Teknillinen korkeakoulu Tietotekniikan osasto
Tietojenkäsittelyopin laboratorio B
Helsinki University of Technology Department of Computer Science and Engineering
Laboratory of Information Processing Science B
Espoo 2008 TKK-TKO-B158

SEMANTIC WEB SERVICES — A SURVEY

Seppo Törmä, Jukka Villstedt, Ville Lehtinen, Ian Oliver, Vesa Luukkala

Teknillinen korkeakoulu
Tekniska högskolan
Helsinki University of Technology

Helsinki University of Technology
Laboratory of Software Technology
P.O. Box 5400
FIN-02015 HUT
URL: http://www.cs.hut.fi/english.html

Copyright ©2008 Seppo Törmä, Jukka Villstedt, Ville Lehtinen, Ian Oliver, and Vesa Luukkala

You may download, display, and print this publication for your own personal use. Commercial use is prohibited.

ISBN 978-951-22-9342-1 (PDF)
ISSN 1239-6893
URL: http://www.cs.hut.fi/Publications/Reports/B158.pdf

This publication is available only as a PDF file. It is designed to be printed on A4 paper.

Abstract

Authors: Seppo Törmä, Jukka Villstedt, Ville Lehtinen, Ian Oliver, Vesa Luukkala

Title: Semantic Web Services — A Survey

Publisher: Helsinki University of Technology, Laboratory of Software Technology

Report code: TKK-TKO-B158 **Number of pages**: 66

Date: 31.3.2008

Semantic Web Services is a technology to enable dynamic, execution-time discovery, composition, and invocation of Web Services. It is built on top of the Web Services technology that provides means for software development-time discovery, manual composition, and statically coded invocation of predetermined services. That underlying technology is extended with rich semantic representations developed in the area of the Semantic Web and with capabilities for automatic reasoning developed in the field of artificial intelligence.

This survey attempts to give an overview of the underlying concepts and technologies. An outline of the elements of the solutions are presented, including service description languages, shared conceptualizations (ontologies) of services, and descriptions of the functional properties of services through their inputs, outputs, preconditions, and effects. The structure of three frameworks proposed for Semantic Web Services (OWL-S, SWSF, and WSMO) as well as the reasoning problems and technologies – terminological reasoning, planning, rule programming, mediation – are described.

Keywords: Semantic Web Services, Semantic Web, Web Services, Service modelling, AI Planning, Description logics, Mediation, Automatic reasoning

Contents

Chapter 1

Introduction

Semantic Web Services is an emerging technological concept with a vision that services available in the Web could be utilized efficiently without human intervention. It combines technologies from the fields of *Web Services*, the *Semantic Web*, and *automatic reasoning*. This survey attempts to give an overview of the goals and central concepts of Semantic Web Services as well as the relevant research and technologies.

1.1 Goals, benefits, and challenges

The goal of *Semantic Web Services* is to enable *dynamic, execution-time discovery, composition, and invocation* of Web Services [18] that would allow *automated ad hoc interaction* between web-based applications [71]. It builds on the existing *Web Services* technology that provides software developers a framework for accurate documentation of services to allow the development-time discovery, composition, and statically coded invocation of services in the Web.

The aim of Semantic Web Services technology is to automate these tasks that currently require human software developers. But how can this be achieved? The solution approach adopted in most of the research is to use the emerging technology of the *Semantic Web* [10] as the representational infrastructure for precise machine-understandable descriptions of services. This would form the basis for automatic, execution-time reasoning about the types, structures, and properties of services. Some of the required reasoning capabilities come directly from the Semantic Web infrastructure – e.g. terminological reasoning – but there is also need for advanced reasoning methods developed in AI planning research. As a combination of these technologies agents can describe services in accurate manner, discover promising services, compose them into complex workflows, and determine the correct way to invoke them.

The anticipated benefits of the Semantic Web Services technology relate to more up-to-date, comprehensive, and effective utilization of Web Services. The decision to use a particular service is made at the execution-time instead of development-time of the service requestor software. It is therefore possible to use services that are not available, or even existing, at development-time but are only discovered at execution-time. This will dramatically improve the usability of Web Services technology in volatile environments, such as in mobile and ubiquitous computing, where the landscape of available services can change rapidly, for example, when a device moves from one space to another.

In addition, there are anticipated benefits that are not restricted to mobile systems, but apply to all open environments, such as the Web, where the available services evolve over the

time and many different kinds of services may be available to satisfy the particular goals of an agent. The late binding implies a shorter time span between the decision to use a service and its actual invocation, which has several benefits: newer, cheaper, or better services can be used, new and improved versions of old services can be utilized immediately, and the likelihood of the unavailability of a bound service is reduced.

It has also been suggested – e.g. in [95] – that the techniques for dynamically discovering and composing services could be used in service-oriented computing, for example, in enterprise information systems. The functionality of the system could be broken into small logical units with service interfaces. The underlying infrastructure would – at the execution time – take care of discovering, composing, and delivering the desired functionality to users.

However, the challenges to the development of Semantic Web Services are also considerable. First, there is a need to develop shared conceptual models about services that would satisfy the *high accuracy and quality demands* of automated action that can have real economical and operational effects. The situation is completely different than in browsing (a typical mode of using the Web), where human evaluation and intervention happens at every step, and the only cost of wrong decisions is the time lost. In automatic reasoning about real services small deviations in early steps can result in large and costly errors down the road. Second, the reasoning required for service discovery and composition is *computationally complex*; it is not known how well the reasoning tools will perform in the real-world service environments.

At this point it is still too early to tell how the benefits will be realized and how the challenges will be tackled. In any case, the goals of the Semantic Web Services are relevant and it seems obvious that the research and development of Semantic Web Services technology will influence the development of mainstream Web Services technology, most probably through gradual adoption of ideas and methods.

1.2 Background

How does the concept *Semantic Web Services* relate to concepts *Semantic Web*, *Web Services*, *semantic*, *Web*, and *service*? In the following we try to clarify and elaborate the meanings and relationships of these concepts.

The *Web* was originally introduced as a distributed system to *publish and access documents* across the Internet. The architectural model of Web is *client-server* and it is based on three main technological components: (1) *URIs* that provide a global addressing scheme for documents, (2) *HTTP* that is a stateless request-reply protocol to retrieve documents by their URIs, and (3) *HTML*, a markup language that can be used to specify the *presentation structure* of a document and to include *links to other documents* using their URIs. The primary software components of the Web are *Web servers* (at the server side) that host HTML documents and *Web browsers* (at the client side) that provide an interface for users to render HTML documents and retrieve linked documents with simple user interface commands. The main manner of use of the Web has been *browsing*: a user retrieves, evaluates, and optionally reads documents with a Web browser. [25, pp. 9-15]

Since the beginnings the Web has expanded to many directions. The types of information presented have gone beyond ordinary documents to cover all kinds of data (weather reports, transportation schedules, restaurant menus, etc), audio and video clips and streams, etc. An attempt to manage the varied content types has led to the specification of a general markup

language framework called XML [103]. Numerous domain-specific representation languages have been created on top of it: SVG for vector graphics [104], SOAP for messages [113], XUL for graphical user interfaces [52], and so on.

The *Semantic Web* and *Web Services* are two large initiatives, building on the notational ground provided by XML, to improve the possibilities for computational agents (or "software systems" or "machines") to act on the infrastructure provided by the Web. The approaches of the two initiatives are quite different: the Semantic Web is *declarative representation* approach to describe domain information in machine processable way, while Web Services provide a mechanisms to *invoke services* across the Web in a way that resembles remote procedure calls.

The Semantic Web initiative attempts to define controlled vocabularies or *ontologies* – set of conceptual terms labeled by URLs – that can be used in XML documents to give XML structures the semantics required by automatic reasoning. But what does the term *semantic* mean in this context?

As pointed out in [22], term *semantics* refers to the *relationship between linguistic objects and their meaning*, and a linguistic object can only be *meaningful to an agent*, human or mechanical. The Web can be considered as a dynamic repository of linguistic objects, and the usefulness of the Web shows that these objects are meaningful to a wide human audience, and often in more than one way. In that sense, the Semantic Web can not be seen as an effort to bring meaning to something lacking it. Quite the contrary, the goal of the Semantic Web is to *restrict the possible meanings* that agents can associate with linguistic objects in the Web. Ultimately the set of possible interpretations should be reduced to a *single interpretation shared by all agents*. This would allow software to be built without the problems of ambiguity, and the resulting need for semantic negotiation [77] between agents, or worse yet, wrong conclusions and actions due to misunderstandings.

It is difficult to achieve this kind of unanimity, and as of now we do not know whether it is possible in any wider scale. It is obvious that it is easier if the domain is restricted and agents belong to a small and homogeneous community.

At the concrete level, the Semantic Web initiative has resulted in a stack of language specifications built on top of XML: Resource Description Framework (RDF) [101], RDF Schema (RDFS) [108], Web Ontology Language (OWL) [106], and rule languages still in development. There are also some ontologies available, for example, the Dublin Core metadata standard [27].

The development of the *Web Services* technology has been motivated by the evolution of the patterns of use of the Web from simple browsing to more general interaction with the server through HTML forms and web applications. These patterns of use still work within a general-purpose client, the web browser. However, even with application specific extensions and plugins, the use of a browser restricts the potential scope of applications. There are tasks that do not need the kind of user interaction that browsers support (e.g. in many enterprise systems) and the browser would only complicate the implementation of such systems. Moreover, these tasks do not necessarily obey the simple request-reply pattern of interaction. They also need principled ways to specify and communicate structured data. For implementing *application-specific clients* on the Web there is thus a need for more complex mechanisms to describe available services and to interact with them. This is the problem that Web Service technologies try to solve. [25]

The central part of Web Services technology are descriptions of the interfaces of services – specified in Web Services Description Language (WSDL) [114] – to make it possible to invoke them over the Internet. There is a number of additional, so-called *WS-** specifications dealing

with many other aspects of Web Service use: orchestration, choreography, addressing, security, resource states, and so on.

WSDL is meant to provide the human developers the sufficient information needed in the development of a static piece of software that calls a set of predetermined services. This kind of static approach is suitable for stable and closed environments with little redundancy in the set of available services. Examples are centrally managed environments like enterprise systems.

The interface description given by WSDL is *purely syntactic*. It defines the structure of messages – the fields, the order of fields, and datatypes of fields – sent between the service requestor and service provider. However, the meaning of different fields or constraints between the values of fields can only be documented as free-form text. There is no principled ways to specify or constrain the meanings in a machine-understandable manner, and consequently there must be a human developer that decides what services will be called, and designs the systems to invoke those services.

By describing the services with technologies provided by the Semantic Web, there is possibility to use uniquely identified names for services and their properties, and to represent constraints between different properties. This should enable the automation of the discovery, composition, and invocation of services and thus make it possible to perform these tasks in the execution time of software, rather than in design time.

1.3 Structure of the survey

The survey will first look at the concept of service to determine how they should be modeled for automatic discovery, composition and invocation. Then and overview of the underlying technologies – Web Service and the Semantic Web – is given. This is followed a presentation of the main elements of different Semantic Web Services frameworks: OWL-S, WSMO, and SWSF. After that the research on reasoning technologies relevant to Semantic Web Services is given especially focusing on service composition and planning, mediation, and logic programming.

Chapter 2

Services

Given the goal to describe services in a semantically rich manner to enable automatic reasoning based on their properties, the natural starting point would be the question *"What constitutes a service?"*.

2.1 Definitions

The concept of service has become a way to organize behavior and interactions in many areas of information and communication technology, such as operating systems, distributed systems, mobile and ubiquitous computation, digital libraries, data management, information systems, and electronic commerce [29]. Outside information technology, services have traditionally meant *economic activities done by one entity on behalf of another*. There is a lot of literature about service engineering, marketing, and management, see for example [65]. Consequently, there are different and somewhat incompatible ways to describe, manage and provide services.

1. *Service as an activity.* The approaches that aim at reasoning about services, e.g. in agent systems [53] or in AI planning [78], usually adopt a straightforward view of services as *activities* (that is, simple or complex tasks) executed by an organization on behalf of a customer. In agent systems services are thus abstractions of business processes. This abstraction typically serves the task of service composition that allows the creation of inter-organizational workflows and resulting virtual enterprises. [29]

2. *Service as a software component.* In middleware systems, data management, and distributed systems a service is regarded as a set of interrelated software functionalities that facilitate the implementation of different kinds of applications. A service is seen as a *software component* dedicated to a particular *aspect* of application functionality, such as transaction services, replication services, or authentication services. Services are also treated in this manner in networking (e.g. name services, directory services) and telecommunications (e.g. roaming services, voice mail services). [29]

   > "We use term *service* for a distinct part of a computer system that manages a collection of related resources and presents their functionality to users and applications. [...] The only access we have to a service is via the set of operations that it exports." [25]

> "A *service* therefore is a discreet domain of control that contains a collection of tasks to achieve related goals" [55]

3. *Service as interactive exchange of intangibles.* In the areas of economics, management and marketing, services are regarded as *intangible* equivalents of goods, the difference being that the provision of service does not result in ownership. Instead, services may result in different kinds of changes in the state of the world: in customers conditions, possessions, and in their intangible assets. The exact nature of changes is usually much more dependent on the specific situation and properties of the customer than in the exchange of goods.

