Cube Network of Excellence, September 2008.

[34] M. Ghallab, C. K. Isi, S. Penberthy, D. E. Smith, Y. Sun, and D. Weld. Pddl - the planning domain definition language. Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[35] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: theory and practice*. Morgan Kaufmann Publishers, 2004.

[36] D. Gries. *The Science of Programming*. Springer, 1981.

[37] R. Gronmo and M. C. Jaeger. Model-driven semantic web service composition. In *APSEC '05: Proceedings of the 12th Asia-Pacific Software Engineering Conference*, pages 79–86, Washington, DC, USA, 2005. IEEE Computer Society.

[38] M. Grüninger and C. Menzel. The Process Specification Language (PSL) Theory and Applications. *AI Magazine*, 24(3):63–74, 2003.

[39] R. Hamadi and B. Benatallah. A petri net-based model for web service composition. In *ADC '03: Proceedings of the 14th Australasian database conference*, pages 191–200, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[40] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosofand, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission, May 2004. Available at: http://www.w3.org/Submission/SWRL, last access on Dez 2008.

[41] R. Hull and J. Su. Tools for composite web services: a short overview. *SIGMOD Record*, 34(2):86–95, 2005.

[42] John McCarthy and Patrick J. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

[43] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto. Web services choreography description language version 1.0. World Wide Web Consortium, Candidate Recommendation CR-ws-cdl-10-20051109, November 2005.

[44] M. Klusch and A. Gerber. Semantic web service composition planning with owls-xplan. In *In Proceedings of the 1st Int. AAAI Fall Symposium on Agents and the Semantic Web*, pages 55–62, 2005.

[45] J. Koehler and B. Srivastava. Web service composition: Current solutions and open problems. In *Proceedings of the ICAPS 2003 Workshop on Planning for Web Services*, pages 28–35, 2003.

[46] K. Kritikos and D. Plexousakis. Requirements for qos-based web service description and discovery. *IEEE T. Services Computing*, 2(4):320–337, 2009.

[47] U. Küster, M. Stern, and B. König-Ries. A classification of issues and approaches in automatic service composition. In *Intl. Workshop WESC 05*, 2005.

[48] J. Kvarnstroḿ and P. Doherty. Tackling the qualification problem using fluent dependency constraints. *Computational Intelligence*, 16(2):169–209, 2000.

[49] F. Lécué, E. M. G. da Silva, and L. F. Pires. A framework for dynamic web services composition. In *2nd ECOWS Workshop on Emerging Web Services Technology (WEWST07), Halle, Germany*, Germany, November 2007. CEUR Workshop Proceedings.

[50] F. Lécué, A. Léger, and A. Delteil. DL Reasoning and AI Planning for Web Service Composition. In *Web Intelligence*, pages 445–453. IEEE, 2008.

[51] S. Majithia, D. W. Walker, and W. A. Gray. A framework for automated service composition in service-oriented architectures. In C. Bussler, J. Davies, D. Fensel, and R. Studer, editors, *ESWS*, volume 3053 of *Lecture Notes in Computer Science*, pages 269–283. Springer, 2004.

[52] A. Marconi and M. Pistore. Synthesis and composition of web services. In M. Bernardo, L. Padovani, and G. Zavattaro, editors, *SFM*, volume 5569 of *Lecture Notes in Computer Science*, pages 89–157. Springer, 2009.

[53] A. Marconi, M. Pistore, and P. Traverso. Specifying data-flow requirements for the automated composition of web services. In *SEFM*, pages 147–156, 2006.

[54] A. Marconi, M. Pistore, and P. Traverso. Automated composition of web services: the astro approach. *IEEE Data Eng. Bull.*, 31(3):23–26, 2008.

[55] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. OWL-S: Semantic Markup for Web Services. Internet (http://www.daml.org/services/owl-s/1.0/owl-s.pdf), 2004.

[56] S. A. McIlraith. Integrating actions and state constraints: A closed-form solution to the ramification problem (sometimes). *Artif. Intell.*, 116(1-2):87–121, 2000.

[57] S. A. McIlraith and T. C. Son. Adapting golog for composition of semantic web services. In D. Fensel, F. Giunchiglia, D. L. McGuinness, and M.-A. Williams, editors, *KR*, pages 482–496. Morgan Kaufmann, 2002.

[58] N. Milanovic and M. Malek. Current solutions for web service composition. *IEEE Internet Computing*, 8(6):51–59, 2004.

[59] R. Miller. Three problems in logic-based knowledge representation. *ASLIB Proceedings: New information perspectives*, 2006.