> "Service is an act or performance offered by one party to another. Although the process may be tied to a physical product, the performance is essentially intangible and does not normally result in ownership of any of the factors of production." [64]

> "Services are economic activities offered by one party to another, most commonly employing time-based performances to bring about desired results in recipients themselves or in objects or other assets for which purchasers have responsibility. In exchange for their money, time, and effort, service customers expect to obtain value from access to goods, labor, professional skills, facilities, networks, and systems; but they do not normally take ownership of any of the physical elements involved." [65]

> "A service is a time-perishable, intangible experience performed for a customer acting in the role of a co-producer." [34]

The difference between the first two meanings – services as activities versus services as software components – can be confusing, especially since the former is generally adopted in Semantic Web Services literature and the latter in the Web Services terminology. However, if the operations provided by a software component can be invoked independently, the difference is actually rather superficial. Instead of modeling a service as an activity, it is possible to model the operations of a service as activities.

If services can be regarded as activities, the last one of the above meanings just underlines the roles of service provider and service requestor, the value of service to requestor, and the variability of service that depends on the requestor. It also stresses that services are not focused on the creation of artefacts, and hence the internal structure or properties of artefacts are typically not crucial in the model of services, as they would be for example in manufacturing activities.

There seems to be some aspects of services that are generally agreed on [76]: (1) services are *actions* performed by an entity *on behalf of* another; (2) services are an *asset*: they have an inherent value that service provider transfers to the service requester; and (3) services can be *decomposed* into a set of simpler services (sub-services).

It is also worth pointing out that the concept of a service is not overly flexible. There are many activities that cannot be naturally modeled as services, most notably peer-to-peer collaboration between agents.

In the following, we describe two aspects of services. Firstly, *service provision* deals with questions around an occurrence of a service: *who* are involved and *why* and *how* are they

involved with the occurrence. Secondly, *service description* deals with the aspects of service that the participants need to discuss with each other.

2.2 Service provision

The interaction that leads to the occurrence of a service has distinct characteristics. It has certain participants, the participants are driven by their goals, and the interaction can be seen as a process consisting of several phases in which one or both of the participants are involved.

2.2.1 Participants

The two principal participants in an occurrence of a service are the

1. the *service requestor*, also called as the *client* or the *customer*, and
2. the *service provider*, also called the *server*.

It should be stressed that both requestor and provider are *relations of an agent to a service*; they are not classes of agents, since one agent can well be the requestor of one service and the provider of another.

There may be other participants acting in mediating roles in the service discovery and provision process. Some examples are the following:

- *matchmaker* attempts to discover services that match the demands of requestors,
- *broker* interacts with the service provider on behalf of the service requestor, and
- *registry agent* stores information about services and provides it to requestors.

The functionalities of these different kinds of mediating agents are overlapping. None of them has an independent logical role in the service provision process. Depending on the architecture of the service system, their functionality can also be incorporated into the service provider and service requestor agents.

2.2.2 Goals of participants

Both the service requester and the service provider have their own goals in the service interaction process [18]. In general, the goal of the requestor seldom coincides directly with one particular service. Rather, an individual service may just be one possible way to getting closer to the goal of an agent.

For instance, if an agent wants to get to Stockholm from Otaniemi, there is no single service available to realize that goal. However, the agent can first use a bus service or a taxi service to get either to the harbour for a subsequent ferry service, or to the airport for a subsequent flight service to Stockholm. This is an example of how the variable needs of customers can be satisfied with simple services that are easy to compose, rather than with complex services trying to accommodate all possible variations in customer requirements.

The goals of a service provider usually deal with the economic compensation or other benefits (e.g., the feeling of contribution or influence) that it receives from the provision of the service. The provision of a high-quality service can naturally be one of the goals, but often it is subordinate to the goal of making profit with the service.

A common way to specify goals is through the notion of a state that needs to be established. For example, the goal of requestor could be that "the location of Me is Stockholm". Similarly the capabilities of services can be described through the states that a service is capable to establish. For instance, a service provider can inform that a particular transportation service has the capability to establish a condition that "location of ?passenger is Stockholm". The discovery of an appropriate service can now happen through matching the goal of the requestor with an effect of a service.

Although it is typical to regard goals as states to achieve and services as means to those goals, there are also situations where a goal cannot be represented as a desired *end state*. For example, if an agent wants to make a round-trip from Helsinki to Stockholm and back to Helsinki, it is not sufficient for the agent to specify its goal simply as "being in Helsinki". If goals are modelled as state, two goal states need to be specified: first "being at Stockholm" and after that "being at Helsinki". This, however, makes the representation of goals significantly more complex; the states need to be associated with temporal extents or ordering relations. Some researchers have suggested that a better way to represent goals would be as an activity to execute [98]. For example, a "round-trip to Stockholm" .

2.2.3 Interaction process

The service interaction process consists of the actions that the service requestor and service provider take to fulfill the goals of the requestor. In [18] the interaction is seen as driven by the goals of the service provider that creates a *service provision process* and the goals of the service requestor that create a *goal achievement process*. These two create an interaction process that can be divided into the following phases:

1. *Discovery*. The requestor finds a set of candidate services using a service registry where the provider has previously registered an available service. To query available services, the requestor uses an abstract characterization of a desired service.

2. *Engagement*. The requestor negotiates with the providers of the candidate services until it finally comes to an agreement about one of the services. The requestor and provider need to talk about the service in much more concrete and detailed manner than what was necessary in the discovery phase. The result is a *service agreement* that both the requestor and the provider are committed to.

3. *Enactment*. The initiation and monitoring of the execution – i.e., delivery and consumption – of the agreed on service. Upon termination, if the contracts objectives are not accomplished, a compensation protocol can be used restore financial equity.

Invocation is the term that denotes the process which begins with the formation of binding *service agreement* between requestor and provider [76]. The agreement does not need to be explicit. In addition, when a service provider – for example, a Web server – has declared a general commitment to provide a service to any requestor, the formation of a service agreement is reduced to a *call* made by the requester.

2.3 Describing services

In the service interaction process the participants need to describe services to each other: the provider describes the services offered, and the requestor the services desired. The descriptions exchanged in the service interaction process are typically partial, and during a successful process the overall description of the service should become more refined and comprehensive. Throughout the process, the service description framework should provide the means of mapping between the provider's description of its offerings and the requestor's description of its needs and goals [95].

The participants need to agree on three aspects of a service: (1) *functional*, (2) *behavioral*, and (3) *non-functional* [93]. *Functional* aspect describes *what a service can do*. Examples of simple functional descriptions are the following: "print a document", "translate text", "ship goods", or "check a car". Behavioral aspect describes *how the functionality can be achieved* both in terms of the externally visible interaction and the internal decomposition into smaller functional pieces. For example, "call a call center, give your location, wait at the location until service personnel arrive, instruct the personnel, accept the service, pay". Finally, the *non-functional* properties capture *constraints over the previous two aspects* [93], such as availability, costs, quality, and so on.

2.3.1 Functionality

Functionality is arguably the most essential aspect of a service. The desired *functionality is the reason* for an agent to invoke a service. Non-functional properties, such as price or availability, are secondary in the sense that they only constrain the selection of a service, once functionally suitable candidates have been identified.

It is, however, complicated to specify functionalities in a general and comprehensive manner. The following approaches have been identified in the literature [69, 67, 39]:

1. *Capability category* – the similarity with the members of a known category. Each sufficiently homogeneous functionality is represented as a class in a functionality hierarchy. For example, there can be a class of `travel service` that has subclasses `transportation service` and `accommodation service`. `Transportation service` can have further subclasses `flight service`, `bus service`, and so on and `accommodation service` subclasses `hotel accommodation service`, `bed-and-breakfast service`, and so on.

2. *Structural capability description* – a many-faceted representation of the properties of a service. In [39] capabilities are represented as verb clauses using a case-grammar style of formalism. Each capability consists of a verb, that specifies what is to be done, and a number of roles, or slots, which specify the parameters to be used in the action. For example, the estimation of the closure date of a particular transportation movement could be specified as:
   `estimate OBJ closure-date OF transportation-1`.

3. *State transformation* – the relation between the state before and state after a service. For example, a particular flight service may specify that in the state before the execution a passenger is in London and in the state after the execution in Helsinki. In the usual terminology, the conditions that must hold before a service can be started are called

preconditions and the conditions that will hold in the end are called *effects*. This representational approach is needed with *world-altering services* [69], for instance, flight booking service, or electronic commerce services.

4. *Input-output function* – the relation between the inputs and the outputs of a service. For example, a flight passenger could use a service to query flight schedules by providing as inputs the source and destination of a desired flight and the time period of planned trip. The output of the service would be a list of possible flights. This representation is especially suitable for *information-providing services* [69] such as local weather forecasts, flight information provision, and cameras.

If compared to procedure call in a programming language, *inputs* and *outputs* correspond to the *parameters* and *return values* while *preconditions* and *effects* correspond to the *assertions* and *side-effects*. It should be emphasized that functionality means the *relation* between these and the body of a procedure implements this relation. *Categories of functionality* is not commonly used in programming, since it is difficult to develop sufficiently expressive and precise classifications.

Categorization of capabilities has been used in all kinds of business catalogues and yellow pages targeted for human use. There are existing industrial classification system (UNSPSC, ISIC, NAICS, etc.) that could form a basis for service classifications.

Classifications can be used in the service discovery as a fast way to identify candidate services. At the same time it should be stressed that the knowledge of the class of a service gives only a rough picture of what it actually does. Consequently, it is likely in many domains that many of the candidates identified just on the basis of a category are not actually applicable. To find out, the requestor and provider need to engage in information seeking dialogue about almost all properties of the service.

Classification is also problematic as a general solution for representing functionalities of services. Services are artefacts – i.e. man-made things – and there is no natural classification for artefacts. Any hierarchical classification scheme can – and often will – be broken by new classes that combine properties of two or more existing classes (e.g., cars and trucks).

Structured capability descriptions make it possible to specify much more information about a service. They are more flexible in the sense that they can be used to specify the relevant aspects of a particular type of service. In addition, different kinds of services can be described in different level of detail. The EXPECT reasoning system [39] uses structured descriptions of capabilities based on description logics. The capabilities can be described as union of capability classes, the values of their properties can be constrained, and so on. The automatic classifier can match the sought out capabilities with the capabilities offered by existing services.

The *state-transformation* and *input/output-function* approaches together define so-called *IOPE*-properties of a service, namely inputs, outputs, preconditions, and effects. This is the common way to specify the functionality of services in Semantic Web Services frameworks. The following is an example of IOPEs presented in [105, parag. 4.2.3] (with relevant parts emphasized).

> To complete the sale, a book-selling service requires as **input** a *credit card number* and *expiration date*, but also the **precondition** that the *credit card exists* and *is not overdrawn*. The result of the sale is the **output** of a *receipt* that confirms the proper execution of the transaction, and as **effect** the *transfer of ownership* and the

physical transfer of the book from the warehouse of the seller to the address of the buyer.

The state transformation approach has been used in the classical AI planning research to compose action sequences to satisfy given goals. However, in that area there has been a proposals to extends the simple precondition-effects -models in various ways:

- *Conditional effects.* It is apparent that practical activities do not always produce the intended effects. They can fail for various reasons or produce alternative effects in some circumstances. There is thus an element of nondeterminism present and a consequent need to represent multiple *different effects qualified by conditions* in which they will result.

- *Maintenance goals* Unlike the achievement goals that define conditions that need to be achieved – that is, hold in the end of the execution – there can also be conditions that must remain true all the way during the execution of a service. For example, the maintenance of some security or integrity conditions. In agent systems maintenance goals are typically managed in a reactive manner, i.e., acted on only during execution if such condition becomes untrue. There are proactive approaches that attempt to anticipate possible failures and plan to prevent conditions to become untrue [28].

- *Temporally extended goals.* A condition can be associated with the time period in which it holds. The period can be specified in absolute time or as depending on other conditions. Extended goals define desired behaviors not only in term of final states to be reached, but also in terms of conditions on the service execution paths.

The fact that a need for these kinds of extensions has been identified in different domains suggests that the simple state transformation model based on preconditions and effects may be insufficient as a generic solution for service modelling.

2.3.2 Behavior

The *behavior of a service* describes *how* the service achieves its functionality. For example, the behavior of a service whose function is to ship goods might be describes as: receive the cargo manifest, load the goods, check the mounting, ask for permission to leave, move to target location, find the contact person, unload the goods, and get signature.

There are two aspects of the behavior of a service [93]:

1. *Externally visible behavior* refers to the aspects of service occurrence that involve interaction between the service provider, the service requestor, and possible other agents related to the service. This includes the visible changes that the service provider will make to the state of the world, and the communicative behavior specifying what kinds of messages will be sent, what are the message layouts, when will the messages be sent, what kinds of interactions can result from a sent message (message exchange patterns), and so on.

2. *Internal behavior* that describes the process – or the workflow – of a service. How the functionality of a service is achieved by aggregating other services? There are basically two aspects of the process:

   (a) *Constituent services.* From what services are this service aggregated from?

(b) *Constraints and conditions.* How are the constituent services combined?

The constituent services typically have constraints on their execution order: the execution of a service can be started only when other services have been completed. There can be different kinds of reasons behind an ordering constraint. Some *artefacts or information can flow* from one service to another. The services may *use shared resources* whose limited capacity does not allow parallel execution, resulting in a disjunctive ordering constraint: either before of after. One service may *destroy the desirable state changes* (e.g., order, or cleanliness) established by another.

The ordering constraints can be approximated either with explicit links or precedence relations between services – in a *graph-like* manner as in project networks – or with control structures such as sequence, parallel execution, and so on – in a *block structured* manner as in programming languages.

The execution of some constituent services may be *conditional* as they may depend on unknown and uncontrollable external conditions. The process may also contain *loops* where certain services are iterated until a desired condition holds. Services may also encounter *exceptional situations* and terminate in abnormal way, which requires reactive measures from the service provider.

If the service allows for alternative internal processes, it is important to understand who can make the decision between the alternatives. If the service provider makes the decisions, the process specification can be thought of as a program or execution specification for the service. If the decision is made by external agent, e.g. service requestor, the process is a specification of a interaction between provider and requestor.

The externally visible behavior is naturally a result of the internal process of a service. However, there can different internal processes that lead to same external behavior.

2.3.3 Non-functional properties

Non-functional properties can be regarded as *constraints on the functionality and behavior* of a service. They play an important role in service selection, negotiation and monitoring processes, as well as in service contracts or service level agreements if such are formulated in an explicit manner.

The research reported in [76] identified the following kinds of non-functional properties based on a review of existing commercial services:

1. *Availability*-related properties. The availability of a service can be subject to different kinds of constraints. Specific properties are *temporal availability* that tells *when*, and *spatial availability* that tells *where* the service is available. The representations of these properties can be complex, especially if mobility must be taken into account.

2. *Channels.* Service interaction can be regarded as occurring through channels that are characterized by the endpoints, the information being transmitted, and the interaction pattern that occurs. A service can be accessible through multiple channels with different properties.

3. *Cost*-related properties. The definition of the compensation provided by the service requestor to the service provider has a number of aspects.

Charging style specifies the way that the price is determined; for example, fixed amount per service delivery (e.g., fixed price local telephone call), based on a unit of measure (length of a long distance call), or on a percentage on some aspect of a service (by commission). The different styles can be mixed and can also be redirected to external parties (e.g., to advertisers).

Settlement model defines the process of fulfillment of the mutual obligations of service provider and service requestor. Basic settlement models are *transactional* and *rental*. The settlement can be specified in an explicit form in a *settlement contract*.

Payment-properties relate to the *schedule* – the number and timing – of monetary transfers. The timing can depend on the occurrence of other events, for instance, on the end time of a successful service delivery. Also relevant are constraints on the *payment instruments* such as cash, cards, checks, or vouchers. Payment instruments are characterized by a number of properties such as usability over the network, acceptability to receiving party, traceability, refutability, liquidity, security, expiration, transferability, immediacy, and negotiability.

Discounts and penalties are properties related to the costs of a service. A service provider may use discounts in various ways to attract new customers or to reward loyal customers. Penalties relate to effects of non-compliance with some obligation of service provision.

4. *Quality*-related properties. A measure of the difference between expected and actual service provision. For example, the customer perception of the quality can be measured through five dimension: *reliability* means the dependability and accuracy of the service, *responsiveness* means the promptness and willingness of the service provider to assist, *assurance* deals with trust conveyed by attributes such as knowledge and courtesy of a provider to a requestor, *empathy* for personalized attention and caring provided, and *tangibles* for concrete aspects of service (e.g. orderliness).

   A service provider can be committed to a certain level of quality, which can be specified in an explicit manner in a *Service Level Agreement*.

   There are two related concepts worth mentioning in this context: *Quality of Service* (QoS) means quality properties of a service that can be *objectively measured*, and *Quality of Experience* (QoE) refers to *subjective evaluations* by the service requestors regarding how satisfactory the service delivery was.

5. *Security and trust*. Security alleviates concerns relating to identity, privacy, alteration and repudiation between parties. With composite services, there are a lot of difficult questions about security management: What is the relationship of authentication of a service and authentication of its subservices? Should service compositions have security certificates? What happens it some subservices require security and others not? Do all subservices obey the same policy with respect to handling of the client's information?

   Trust becomes and important issue in an open environment characterized by lots of interactions among strangers: people, corporations, agents. Trust deals with the management of interactions at the application level [86]: while security is about authenticating another party or authorizing actions, trust is about a party acting in the best interest of another and choosing the right actions from those authorized. Trust relate both to the *intentions* and the *competence* of an agent [76]. In [86] identifies the following ingredients of trust:

capability, *sincerity*, *helpfulness*, *duty*, *predictability*, and *understanding*. There has been attempts to manage trust, for instance, with reputation mechanisms.

6. *Ownership and rights*. Provision of a service does not usually create ownership. However, service requestors typically have following kinds of rights. *The right to comprehend* means that a requestor has the right to question the service provider about the service to better understand it. *The right to retract* means that the requestor does not need to request a service refined into a negotiated contract. *The right of premature termination* means that requestors may have the right to prematurely terminate a service. *The rights of suspension and resumption* mean that the delivery of a service can be temporarily interrupted.

It should be noted that in some service description frameworks the term "non-functional properties" is often used in a sloppy or confusing manner. In WSMO [30], for instance, the term is applied to properties of the *description* of a service, not properties of the service itself. Examples of such description-related properties include *creator*, *date*, *format*, *language*, *owner*, *publisher*, and *version*. These should be called "properties of service descriptions" or something like that, instead of non-functional properties.

Chapter 3

Web Services

While the *Web* provides a solution to flexible human-machine interaction, *Web Services* aim at flexible machine-machine interaction. The concept refers specifically to mechanisms that *software agents* can use to access services in the Web and should not be confused with those services in the Web that are designed for human access.

The definition of different aspects of Web Services has gradually grown into a large set of complementary and sometimes overlapping and competing specifications. The core specifications come from W3C[1], a body responsible for the development of the Web, and OASIS[2], an industry consortium for interoperability standards. In this chapter we try to give an overview of those aspects of Web Services that is needed to understand the role of the Semantic Web Services technology.

3.1 Architecture

Web Services are software systems designed to support interoperable machine-to-machine interaction over a network [109]. They are modular applications with interface descriptions that can be published, located, and invoked across the Web. They provide something like remote procedure call (RPC) over the Internet based on existing protocols and using widely accepted standards (e.g., XML) [25].

Originally, Web Services were characterized by three technologies that roughly correspond to HTML, HTTP, and URIs in the WWW-architecture [31]:

1. *WSDL* (Web Services Description Language) [114] is a standardized way to describe service structures (operations, messages types, bindings to protocols, and endpoints) using XML. Services are regarded as software components that provide a set of closely related operations associated with a set of resources.

2. *SOAP*[3] is a message layout specification that defines a uniform way of passing XML data.

3. *UDDI* (Universal Description, Discovery, and Integration) [73] provides a registry services that allows the software developers to discover available Web Services.

[1] The World-Wide Web Consortium, http://w3.org

[2] Organization for the Advancement of Structured Information Standards, http://oasis.org

[3] SOAP originally stood for "Simple Object Access Protocol". This expansion has been dropped as misleading, as well as some later proposals like "Service Oriented Access Protocol". SOAP is now just SOAP.

The underlying architecture shown in figure 3.1 assumes three different kinds of agents: service provider, service requestor, and service registry. The interactions between the agents can be described by three verbs: *publish*, *find*, and *bind*. Publish is the step of where service provider stores the service description to a registry agent. Find is a service discovery performed by the service requestor. Bind is the step that allows the requestor to connect to a Web Service at a particular web location (endpoint) and start interacting with it.

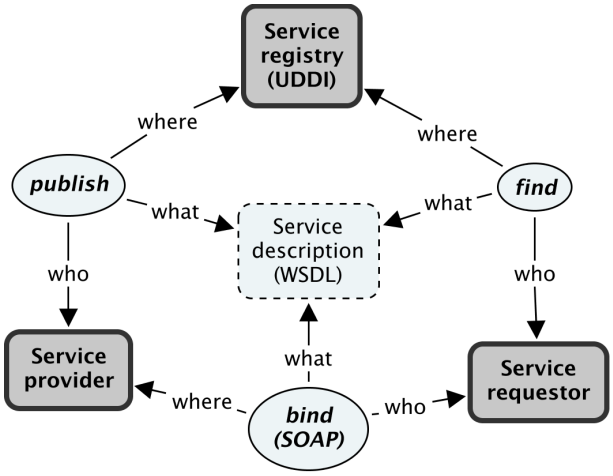

Figure 3.1: Web Services: agents and their interactions

Over the time WSDL has emerged as the key technological component of Web Services. SOAP is just one possible message layout specification to invoke Web Services. The idea of UDDI as the global solution for service discovery was dealt a blow in the end of 2006 when the public UDDI registries were shut down. ebXML [72] has emerged as a strong alternative for such registries.

3.2 Characteristics

Web Services are suitable for loosely coupled, coarse granularity distributed systems that need to cross organizational boundaries. When compared to distributed object models like CORBA [75], Web Services have the following distinguishing characteristics [25, pp. 799-800].

- They are intended to be used in a *global scope*, not just in local organizational contexts.
- Due to a more *loose coupling* between components, they provide a better support for ad hoc distributed computing.
- Web Services are easy to use since they rely on *infrastructure readily available* in commonly used operating systems.
- The use of transport protocols such as HTTP, SMTP or SIP makes Web Services easier to *access through firewalls* of organizations.

Web Services are nominally *stateless*. A Web Service does not remember information – or keep state – from one invocation to another. Services should be independent, self-contained requests, which do not require the exchange of information or state from one request to another,

or depend on the context or state of other services. When dependencies are required, they are best defined in terms of common business processes, functions, and data models, not implementation artifacts (like a session key). Of course, requestor applications require persistent state between service invocations, but this should not mean that the service provider should keep any state.

3.3 Service description

WSDL is a language for describing the interfaces of services. In addition to giving the interface of a Web Service, it provides concrete details on how to access the service (e.g. protocols, URI). There can be different implementations of an interface as well as multiple agents that run an implementation. A service is a collection of related operations that can be accessed over the Internet by sending messages (inputs) and that send back messages (outputs).

The second major version of Web Service Description Language, WSDL 2.0, became the official W3C recommendation in May 2007. The basic concepts of a WSDL description are as follows:

- *Interface as a set of operations.* Each operation is described by
  - *Name* which is a string that identifies the operation.
  - *Set of inputs.* Each input has a *name* and a *type.*
  - *Set of outputs.* Each output has a *name* and a *type.*
  - *Set of faults.* Faults are similar to outputs but represent exceptional conditions.
  - *Message exchange pattern.* Options are request-reply patterns In-Out (client initiated) and Out-In (server initiated), one-way notifications In-Only and Out-Only, and guaranteed delivery patterns Robust-In-Only and Robust-Out-Only.
- *Set of message types.* Needed to specify the type of inputs, outputs and faults of operations, unless the message type is one of the built-in types of XML. Each type is specified as an XML Schema [102] definition.
- *Binding.* Specifies how the messages are to be communicated. The type of messages is typically SOAP and the protocol HTTP; these are not however, the only alternatives.
- *Service as a set of endpoints.* Each endpoint is specified through
  - *Address.* The URI where the invocation message is sent.
  - *Binding.* As specified above, what kind of message is sent and what protocol is used.

  There can be multiple URLs for one binding and different URLs for different protocols.

Below is an example of a Web Service (adapted from WSDL Primer [111]) that allows the client (1) to check the availability of and (2) make reservations for hotel rooms. The structure of the `checkAvailability` message is directly described with the XML Schema while the structure of the `makeRevervation` message is imported from another location.