[60] R. Milner. *Communication and concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[61] R. Milner. *Communicating and mobile systems: the pi-calculus*. Cambridge University Press, fifth edition, 2004.

[62] S. Narayanan and S. A. McIlraith. Simulation, Verification and Automated Composition of Web services. In *WWW*, pages 77–88, 2002.

[63] E. Newcomer and G. Lomow. *Understanding SOA with Web Services*, chapter 1. Addison Wesley Professional, December 2004.

[64] OASIS. Web services business process execution language version 2.0. Padra o, April 2007.

[65] Object Management Group, Framingham, Massachusetts. *The Common Object Request Broker: Architecture and Specification — Version 2.6*, December 2001.

[66] I. Paik and D. Maruyama. Automatic web services composition using combining htn and csp. In *CIT*, pages 206–211. IEEE Computer Society, 2007.

[67] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *Computer*, 40(11):38–45, 2007.

[68] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *IEEE Computer*, 40(11):38–45, 2007.

[69] P. F. Patel-Schneider. A Proposal for a SWRL Extension to First-Order Logic. Internet (http://www.daml.org/2004/11/fol/proposal), 2004.

[70] J. Peer. A pop-based replanning agent for automatic web service composition. In A. Go'mez-Pe'rez and J. Euzenat, editors, *ESWC*, volume 3532 of *Lecture Notes in Computer Science*, pages 47–61. Springer, 2005.

[71] M. Phan and F. Hattori. Automatic web service composition using congolog. In *ICDCS Workshops*, page 17. IEEE Computer Society, 2006.

[72] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, and P. Traverso. Planning and monitoring web service composition. In *AIMSA*, pages 106–115, 2004.

[73] M. Pistore, A. Marconi, P. Bertoli, and P. Traverso. Automated composition of web services by planning at the knowledge level. In *IJCAI*, pages 1252–1259, 2005.

[74] M. Pistore, P. Traverso, and P. Bertoli. Automated composition of web services by planning in asynchronous domains. In *ICAPS*, pages 2–11, 2005.

[75] J. Rao, D. Dimitrov, P. Hofmann, and N. M. Sadeh. A mixed initiative approach to semantic web service discovery and composition: Sap's guided procedures framework. In *ICWS*, pages 401–410. IEEE Computer Society, 2006.

[76] J. Rao, P. Kuïgas, and M. Matskin. Logic-based web services composition: From service description to process model. In *ICWS*, pages 446–453. IEEE Computer Society, 2004.

[77] J. Rao and X. Su. A survey of automated web service composition methods. In J. Cardoso and A. P. Sheth, editors, *SWSWPC*, volume 3387 of *Lecture Notes in Computer Science*, pages 43–54. Springer, 2004.

[78] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web Service Modeling Ontology. *Applied Ontology*, 1:77–106, 2005.

[79] M. Shanahan. The ramification problem in the event calculus. In T. Dean, editor, *IJCAI*, pages 140–146. Morgan Kaufmann, 1999.

[80] D. Skogan, R. Gronmo, and I. Solheim. Web service composition in uml. *Enterprise Distributed Object Computing Conference, IEEE International*, 0:47–57, 2004.

[81] S. Sohrabi and S. A. McIlraith. Optimizing web service composition while enforcing regulations. In *ISWC 2009: Proceedings of the 8th International Semantic Web Conference, Chantilly, VA, USA*, pages 601–617, 2009.

[82] S. Sohrabi, N. Prokoshyna, and S. A. McIlraith. Web service composition via the customization of golog programs with user preferences. In A. Borgida, V. K. Chaudhri, P. Giorgini, and E. S. K. Yu, editors, *Conceptual Modeling: Foundations and Applications*, volume 5600 of *Lecture Notes in Computer Science*, pages 319–334. Springer, 2009.

[83] M. Thielscher. The qualification problem: A solution to the problem of anomalous models. *Artif. Intell.*, 131(1-2):1–37, 2001.

[84] 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 *ICSOC*, pages 495–501, 2005.

[85] P. Traverso and M. Pistore. Automated composition of semantic web services into executable processes. In *International Semantic Web Conference*, pages 380–394, 2004.

[86] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, 2009.

[87] Z. Wu, A. Ranabahu, K. Gomadam, A. P. Sheth, and J. A. Miller. Automatic composition of semantic web services using process and data mediation. Technical report, Kno.e.sis Center, Wright State University, 2 2007.

[88] L. Zeng, A. H. H. Ngu, B. Benatallah, R. M. Podorozhny, and H. Lei. Dynamic composition and optimization of web services. *Distributed and Parallel Databases*, 24(1-3):45–72, 2008.