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/2004/08/wsdl"
    targetNamespace= "http://greath.example.com/2004/wsdl/resSvc.wsdl"
    xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc.wsdl"
    xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc.xsd"
    xmlns:wsoap= "http://www.w3.org/2004/08/wsdl/soap12"
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <types>
    <xs:schema
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://greath.example.com/2004/schemas/resSvc.xsd"
        xmlns="http://greath.example.com/2004/schemas/resSvc.xsd">
      <xs:element name="checkAvailability" type="tCheckAvailability"/>
      <xs:complexType name="tCheckAvailability">
        <xs:sequence>
          <xs:element  name="checkInDate" type="xs:date"/>
          <xs:element  name="checkOutDate" type="xs:date"/>
          <xs:element  name="roomType" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
      <xs:element name="checkAvailabilityResponse" type="xs:double"/>
      <xs:element name="invalidDataError" type="xs:string"/>
    </xs:schema>
    <xs:import namespace="http://greath.example.com/2004/schemas/resSvc.xsd"
               schemaLocation= "http://greath.example.com/2004/schemas/resSvc.xsd"/>
  </types>
  <interface  name = "reservationInterface" >
    <fault name = "invalidDataFault" element = "ghns:invalidDataError"/>
    <fault name = "invalidCreditCardFault" element = "ghns:invalidCreditCardError"/>
    <operation name="opCheckAvailability" pattern="http://www.w3.org/2004/03/wsdl/in-out" >
        <input messageLabel="In" element="ghns:checkAvailability" />
        <output messageLabel="Out" element="ghns:checkAvailabilityResponse" />
        <outfault ref="tns:invalidDataFault" messageLabel="Out"/>
    </operation>
    <operation name="makeReservation" pattern="http://www.w3.org/2004/03/wsdl/in-out" >
        <input messageLabel="In" element="ghns:makeReservation"/ >
        <output messageLabel="Out" element="ghns:makeReservationResponse"/ >
        <outfault ref="invalidDataFault" messageLabel="Out"/>
        <outfault ref="invalidCreditCardFault" messageLabel="Out"/>
    </operation>
  </interface>
  <binding name="reservationSOAPBinding"
           interface="tns:reservationInterface"
           type="http://www.w3.org/2004/08/wsdl/soap12"
           wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP">
    <operation ref="tns:opCheckAvailability"
               wsoap:mep="http://www.w3.org/2003/05/soap/mep/request-response"/>
    <fault ref="tns:invalidDataFault"
           wsoap:code="soap:Sender"/>
  </binding>
  <service name="reservationService"
           interface="tns:reservationInterface">
     <endpoint name="reservationEndpoint"
               binding="tns:reservationSOAPBinding"
               address ="http://greath.example.com/2004/reservation"/>
  </service>
</description>
```

A large part of a WSDL description resembles an interface definition in a programming language. Using that analogy, the operations are like procedures (methods, functions), inputs are like parameters of those procedures, outputs are the (possibly multiple) values returned by procedures, and faults are like exceptions. Type definitions resemble complex types used

in languages such as ML or Haskell. However, WSDL is more flexible than typical interface definitions sublanguages in programming languages.

Interfaces in WSDL can be specified in modular fashion. For instance, same types can be used in different services by including them from a separate document using a URI, and the same interface can be bound to different URIs in different WSDL documents.

WSDL uses XML syntax and the descriptions written in it are therefore human readable. However, in reality, due to the complex and indirect syntax of XML, they could better be characterized as readable for software developers. WSDL cannot be used as a presentation format of services for any wider audiences. Even for software development, there are tools available to specify a service description without the need to work directly with WSDL (for instance, Eclipse WTP), and to invoke a Web Service without the need to deal directly with the details of a WSDL description (for instance, Apache WSIF).

3.4 Composite services

Often the invocation of one Web Service is not enough to satisfy the particular needs or goals of a client. For instance, a purchase of a book from the web may require the invocation of the web services of a credit card company and a shipping company in addition to the bookstore itself. Another example is travel booking, where the services of several transportation and accommodation service providers need to be combined. The need for invoking multiple services to satisfy one need creates a whole spectrum of coordination problems that need to be solved.

The basic approaches to Web Services coordination are known as *orchestration* and *choreography*. When looking from the perspective of one service, orchestration deals with the internal implementation of the service while choreography deals with the external interaction between the service and the clients using the service.

3.4.1 Orchestration

Web Service orchestration means that a new service is specified through recursive composition of existing services. The result is a new executable web service. The coordination approach is centralized, "conductor-oriented". The focus is on the internal implementation of the service, and as such it resembles computer programming.

The current de facto standard among orchestration languages is WS-BPEL (Web Services Business Process Execution Language) [74], formerly known as BPEL4WS. WS-BPEL is an XML language, and as the name suggests, it has the background in business process modeling, an area where competing workflow languages proliferate: WSFL, XLANG, PDL, XPDL, BPSS, EDOC, BPML, WSCI, BPML, Staffware, FLOWer, and so on.

WS-BPEL builds specifically on two earlier workflow specification languages [100]: the IBM's WSFL (Web Services Flow Language) and Microsoft's XLANG (Web Services for Business Process Design). XLANG is a block-structured language whose control structures include sequence, while, switch (for conditional routing), all (for parallel routing), and pick (for race conditions). WSFL, by contrast, allows for directed graphs that can be nested but need to be acyclic. There are no specific constructs for iteration which is only supported through exit conditions: a subprocess is iterated until its exit condition is met.

WS-BPEL combines the graph-based approach of WSFL with the block-structured constructs of XLANG, making the resulting language very complex [100]. It can support a wide

variety of workflow patterns, such as sequence, parallel split, synchronization, exclusive choice, simple merge, multi-choice, synchronizing merge, implicit termination, deferred choice, cancel activity, and cancel case. [115] provides a simple example of two parallel activities (`activityA1` and `activityA2`) followed by one activity (`activityB`). Figure 3.2 shows this example modeled both in a block-structured and a graph-based manner using WS-BPEL:

```
<sequence>
  <flow>
    activityA1
    activityA2
  </flow>
  activityB
</sequence>
```

```
<flow name=''F''>
  <links>
    <link name=''L1''/>
    <link name=''L2''/>
  </links>
  activityA1
    <source linkName=''L1''/>
  activityA2
    <source linkName=''L2''/>
  activityB
    joinCondition=''L1 AND L2''
    <target linkName=''L1''/>
    <target linkName=''L2''/>
</flow>
```

Figure 3.2: WS-BPEL: block-structured (left) vs graph-based (right)

WS-BPEL can be used to model both *executable* and *abstract* processes [115]. An abstract process is a protocol that is used to publish the message exchange patterns between different parties without revealing their internal behavior. An executable process specifies a set of constituent activities, their execution order, the parties involved, the messages exchanged between parties, faults that may result, and the exception handling mechanisms to deal with them. If WS-BPEL is regarded as a scoped programming language, the abstract processes are behavioral interfaces.

Examples of engines for the execution of WS-BPEL processes are Active BPEL Open Engine (open source) and the Oracle BPEL Process Manager. While WS-BPEL does not include a diagrammatic representation, there are graphical tools for the design of BPEL processes, such as Active BPEL Designer.

3.4.2 Choreography

Web Service choreography means that the coordination touch-points between services are described through message exchange patters. The result is descriptive and requires additional steps to produce executable service combinations. The coordination approach is peer-to-peer. A coordination approach based on choreographies has been specified by W3C as WS-CDL (Web Services Choreography Description Language) [110].

A WS-CDL description is a multi-participant contract that describes, from a global viewpoint, the common observable behavior of the collaborating Web Service participants. A choreography is specified as a set of interactions, each consisting of a number of message exchanges among the participants. Below is a fragment of a choreography specification for simple QuoteRequest-QuoteResponse -interaction between a Buyer and a Seller:

```
<choreography name="QuoteChoreography" root="true">
  <relationship type="tns:Buyer2Seller"/>
  <variableDefinitions>
      <variable name="Buyer2SellerC" channelType="tns:Buyer2SellerChannel"
```

```
                roleTypes="tns:BuyerRole tns:SellerRole" />
        </variable>
        <variable name="quoteRequest" informationType="tns:QuoteRequestType"
                roleTypes="tns:BuyerRole tns:SellerRole" />
        <variable name="quoteResponse" informationType="tns:QuoteResponseType"
                roleTypes="tns:BuyerRole tns:SellerRole" />
        <variable name="faultResponse" informationType="tns:QuoteResponseFaultType"
            roleTypes="tns:BuyerRole tns:SellerRole" />
    </variableDefinitions>
    <interaction name="QuoteElicitation" operation="getQuote" channelVariable="tns:Buyer2SellerC">
           <participate relationshipType="tns:Buyer2Seller" fromRoleTypeRef="tns:BuyerRole"
                       toRoleTypeRef="tns:SellerRole"/>
            <exchange name="QuoteRequest" informationType="tns:QuoteRequestType" action="request">
                <send variable="cdl:getVariable('quoteRequest','','')"/>
                <receive variable="cdl:getVariable('quoteRequest','','')"/>
            </exchange>
            <exchange name="QuoteResponse" informationType="tns:QuoteResponseType" action="respond">
                <send variable="cdl:getVariable('quoteResponse','','')"/>
                <receive variable="cdl:getVariable('quoteResponse','','')"/>
            </exchange>
            <exchange name="QuoteResponseFault" informationType="tns:QuoteResponseFaultType"
                      action="respond" faultName="InvalidProductFault">
                <send variable="cdl:getVariable('faultResponse','','')"/>
                <receive variable="cdl:getVariable('faultResponse','','')"/>
            </exchange>
    </interaction>
</choreography>
```

The description is not done from the perspective of any single participant but the view is from above. The description attempts to capture the necessary ordering conditions and constraints under which messages are exchanged. Each participant can use the global definition to build and test services that conform to it. Since choreography only concerns the externally visible behavior of agents, and not their internal behavior, the internal processes can be changed as long as the observable sequence of messages remains the same.

Corporate entities are often unwilling to be orchestrated since that would mean that they delegate certain amount of control of their business processes to their integration partners. Choreography offers a way to clearly define the rules of participation within a collaboration. Each party can then implement its portion of the choreography as determined by their common view. [110]

Since the collaboration parties must implement software based on a choreography specification, this approach also means static binding to pre-determined services.

3.5 Applications

Web Services are currently used in a wide scale in following areas:

- *Grid-computing.* Grids are distributed computing frameworks based on three principles: (1) computing resources are not managed centrally, (2) open standards and protocols are used, and (3) some non-trivial quality of service is achieved. The Open Grid Services Architecture [41] describes service as Web Services, using WSDL and SOAP (but not UDDI). Grid applications require stateful Web Services, which has resulted in a WS-Resource specification.

- *Enterprise systems based on SOA.* Service Oriented Architecture (SOA) for enterprise

information systems has gained a lot of momentum through the the use of WSDL in service description and UDDI for internal registry to locate services.

- *Electronic commerce.* Electronic commerce is an area where various Web Service interfaces have been provided. Examples are the Amazon Web Services[4] and Galileo Web Services[5] in the travel industry.

3.6 Problems

Given the goal to achieve capabilities for ad hoc interoperation among web-based applications, there are several problems in the plain Web Services approach.

1. *Insufficient coverage of service properties.* The current WSDL descriptions are meant to describe the interface that can be used to invoke a service. WSDL does not support the description of its (1) functional properties (neither through preconditions and effects, or structured capabilities), (2) non-functional properties (costs, quality, availability) other than channels, or (3) internal behavior. It has, however, partial support for the description of the externally visible behavior, i.e., message types and simple message exchange patterns.

   It means that many semantically interesting properties (functionality, non-functional properties, and internal behavior) of services should be described and communicated with other means than WSDL. These kinds of properties are important in almost all tasks in automatic ad hoc interoperability between services: discovery, selection, substitution, and composition.

   However, WS-BPEL, WS-CDL, WS Policy and other recent specifications provide some representational capabilities for these missing areas.

2. *Free-form terminology in service descriptions.* WSDL gives a structural but purely syntactical description of web services. It is like the API documentation of a library of a programming language in the sense that the names of operations and types reveal as much about the meaning of a service (e.g., expected behavior and interaction patterns) than names of procedures and types in programming languages. Therefore, the use of a web service or coordination of multiple web services always requires a software developer that reads the documentation and decides whether to use the service and how to do that.

   The Semantic Annotations for Web Services (SA-WSDL) is a W3C specification that attempts to alleviate this problem by introducing the following attributes that can be added to XML Schema declarations and WSDL interfaces:

   - `modelReference`: Specifies the association between a WSDL component and a concept in a semantic model.
   - `liftingSchemaMapping` and `loweringSchemaMapping`: Specify mappings between semantic data and XML that can be used during service invocation.

[4] `http://www.amazon.com/`
[5] `http://www.galileo.com/galileo/en-us/agency/Products/GalileoWebServices.htm`

This makes it possible to have unique names to use in descriptions to avoid the problems caused by the accidental similarity of the terms.

It is worth noticing that the Semantic Annotations Working Group attempted to standardize the IOPE properties of services but could not agree on the proper way of doing that.

3. *Static binding to predetermined services.* The dependence of human intervention in the discovery of and interaction with a service puts limits to the idea of machine-to-machine interoperability over a network. Software development is slow and tedious, and cannot take advantage of all the services that could possibly be introduced to the Web. The pace of introduction of service to (and possibly also retraction from) the Web can be much higher than the pace in which programmers can adapt their systems to these services.

   In volatile systems – e.g., in mobile and ubiquitous computing – the normal state of the system is characterized by frequent appearances and disappearances of services [25]. The exploitation of services in such environments cannot depend on static bindings to specific URIs of services. The question is how to increase de-coupling of components so that interchangeable services or functionally equivalent service combinations could be utilized. This requires some ways to represent when two services or service combinations are interchangeable.

4. *Simplistic publish-find-bind model.* The Web Services use is largely supposed to follow the simple *publish-find-bind* model shown in Figure 3.1. This model hides many complicated phases carried out by software developers but that must be explicitly be taken into account in automated ad hoc interoperation. The model may have to include many steps in the service interaction process such as define, publish, discover, negotiate, invoke, monitor, and so on.

Chapter 4

The Semantic Web

As a technology, the Semantic Web can be summarized as "knowledge representation meets the Web" [43]. The goal is to create declarative representational notations, i.e. languages, that would enable automatic processing and composition of information in the Web.

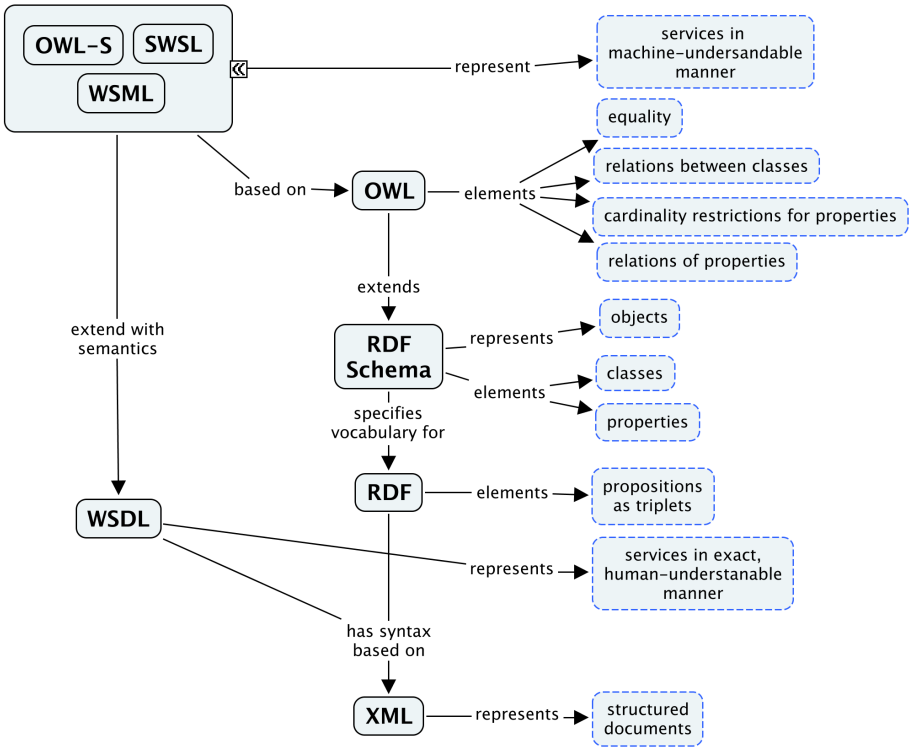

Figure 4.1: The relationships of the languages defined for the Semantic Web

4.1 Languages

The practical approach in the development of the Semantic Web technology is to combine the two key areas of knowledge representation research, *description logics* and *rule languages*, with *XML*, the representational infrastructure of the Web. This development effort has so far specified a set of languages on top of XML, such as RDF, RDFS, OWL, and RuleML.

Figure 4.1 shows the relationships and roles of these languages and their relationships to

Web Service specifications and proposals. The exact syntactical and semantic relationships of different languages are not completely completely understood. Some of the language specification are relatively recent – e.g., OWL – or they may still be under development, like the rule languages. All languages in the figure have not been implemented, for instance, OWL Full.

The original vision of the relationships of different languages was a "single stack architecture" where URIs, XML, RDF, RDFS, OWL, and rule languages form a single stack where each language is seen as a (syntactic and semantic) extension to previous ones. In more recent "multi stack architecture" the description logic and rule languages form two different branches on top of RDFS and a small intersection language of "description logic programs". [58]

4.1.1 XML

XML has originally been designed for the representation of *structured documents*. It provides a syntactical framework for specific types of documents but imposes no semantic constraints on their meaning. XML belongs to a class of *markup languages* that allow structuring information to be inserted into text documents. For instance, part of a text can be surrounded with specific marks to tell that the text represents a name, or an address, or telephone number. XML allows markups inside other markups and hence the representation of *typed hierarchical structures*, like in the following example:

```
<person>
  <name><first>Joe</first><last>Doe</last></name>
  <telephone><areacode>123</areacode><number>111222333</number></telephone>
  <address>...</address>
  ...
</person>
```

The benefit of XML is that it provides a standard notation for information exchange between agents. Using the XML Namespace mechanism, it is possible to include descriptions written in different XML languages to a same document, without the fear of name conflicts. Furthermore, the XML Schema language allows a detailed specification of the structure of XML document, including the SOAP messages used to invoke Web Services.

However, neither the *document-oriented* nature of XML nor the *hierarchical* approach to representation are very useful for knowledge representation and reasoning. It is unnatural to package information in an open, distributed, and evolving environment into closed documents. For example, reasoning often produces bits and pieces of new information. Neither the alternative of regarding these new pieces of information as small documents, nor the encoding of the complete new state of knowledge as a new document, are good solutions. Moreover, the relations between the pieces of knowledge form general graph-like structures. While knowledge can certainly contain hierarchical structures, no single hierarchy can be sufficient in general knowledge representation.

4.1.2 RDF

Resource Description Framework (RDF) [101] is a language that uses the XML notation to represent pieces of information as *propositions* (not documents) and that can form general *graph-like structures* (not just hierarchies). Propositions are represented in the form of *subject-predicate-object* -triplets and terms used are uniquely identified with URIs.

RDF was originally specified for the representation of meta-information about Web resources, including other RDF statements. The overall RDF information is not a hierarchical document but constitutes an open collection of triplets that can be thought of as a labeled directed graph.

The advantage of RDF over XML is that it supports the idea of combining information from multiple sources over the Web and doing inference that may result in pieces of additional information.

RDF has different syntaxes, the normative ones based on XML. The resources and predicates are identified with URIs. An RDF statement itself can be "reified" by giving it a URI. This allows RDF statements to be made of other RDF statements, e.g. regarding their context (time, author, circumstances), status, justifications, credibility, etc.

RDF has different serialization notations. *RDF/XML* is a serialization notation to XML while *N3* [112] and *N-Triples* [107] are more readable formats where each statement is written separately. The following is an example of some meta-data in N-Triples format about Wikipedia-pages describing the concept of 'service':

```
<http://en.wikipedia.org/wiki/Service> <http://purl.org/dc/elements/1.1/title> "Service" .
<http://en.wikipedia.org/wiki/Service> <http://purl.org/dc/elements/1.1/publisher> "Wikipedia" .
```

The first part (subject) is a Web resource, the second identifies a predicate defined in Dublin Core Metadata Initiative and last (object) is a string. In RDF/XML serialization format this would be encoded as follows:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/">
    <rdf:Description rdf:about="http://en.wikipedia.org/wiki/Service">
            <dc:title>Service</dc:title>
            <dc:publisher>Wikipedia</dc:publisher>
    </rdf:Description>
</rdf:RDF>
```

4.1.3 RDF Schema

RDF Schema [108] can be used to specify vocabularies (classes and properties) to use in RDF descriptions. RDF Schema makes it possible to say what kinds of properties can be associated to different resources and what are the relationships of those properties. The main constructs of RDF Schema are:

1. Classes

   - `rdfs:Class` allows to declare a resource as a class using `rdf:type`. For example:

     ```
     foaf:Person rdf:type rdfs:Class
     ex:Joe rdf:type foaf:Person
     ```

   - `rdfs:subClassOf` allows to declare a subclass relationship between two classes. For example:

     ```
     foaf:Agent rdf:type rdfs:Class
     foaf:Person rdf:type rdfs:Class
     foaf:Person rdfs:subClassOf foaf:Agent
     ```

2. Properties

- `rdfs:domain` of an rdf:property declares which class can have the property. For example:

```
ex:brother rdf:type rdf:property
ex:Person rdf:type rdfs:Class
ex:brother rdf:domain foaf:Person
```

- `rdfs:range` of an rdf:property declares to what class or datatype the value of the property must belong to. Continuing the previous example:

```
ex:Male rdf:type rdfs:Class
ex:Male rdfs:subClassOf foaf:Person
ex:brother rdf:range foaf:Male
```

4.1.4 OWL

Web Ontology Language (OWL) [106] extends RDF and RDF Schema with additional representational constructs that, for example, allow to say that two classes are disjoint, that number of values of certain property are limited, and so on. OWL comes in three variations, OWL Lite, OWL DL and OWL Full, from least expressive to most expressive.

OWL is based in part on *description logics* studied in the field of knowledge representation. In description logics, structural descriptions of entities are automatically classified, i.e., their *subsumption* relationships with other descriptions are computed.

The following simplistic example of OWL definitions in the abstract (unofficial) syntax is presented in [5]. The descriptions say that a `driver` is exactly a `person` who has the relationship `drives` with a `vehicle`. A `driver` is an `adult` but not all `adult`s need to be drivers. A `grownup` is exactly an entity that is both an `adult` and a `person`.

```
Class(a:driver complete intersectionOf(a:person restriction(a:drives someValuesFrom (a:vehicle))))

Class(a:driver partial a:adult)

Class(a:grownup complete intersectionOf(a:adult a:person))
```

Based on these definitions, it is possible for the terminological reasoner to conclude that a `driver` is a `grownup`. That is, '`grownup`' is a concept that subsumes '`driver`'.

Due to the use of description logics, the expressivity of the representation language should be kept in minimun to avoid intractable computational problems (see e.g. [1]). For this reason, the language features of different OWL variations are hand-picked with the *objective to keep terminological reasoning practical*. Some intuitively useful constructs such as property paths ("an uncle is a bother of a mother") cannot be not included into a language. The OWL variations can be characterized as follows [106]:

1. *OWL Lite* is suitable when a classification hierarchy and simple constraints are sufficient, like in the formalization of existing thesauri or taxonomies.

2. *OWL DL* provides the maximum expressiveness that still retains computational completeness (all conclusions will be computed) and decidability (the computations will finish in finite time).

3. *OWL Full* is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right.

The essential language constructs of OWL as shown in the Figure 4.2, adopted from [106]. OWL Lite does not contain any of the class axioms, no other class expressions besides `intersectionOf`, no `hasValue` construct, and the possible cardinality restriction values are 0 and 1. OWL DL contains all the language constructs show but has limitation on their use. For example, in OWL DL a class cannot be deal simultaneously as a category and an individual, which is possible in OWL Full. Besides the language constructs shown in Figure 4.2, OWL has additional constructs dealing with versioning and annotations.

RDFS Features	**Equality**	**Property characteristics**
- Class (Thing, Nothing) - rdfs:subClassOf - rdf:Property - rdfs:subPropertyOf - rdfs:domain - rdfs:range - Individual	- equivalentClass - equivalentProperty - sameAs - differentFrom - AllDifferent - distinctMembers	- ObjectProperty - DatatypeProperty - inverseOf - TransitiveProperty - SymmetricProperty - FunctionalProperty - InverseFunctionalProperty
Property restrictions	**Cardinality restrictions**	**Header information**
- Restriction - onProperty - allValuesFrom - someValuesFrom	- minCardinality - maxCardinality - cardinality	- Ontology - imports
Class combinations	**Class axioms**	**Filler information**
- unionOf - complementOf - intersectionOf	- oneOf, dataRange - disjointWith - equivalentClass - rdfs:subClassOf	- hasValue

Figure 4.2: The essential language constructs of OWL

4.2 Description logics

Description logics [1] are formalisms for the representation of conceptual knowledge. They can be used to give structural descriptions of entities, such as “Shipment whose source is Helsinki and target is London”.

Description logics aim to take the meaning of various representational constructs seriously. If the semantics of the representation language would sanction some inferences, the representational framework should implement the necessary reasoning. For example, if the language makes it possible to represent concepts ”Car manufactured in China” and ”Vehicle manufactured in Asia”, and to represent the relations of China and Asia (geographic inclusion) as well as Car and Vehicle (subclass), the reasoner should be able to derive subsumption relationship between the concepts (the latter subsumes the former, or in other words, the extension of the former is a subset of the extension of the latter).

In description logics the representation of the domain knowledge is divided into two parts:

1. *TBox* (terminology) contains all sentences describing concepts and their relationships. For example, “every employee is a person”.

2. *ABox* (factual assertions) contains, among other things, all sentences where individuals appear. For instance, "Bob is a person".

This distinction is not significant from the perspective of first-order language that could equally well be used to represent both sentences. However, in practice it is possible to use the division to introduce specialized reasoners to the TBox and get a significantly improved performance over the first-order language.

Terminological reasoning includes the following reasoning tasks:

1. *Subsumption* is the task to determine whether a concept description subsumes another.
2. *Classification* or definition of all subsumption relationships of a new concept description with respect to other concept descriptions. It is used to determine the correct position of a description in a conceptual system.
3. *Concept satisfiability* answers the question whether it is possible that concept has instances.
4. *Satisfiability* of the set of concept descriptions.

   Instance checking, that is, can an individual be an instance of a class),
5. *Retrieval* of all instances of a concept
6. *realization*, retrieval of all the concepts that an individual is an instance of.

The research on terminological reasoning started in the 1980's in KL-ONE knowledge representation system where the determination of subsumption turned out to be undecidable. It soon became evident that if the expressive power of the representation language is increased, the complexity of the required reasoning increases dramatically.

The language that leads to tractable (polynomial time) reasoning has really a very limited expressive power. This is so-called structural description logic FL^- that contains atomic concept names, concept intersection, limited existential quantification, and universal role value restrictions.

The complexity of subsumption in description logic ALC that extends FL^- with atomic and complex negation, is already in PSPACE.

OWL-DL 1.0 uses description logic called $SHOIN^{(D)}$ that in addition to previous representational capabilities includes role hierarchies (e.g., sister is a subproperty of sibling), enumerated classes (value belongs to one of classes mentioned), inverse properties (e.g., parent is an inverse of child), and cardinality restrictions (e.g., node has at most one parent).

OWL 1.1 uses $SHOIQ$ that extends $SHOIN^{(D)}$ with qualified cardinality restrictions and some other minor features. The worst case complexity of $SHOIQ$ is EXPTIME but the highly optimized reasoners (e.g., FaCT++ [97], Racer, or Dlp) still work in practice.

Description logics are used in the semantic web mostly to support the design of ontologies to improve the quality of resulting definitions. A designer can classify a new description to see how it relates to the existing concept descriptions. Classification can be used to produce a hierarchy of concept descriptions for visualization, etc. If a service provider uses an ontology to specify a web service with WSDL, the consistency in the use of the concepts could be checked.

From the perspective of Semantic Web Services this kind of terminological reasoning is done offline, that is, statically and before any functions relevant to the ad hoc interoperation will be performed.

4.3 Discussion

There are some basic tensions among the Semantic Web languages that stem from their different backgrounds. For example,

1. RDF/RDFS is targeted to representation of meta-information while
   OWL is based on first-order language, and

2. OWL is based on classical negation while
   rule languages are based on default negation.

The model and syntax of RDF is too flexible from the point-of-view of OWL [50, 42]. Many OWL constructs need to be encoded in several RDF-triplets. However, these triplets are independent, and from the perspective of RDF there are no requirements that the triplets must always occur together. In addition, there are no requirements for the occurrence of extra triplets. Furthermore, the existence of unusual (e.g., circular) triples cannot be ruled out. The meta-modeling capabilities of RDFS are too strong to be supported by terminological reasoners. As noted also in [106]:

> OWL Full can be viewed as an extension of RDF, while OWL Lite and OWL DL can be viewed as extensions of a restricted view of RDF.
> Every OWL (Lite, DL, Full) document is an RDF document, and every RDF document is an OWL Full document, but only some RDF documents will be a legal OWL Lite or OWL DL document.

If OWL Full turns out to be a language that will not have practical implementations, the language of choice will be OWL DL, and the compatibility between OWL Full and RDFS will be pointless. It has been suggested that the Semantic Web language architecture should be simplified by restricting RDF and RDF Schema (e.g. by suppressing metamodeling features) so that they would become compatible with OWL DL [42].

Another problem deals with the relationship between OWL and rule languages. OWL has its background in first order logic and among other things is based on classical negation. Rule languages as they have been developed and used during past thirty years rely on the default negation (e.g. negation as failure). Moreover, rule languages do not really benefit from the the capabilities provided by OWL. The attempt to combine OWL to rules would only result in an unnecessarily complicated language, whose semantics may be difficult to specify. [58]

The Semantic Web languages are going to be used in the specification of ontologies which have high quality demands. They are supposed to be the basis of automated actions of computational agents, ultimately without any human monitoring and supervision. Such activity is much more fragile than human browsing behavior; small errors can lead into large deviations down the road. Ontology development efforts will be much more extensive than the efforts to develop underlying representation languages have been and it is worth asking whether the current language infrastructure is simple and clean enough to serve as a basis for that kind of work.

In addition to the tensions between languages mentioned above, there is the problem of human readability of the ontology specifications. Currently the XML/RDF syntax of OWL is too complex to support the development work of ontologies at the conceptual level. The syntax does not manage to hide the complexities of the lower language levels and thus gives

a poor syntactic support for higher levels. Hopefully, advanced ontology editors and other tools will provide a simpler model for the developers to allow them to focus on the difficult questions dealing with the analysis and specification of conceptual systems. If and when such tools emerge, an interesting question is, what is the actual language that these editors and other tools actually implement. Would it be possible to give that language a better syntax?

Chapter 5

Semantic Web Services

The development of Semantic Web Services technologies attempt to enrich the underlying Web Services technologies with capabilities supplied by the Semantic Web technologies. In the following we outline the framework proposals for Semantic Web Services and discuss relevant reasoning tasks and technologies.

5.1 Representation frameworks

To work with Semantic Web Services there needs to be a way to represent relevant information (e.g., available services), reasoning problems (e.g., the goals an agent wants to achieve) and solutions to the problems (e.g., service compositions) in a consistent manner. Three large proposals for such representation frameworks have been presented: OWL-S, WSMO, and SWSF. OWL-S uses a description logic (OWL) as the basic language for service ontology. WSMO stresses the loose coupling between services and attempts to achieve this with separate goal and service ontologies and advanced mediator architecture. SWSF is based on OWL-S but improves its process model by incorporating an existing ontology, Process Specification Language (PSL). It uses the first-order language as the basic language for ontologies, allowing better axiomatization of PSL process models than what is possible with OWL.

5.1.1 OWL-S - Web Ontology Language for Services

Web Ontology Language for Services (OWL-S) [105] (formerly DAML-S) has been developed as part of the DAML program funded by DARPA in the years 2000-2004. In 2004 the group submitted a W3C member submission describing OWL-S. The latest version is the pre-release of OWL-S 1.2 [89] from early 2006.

OWL-S attempts to combine the representational technologies of the Semantic Web (RDF and OWL) with the dominant Web services standards, such as WSDL [67]. The rich service descriptions allowed by OWL are grounded to WSDL-based messages for actual service invocations.

Technically OWL-S is an OWL ontology with three interrelated subontologies [67]:

1. *Profile* that specifies *what a service does*. It is essentially a *functional* description of a service to enable advertising, construction of service requests, and matchmaking. Profile contains a representation the functional properties as IOPE properties as well as structural descriptions. It also contains some non-functional properties of a service. A service

provider can create a service profile for advertisement and a service requester can create a profile to search for available services. After the selection and engagement of service the profile becomes useless; the process model will be the basis of interaction among the requester and provider.

2. *Process model* that describes *how does a service work*. It is a *behavioral* description of a service to enable invocation, enactment, composition, and monitoring of service, and recovery from failures. The process model is not an executable program but a specification of the ways a client may interact with a service.

   In the process model the root class *Process* has three subclasses *Atomic process*, *Composite process* and *Simple process*. Atomic processes can be invoked with one message, from the service requestors perspective execute in one step, and return a message to the requester.

   Composite process are decomposed into simpler processes through control constructs, such as *Sequence*, *If-then-else*, *Split*, *Split-Join*, and *Repeat-Until*. The control structures define the paths that the service requester can perform by sending and receiving appropriate messages.

   Simple processes are not invokable; they are abstractions that can be realized either by an atomic process or a composite process. They can be used to provide perspectives to processes in order to support specific tasks, such as service composition.

   The IOPE/IOPR-properties are specified in the process model ontology, but they can be used – in a simplified form – also in the service profile.

3. *Grounding* specifies *how to use a service* by mapping the constructs of the process model onto concrete message formats and protocols typically expressed in WSDL.

The *operation* in WSDL description maps to the OWL-S concept *atomic process*. Its functionality is characterized by its inputs, outputs, preconditions, and effects (IOPEs). The input is a set of parameters that the client needs to provide for the service provider. The output is a set of variables for which the service provides values. Both the input and output variables are identified by URI:s that need to be defined in an OWL ontology. The preconditions are requirements for the worlds state before the service can be executed and the effects are the changes in the worlds state that the service does when it is executed.

To manage the conditional effects of services, OWL-S versions 1.1 and 1.2 [89] has extended the IOPE-representation with a concept of *results*, and the new representation is called IOPR. A result is a conditional bundling of effects and outputs: if the specified condition holds, then the effects and outputs are as specified. For instance, the ordering of a book could, in the condition that the seller has the book, result in the effect of owning the book and output is a confirmation message; in condition that the seller does not have the book, the result is just the output of a failure message.

In addition to IOPE/IOPRs, a service can be classified by pointing to an OWL ontology that describes all available functionality classes and their subsumption relationships. A service can for example belong to "book selling services" -class which is a sub class of "selling services" -class. OWL thus combines three different ways for specifying the functionality: input/output-function (IO), state transformation (PE), and structural capability descriptions. Due to the possibility for automatic classification of capability descriptions, these approaches can be seen as supporting each other.

There is a lot of research building on top of the OWL-S specification. For service discovery, there has been schemes to combine OWL-S with UDDI registries [90]. For service composition, there has been a lot of derived work. For instance, SHOP2 [98] and OWLS-XPlan [96] are planners that can be used for automatic composition of services that are described with OWL-S. Sirin et al [88] describes how SHOP2 can be used with OWL-S and Klusch et al [59] describes the same for OWLS-XPlan.

One problem in OWL-S relates to the propositional nature of OWL: it does not contain variables that are often required to specify planning operators. The preconditions and effects, for instance, are expressions that describe states and the descriptions often contain variables. From the perspective of OWL, such expressions need to be treated as literals of another language [105, Sect. 5.1], such as SWRL (Semantic Web Rule Language) [49], KIF (Knowledge Interchange Format) [36], or PDDL (Planning Domain Description Language) [38].

5.1.2 WSMO - Web Services Modeling Ontology

WSMO (Web Service Modeling Ontology) is being developed by ESSI Cluster, a group of European research projects in the field of the Semantic Web (SEKT, DIP, KnowledgeWeb, and ASG).

The conceptual background of WSMO is in Web Services Modeling Framework (WSMF) [31] which in turn is based on The Unified Problem-solving Method Development Language (UPML) [32]. The core idea of WSMF is to aim at maximum *de-coupling* between various components in the web of services with the aid of *mediation services* that allow anybody to speak with everybody.

WSMF consists of four main elements:

1. *Ontologies* that define the shared terminology,
2. *Goals* that define the space of problems to be solved (request types),
3. *Web services* that define the elements of the solutions (services), and
4. *Mediators* to solve the interaction problems between the other elements.

The main part of the web services descriptions in WSMO constitutes of preconditions, postconditions, assumptions and effects, which correspond to the inputs, outputs, preconditions and effects in OWL-S respectively.

In addition to ontology, WSMO includes the Web Services Modeling Language (WSML) [116] and the Web Services eXecution environment (WSMX) [24]. There is also another execution environment available for WSMO called Internet Reasoning Service (IRS) [60].

There are five different levels of the WMSL, based on the stacking principle. The most basic one is WSML-Core that is defined as an intersection of logic programming and description logic, thus bridging these two worlds. WSML-Flight is a language that extends WSML-Core with non-monotonic reasoning. WSML-Rule is the extension of WSML-Flight into a full-blown logic programming language. WSML-DL extends WSML-Core to full-blown description logic (comparable to OWL-DL). WSML-Full is the most expressive language including the logic programming and description logic capabilities.

5.1.3 SWSF - Semantic Web Services Framework

Semantic Web Services Framework (SWSF) [2] has taken intuitions and lessons-learned from OWL-S and attempts to build a more comprehensive framework by defining a larger set of concepts. SWSF has been specified by a group that has members from NIST, SRI, Stanford, Toshiba, Southampton Univ, Toronto Univ, MIT, and Bell Labs.

The main parts of SWSF are Semantic Web Services Language (SWSL) [3] and Semantic Web Services Ontology (SWSO) [4]. The essential novelties of SWSF over OWL-S are the following:

1. *The use of the first-order language.* OWL-S is based on OWL DL, a description logic, that is not expressive enough to allow the full axiomatization of the process model. SWSL is based on much more expressive first-order language that facilitates the definition of the actual service ontology (SWSO). It also includes URIs, XML built-in types, XML namespaces, and XML import mechanisms.

2. *More comprehensive process model.* SWSO contains a rich behavioral process model based on a pre-existing standard ontology for manufacturing processes, the Process Specification Language (PSL) [70]. Moreover, SWSF extends PSL with concepts required by Web Services, such as messages, channels, inputs, and outputs.

SWSL has two syntaxes: a simplified presentation syntax and an XML-based serialization syntax RuleML. SWSL has two sublanguages with similar syntax but different semantics:

- *SWSL-FOL* is meant for the specification of service ontologies. It is based on first-order logic with extensions from F-Logic and HiLog.

- *SWSL-Rules* is meant for reasoning about the use of services. It is based on the logic programming paradigm, and combines features from F-Logic, HiLog and Courteous logic programming.

The syntaxes of these sublanguages are similar, but they have different interpretation of same expressions. They are thus semantically incompatible and should not be used in a same document.

SWSO contains a conceptual model of a Web Service and an axiomatization of that model. It has two subforms:

- *FLOWS* (First-Order Logic Ontology for Web Services) that uses SWSL-FOL.

- *ROWS* (Rules Ontology for Web Services) that uses SWSL-Rules. It has been derived from FLOWS through systematic translation. The primary development effort has been on FLOWS.

The part ontologies of FLOWS are *service descriptor* ontology, *process* ontology, and *grounding*. FLOWS provides a first-order axiomatization of the intended semantic of OWL-S, which OWL-S cannot do since OWL is too weak for that and OWL-S does not contain first-order language. Using OWL-S there must be external formalisms to provide the axiomatization of the semantics but FLOWS will be able to use the axioms directly.

FLOWS Core ontology corresponds to the PSL Outercore ontology that includes PSL Core (activities, activity occurrences, timepoints, and objects), subactivity theory, theory of occurrence trees, theory of discrete states, theory of atomic activities, theory of complex activities,

and activity occurrence. To get a flawor of PSL, Figure 5.1 shows the contents of the PSL Core ontology written in CLIF (the Common Logic Interchange Format) [70].

There are number of ontologies that extende the PSL Outercore, dealing with agency, duration of aggregated activities, concurrency, preconditions, effects, resource requirements, resource roles, resource capacities, resource sets, and so on. Together these ontologies present a comprehensive axiomatization for processes.

The logical roots of PSL is in situation calculus which incorporates actions into logics using tree-based models of the execution. The nodes of the trees correspond to atomic activities. An activity has a child activity in the tree, if the child can be executed immediately following the parent activity. The possible execution histories of a specified process model form a branching tree. The structure of the tree can be controlled with constraints on the process model.

FLOWS can act as a foundation for analyzing Web Service models – or any procedural-style description formalism – and comparing them with each other [51].

5.1.4 Other schemes

Triple Space Computing (TSC) [23] is a European research project that attempts to extend the paradigm of *tuple space computing* to area of semantic web services. *Triple space* means specifically the space of RDF triples that can be used to implicit communication between agents using semantically rich representations.

Task computing project [35] attempts to develop new ways for users to interact with devices and services in their environment. Their goal is to allow users to focus on tasks they are performing with devices and not be distracted by the questions of interaction with devices. This is closely related to the goal-service distinction mentioned earlier. The idea is to approach this distinction from the perspective of user tasks (as in hierarchical task network planning), not from the perspective of goals (as in classical, state-based planners).

The project is being carried out in collaboration between Fujitsu and MindSwap project in University of Maryland [68]. It is developing a Task Computing Environment TCE consisting of task computing clients (TCCs), semantically described services (SDSs), semantic service discovery mechanisms (SSDMs), and service controls. The framework tries to use standard technologies such as Web Services, UPnP, and OWL-S.

5.2 Service composition

Web Service composition addresses the situation when a request cannot be satisfied by a single pre-existing service but can be satisfied with a suitable combination of them. With suitable representations and reasoning capabilities, the goal of a service requestor can be achieved by automatically selecting and combining available Web Services at the execution-time.

In the following we attempt to give an overview of the composition technology based on existing surveys and overviews presented by Peer [78], Hull and Su [51], and Sirin [87].

5.2.1 Dimensions of the composition problem

Within the artificial intelligence community, the study of action composition problems has a long history. In the following, we take this field – known as *AI Planning* – as the reference point, and describe the composition problem from the perspective of planning research. The formulations

```
Primitive relations:            Functions:             Constants:      Defined relations:
    (object ?x)                    (beginof ?occ)         inf+            (between ?t1 ?t2 ?t3)
    (activity ?a)                  (endof ?occ)           inf-            (beforeEq ?t1 ?t2)
    (activity_occurrence ?occ)                                            (betweenEq ?t1 ?t2 ?t3)
    (timepoint ?t)                                                        (exists_at ?x ?t)
    (before ?t1 ?t2)                                                      (is_occurring_at ?occ ?t)
    (occurrence_of ?occ ?a)
    (participates_in ?x ?occ ?t)

Axioms 1-4: The before relation holds between timepoints, and is a total ordering, irreflexive, and transitive.
        (forall (?t1 ?t2)
            (if (before ?t1 ?t2)
                (and (timepoint ?t1) (timepoint ?t2))))
        (forall (?t1 ?t2)
            (if (and (timepoint ?t1) (timepoint ?t2))
                (or  (= ?t1 ?t2) (before ?t1 ?t2) (before ?t2 ?t1))))
        (forall (?t1) (not (before ?t1 ?t1)))
        (forall (?t1 ?t2 ?t3)
            (if (and (before ?t1 ?t2) (before ?t2 ?t3))
                (before ?t1 ?t3)))
Axioms 5-6: All timepoints are between inf- and inf+ (except inf- and inf+ themselves)
        (forall (?t)
            (if (and (timepoint ?t) (not (= ?t inf-)))
                (before inf- ?t)))
        (forall (?t)
            (if (and (timepoint ?t) (not (= ?t inf+)))
                (before ?t inf+)))
Axioms 7-8: Given any timepoint t other than inf-, there is a timepoint between inf- and t, and same for t andd inf+.
        (forall (?t)
            (if (and (timepoint ?t) (not (= ?t inf-)))
                (exists (?u) (between inf- ?u ?t))))
        (forall (?t)
            (if (and (timepoint ?t) (not (= ?t inf+)))
                (exists (?u) (between ?t ?u inf+))))
Axioms 9-10: Everything is either an activity, activity occurrence, timepoint, or object, but no any two of these.
        (forall (?x)
            (or (activity ?x) (activity_occurrence ?x) (timepoint ?x) (object ?x)))
        (forall (?x)
            (and (if (activity ?x) (not (or (activity_occurrence ?x) (object ?x) (timepoint ?x))))
                 (if (activity_occurrence ?x) (not (or (object ?x) (timepoint ?x))))
                 (if (object ?x) (not (timepoint ?x)))))
Axioms 11-13: The occurrence relation holds between activities and occurrences, and must hold for every occurrence with a unique activity.
        (forall (?a ?occ)
            (if (occurrence_of ?occ ?a)
                (and (activity ?a) (activity_occurrence ?occ))))
        (forall (?occ)
            (if (activity_occurrence ?occ)
                (exists (?a) (and (activity ?a) (occurrence_of ?occ ?a)))))
        (forall (?occ ?a1 ?a2)
            (if (and (occurrence_of ?occ ?a1) (occurrence_of ?occ ?a2))
                (= ?a1 ?a2)))
Axioms 14-15: The begin and end of an activity occurrence or object are timepoints, and begin is before or equal the end.
        (forall (?a ?x)
            (if (or (occurrence_of ?x ?a) (object ?x))
                (and (timepoint (beginof ?x)) (timepoint (endof ?x)))))
        (forall (?x)
           (if (or (activity_occurrence ?x) (object ?x))
               (beforeEq (beginof ?x) (endof ?x))))
Axioms 16-17: The participates_in holds between objects, activity occurrences, and timepoints, and the activity must be occurring.
        (forall (?x ?occ ?t)
            (if (participates_in ?x ?occ ?t)
                (and (object ?x) (activity_occurrence ?occ) (timepoint ?t))))
        (forall (?x ?occ ?t)
            (if (participates_in ?x ?occ ?t)
                (and (exists_at ?x ?t) (is_occurring_at ?occ ?t))))
Definitions 1-5: Between, before, betweenEq, and beforeEq.
        (forall (?t1 ?t2 ?t) (iff (between ?t1 ?t2 ?t3)
            (and (before ?t1 ?t2) (before ?t2 ?t3))))
        (forall (?t1 ?t2)
            (iff (beforeEq ?t1 ?t2)
                 (and (timepoint ?t1) (timepoint ?t2) (or (before ?t1 ?t2) (= ?t1 ?t2)))))
        (forall (?t1 ?t2 ?t3)
            (iff (betweenEq ?t1 ?t2 ?t3)
                 (and (beforeEq ?t1 ?t2) (beforeEq ?t2 ?t3))))
        (forall (?x ?t)
            (iff (exists_at ?x ?t)
                 (and (object ?x) (betweenEq (beginof ?x) ?t (endof ?x)))))
        (forall (?occ ?t)
            (iff (is_occurring_at ?occ ?t)
                 (betweenEq (beginof ?occ) ?t (endof ?occ))))
```

Figure 5.1: PSL Core ontology

of the approaches originating from other fields – such as automata theory or operations research – will be contrasted with those of AI Planning.

The planning problem does not have an unambiguous definition. One reason for the existence of many different approaches – apart from the attempt to improve the performance of planning systems – is that they are actually solving slightly different version of the planning problem. This makes the comparison of planning techniques a challenging problem. Some dimensions of differences are the following:

- *Formalization of the domain.* Approaches within AI Planning that differ on the rigour and expressivity range from logical approaches of situation calculus, event calculus, and action theories to more STRIPS notation and its extensions, such as ADL and PDDL. Outside AI Planning, there are approaches based on automata theory and Petri nets.

- *Representation of goals.* A goal is usually expressed either as a world state to achieve or a high-level activity to perform. However, the goal of compositions can also be the realization of a behavior given as a state transition system. In manual compositions approaches (WS-BPEL and so on) goals are typically not represented at all [91].

- *Representation of plans.* The simple notion of a plan as a sequence of activities is widely used in research but clearly inadequate in many practical domains. Extensions to that simple model range from parallel activities to conditional plans, and even the possibility of plans containing loops have been discussed. The are approaches that produce situated plans (or policies) that provide a mapping from all states to actions to take. The workflow-based orchestration languages – such as WS-BPEL – offer a rich collection of control structures for compositions as well as mechanisms for handling faults and exceptions.

- *Control knowledge.* Can planning be constrained or directed with control knowledge, and if so, what should the control knowledge be like? It is difficult to include control knowledge to classical planning. However, the hierarchical task network planning requires perhaps too much control.

- *The role of non-functional properties.* There are often some implicit preferences in plan generation algorithms, such as the minimization of the length of the longest operation sequence in a plan. These kinds of criteria may be largely irrelevant for typical requestors of Web Services. In practice, the preferences and constraints on plans need to be formulated in terms of the desired non-functional properties of services, such as costs, quality, availability, and security.

- *Single use versus multiple uses.* The compositions created manually are usually meant for multiple uses, while the approaches based on AI Planning create a plan for a single use. There are also automatic composition approaches that create compositions for multiple uses.

- *Peer-to-peer versus centralized composition.* Does the creation of a composition mean also a creation of a coordinating service that will manage the execution of a service? Is the plan generation approach closer to orchestration or choreography?

5.2.2 Approaches within AI Planning

Within AI Planning community, speaking informally, *planning* is seen as a reasoning task whose goal is to solve a *planning problem*. Planning problem is usually represented as the gap between an *initial state* – the current state of the agent – and a *goal state* – the state that the agent wants to hold. The elements of solutions are represented as a *set of available activities*. The solution to a planning problem is a *plan* that is a *set of activities* subject to *a set of ordering constraints*. Web Services can now be seen as activities and the planning can be used to create a composite service (plan) that satisfies the goals of a service requestor.

An influential early formulation of planning problem is the STRIPS notation [33], developed in the 1970's for a robot system called "Shakey". The planning operators is STRIPS were described by specifying a *precondition*, an *add*-list and a *delete*-list, all represented as conjunctions of atoms. The meaning of such operator descriptions is that an operation is applicable if the precondition is satisfied by the current world state. After the execution of the operation, the atoms in the *add*-list will be added to the world state and those in the *delete*-list will be removed. For example, the transportation operator `move(A, L1, L2)`, where `A` is an object and `L1` and `L2` locations, should have the precondition `at(A, L1)`, the add-list consisting of `at(A, L2)` and the delete-list consisting of `at(A, L1)`. The exact logical semantics of STRIPS has been a subject of debate for long time.

The STRIPS notation is less powerful that may seem at the first glance. No arithmetic operations or numerical comparisons are allowed, and it is not possible to represent constraints that say that if a truck moves from one location to another, all the goods contained in it will also move.

AI Planning research is based on the implicit quality requirement that any plan produced should *guarantee the achievement of the goal state* from the initial state of given problem. Many of the complications of planning algorithms stem from the need to satisfy this rather strict requirement, e.g., by taking care during the plan generation that all possible conflicts are ruled out. In contrast, when people are making practical plans, the resolution of unlikely problems is often left to the execution time. This idea is used in dynamic planning approaches in which execution and planning are interleaved. Planning could be also based on a probabilistic model, in which case planning can be seen as policy generation for Markov decision processes. Planning determines actions to maximize the expected utility of an agent. Both of these approaches have their advantages and disadvantages. The probabilistic model is flexible and works in nondeterministic environments. However, often agents want to have some form of guarantee that the plan will achieve the goals before starting the possibly costly or irreversible execution.

Planning as state-space search

Given the STRIPS-type formulation of a planning problem, it is a straightforward idea to organize the problem solving as a search in a state-space. Nevertheless, advanced state-space planners can have a surprisingly good performance, at least in some domains.

In *forward state search*, planning starts from the situation where the search-tree contains only one node, the initial state s_0 that is a set of all the initial conditions. If the search-tree contains a node s that contains all goal conditions S_G then the path to s, that is, the sequence of operations from s_0 to s, can be returned as a solution for the planning problem. If such node does not exist, the search-tree can be expanded by picking any node s (a non-deterministic choice) whose all successors have not yet been generated. If a planning operator of type t and

a combination of variable values for t can be picked (a non-deterministic choice) so that no successor of s has the identical type and value combination, then t can be instantiated as a, an edge between s and a new state s'. State s' contains the same conditions than s, except for those that a removes and adds to it. If all the non-deterministic choices have been tried, and no solution has been found, the problem is unsolvable.

An alternative way is to organized planning as *backward state search*. The planning starts from the goal state and operators are used to transform it to the initial state.

It is possible to implement the state-space search with any search algorithm. Simple search algorithms, however, are not able cope with the vast search spaces inherent in planning problems. There is a need for advanced methods that (1) reduce the search space by pruning away unfruitful branches of the search tree and making that as early as possible, and (2) direct the search to most promising branches of the search tree.

To direction of search is usually based on *heuristic functions* that estimate the fruitfulness of alternative actions that the planner can choose from. The early versions of heuristic functions were as simple as the number of open flaws in a state. However, more advanced heuristics have been developed.

In HSP (Heuristic Search Planner) [14] the heuristic evaluation of a state is based on the estimated cost of achieving the subgoals that are still open in a state. Assuming that the subgoals independent (which is not generally true), the cost of a state is the sum of costs of its open subgoals. The cost of each subgoals is computed by solving a relaxed (and computationally much easier) version of the planning problem where negative effects of operations are ignored to find out how many steps are required before a subgoal is achieved. Empirical results from planning system competitions shows the that heuristic power of assumption of subgoal independence and distance of subgoal achievement in relaxed problem [17]. More recent work in this line is HSP2 [16] that uses best-first search instead of hill-climbing used in HSP, and HSPr [15] that is based on a combination of backward state search and forward generation of heuristics in an attempt to avoid the effort of regenerating the heuristic function at each step of the search. HSPr is substantially faster than HSP in some domains but slower in others.

FF (Fast Forward) [48] is another planner based on forward state search guided by a heuristic estimates based on a relaxed problem. The heuristic values are based on a generation of *planning graph* (see next subsection). The cost of a state is the number of actions to reach the goal in the planning graph. In the relaxation of the problem the negative effects are ignored but the positive interactions between subgoals are taken into account. FF used a search strategy called *enforced hill-climbing* where at each search step all successors nodes are evaluated, possibly even using breath-first search, until a strictly better evaluation is found. This allows the search to escape local minimums. FF was one of the winners of the ICP-2000 planning competition.

FF has been extended in Metric-FF to handle numerical variables, constraints, and effects as specified in PDDL 2.1 level 2. The preconditions can contain numerical constraints such as $amount > 20$ and arithmetic operations in effects, such as $amount- = 10$.

The performance improvements in state-space search come from better heuristics derived from a problem relaxed by ignoring the negative effects of operations. It is not surprising that such planners do not perform particularly well in problems characterized by interactions where one operation deletes the preconditions or effects of another. Typical examples are the classical blocks world problems and different kinds of logical puzzles. It is quite possible that these characteristics are not typical in practical Web Service composition problems.

One of the limitations of state-space search is that the structure of resulting plans is a simple sequence of operations. It seems difficult to extend state-space search to produce more complex plan structures.

Planning graphs

Blum and Furst [12, 13] introduced a planning framework called GRAPHPLAN that is based on the generation of a *planning graph*, an intermediate data structure to reduce the amount of search needed to find the solution.

Planning graph is a *directed graph* consisting of *alternating levels* of *proposition nodes* and *action nodes*. The first level is a proposition level which consists one node for each initial condition of the planning problem. The second level consists of all those action instances whose preconditions are satisfied at the previous level. The third level is a proposition level consisting of all propositions at the first level and all the effects of all actions at the second level. The planning graph is generated from the first level forward until two identical successive proposition levels are encountered. This will always occur and hence the generation will terminate.

The graph can be annotated with additional information that tells within each level which propositions or actions are mutually incompatible. The mutual exclusion (or mutex) relations hold between two propositions at the same level if one is a negation of the other, or if all possible actions to establish one are mutually exclusive with all possible actions to achieve the other. The mutual exclusion holds between two actions, if one negates an effect of another (*inconsistent effect*), if one of the effects of one is a negation of one of the preconditions of the other (*inference*), or when one of the precondition of one is the negation of a precondition of the other (*competing needs*).

Planning graphs have the property that the set of propositions and actions increases monotonically when new levels are expanded while the set of mutexes decreases monotonically. This matches with the intuition that the longer the execution of a plan is allowed to take, the more possibilities there are to find different kinds of plans. Planning graphs can be computed in polynomial to the size of the planning problem determined by the size of the initial state, size of the set of planning operators, maximum size of a precondition, and number of time points.

Now the plan generation in GRAPHPLAN works as follows. A planning graph is grown until a proposition level containing all goal propositions with no mutexes between them. This is a necessary but not sufficient condition for plan existence. A plan extraction is started as a backward search in the planning graph. A set of actions from the last action level that together have all the goal propositions as effects are included in the plan. The same are repeated in the preceding action level for the preconditions of the included actions and so on. In case of failure, GRAPHPLAN backtracks and tries different action sets. If no plan cannot be find, the planning graph is expanded until another promising proposition level is reached and plan extraction is tried again.

Plan extracted from a planning graph is thus not a path in the graph as in a state-space planner but more like a flow in a network flow sense. The resulting plans are also not simple action sequences, since GRAPHPLAN allows for some amount of concurrency. The actions belonging to the same level of planning graph can be executed concurrently. Other advantages of GRAPHPLAN are the soundness, completeness, generation of shortest plans, and termination on unsolvable problems.

While the original GRAPHPLAN was limited to the pure STRIPS notation, subsequent re-

search has expanded the capabilities of planning graphs to many directions. In IPP planner the planning graph approach has been extended with the ability to handle conditional and universally quantified effects [61], and in SENSORY GRAPHPLAN the imcomplete knowledge of the initial state. The performance of Graphplan is decreased if the problem definition contains irrelevant planning operators. This problem was address among other's in STAN planner [62] that performs a static analysis of the planning problem before the expansion of the planning graph to reduce the set of operators considered.

GRAPHPLAN – and planning graphs in general – has had an enormous impact to different planning approaches as a way to generated heuristic and estimates to improve the performance of planners.

Planning as propositional satisfiability

Kautz and Selman [56] presented a way to formulate a planning problem as a set of axioms in such a way that any model of axioms correspond to a valid plan. This approach has been motivated by the development of high-performance generic satisfiability solvers.

The planning problem presented in a STRIPS-like notation can be compiled into the following manner. In the blocks world domain, the planning problem of reaching the goal $on(B, A)$ from initial state $on(B, A) \wedge on(B, Table)$ can be expressed as:

$$on(A, B, 1) \wedge on(B, Table, 1) \wedge clear(A, 1) \wedge on(B, A, 3)$$

In this formulation, each predicate has one additional argument representing the (abstract) time step. The example formulation tries to find a solution to the problem by time step 3.

A move operation in blocks world domain would be represented in STRIPS as follows:

```
MOVE(x, y, z)
   PRE: CLEAR(x), ON(x, y), CLEAR(z)
   ADD: CLEAR(y), ON(x, z)
   DEL: ON(x, y), CLEAR(z)
```

In the most straightforward encoding (so called linear encoding [56]) to a satisfiability problem this can be formalized as:

$$\begin{aligned} \forall x, y, z, i. move(x, y, z, i) \subset \; & clear(x, i) \wedge on(x, y, i)) \wedge clear(z, i) \wedge \\ & clear(y, i + 1) \wedge on(x, z, i + 1) \wedge \\ & \neg on(x, y, i + 1) \wedge \neg clear(z, i + 1) \end{aligned}$$

The preconditions and effects would be used in a symmetric manner: the occurrence of an activity at time step i implies its preconditions at step i and the positive effects at $i + 1$ and the negation of negative effects at $i + 1$.

These axioms are instantiated with all variable value combinations (the domains need to be finite) and are treated as boolean variables in fast propositional search by satisfiability solvers.

If frame axioms and axioms that rule out parallel actions are added, the satisfiability solver can produce model that is the solution to the planning problem:

$$\{move(A, B, Table, 1), move(B, Table, A, 2)\}$$

There are many different encoding developed for satisfiability planning: state-based encoding, parallelized encodings, causal encoding, etc. The lengths and variable counts of these encodings are different which influences the performance of the planning systems.

There are a number of satisfiability solvers available. In [57] several systems are mentioned: Sato, satz, zChaff, Jerusat, Siege, and Minisat [20]. The basic approaches of the solver are systematic and stochastic search. The stochastic techniques appeared superior in the 1990's while the systematic search is responsible for the recent performance improvements.

The planning as satisfiability approach has resulted in a series of high-performance planners: SatPlan [99], MEDIC, BlackBox, MaxPlan, SatPlan2004, and SatPlan2006.

The early SatPlan at 1992 was essentially a set of conventions to encoding STRIPS-problems into satisfiability problems. A significant increase in performance became in 1996 when first parallel encoding were developed, allowing non-interfering actions to occur at the same time step. The BlackBox system borrowed the mutex propagation techniques from GraphPlan, resulting in a better encoding. In IPC 1998 the performance of BlackBox was among the best systems; others were mostly based on GraphPlan. However, in IPC 2000 BlackBox did not perform well; the winning system were based on GraphPlan and heuristic search. The increase of the power of generic satisfiability solvers, and possibly the more realistic and harder problems used in IPC 2004 made BlackBox a clear winner of the competition [57].

Partial-order planning

Partial order planning is based on the idea of planning as search in a plan space. The nodes of the search graph are thus *partial representations of plans* and the edges between nodes represent *plan refinements*. The search can be terminated when a node is found containing a fully specified plan that achieves the goal conditions from the initial state. The intermediate nodes in the search tree contain plans that have *flaws* – threats and open conditions – that need to be resolved, and each refinement edge in the graph means a resolution of a flaw.

Initially the goal conditions belong to the set of open conditions. The plan can be refined by selecting one of the open conditions c and an action a that establishes c as its effect. The preconditions of a are then added to flaws. Now, in order to keep track of the rationale of the refinement, a causal link is created that tells that a was introduced to establish condition c. If at any time another action is introduced that would negate c, it is called a threat to the causal link; the resolution is to add ordering constraints to the plan that force the new action to occur either before or after the causal link is active. When all flaws of a plan are resolved, it is a solution to the planning problem.

The plans produced consist of a set of partially ordered actions. It contains only the necessary ordering constraints between activities, with the exception of those that were introduced to resolve threats to causal links. While the strict interpretation of the plan is that any linearization of the plan is guaranteed to work, in practice it allows parallelism.

Within AI Planning a number of partial-order planners were created. The best known are NOAH, NONLIN, TWEAK, SNLP, and UCPOP. However, when new planners emerged in 1990's – Graphplan, SatPlan, and fast heuristic state space planners – partial order planners could not compete with them and were out of fashion for a long time. In recent years they have regained some interest when the ideas of Graphplan and heuristic planners were incorporated in partial-order planning in systems such as RePOP and VHPOP.

The major advantages of partial-order planning is the openness and flexibility of the framework, and the ability to produce compact plans.

Hierarchical task network planning

The origins of *hierarchical task network planning* are in ABSTRIPS (Abstraction-Based STRIPS) system from 1973. In addition to operations (*primitive tasks*) there are in HTN planning also *methods* that describe ways to decompose given tasks into a set of subtasks. These methods are hand-coded by humans and they can be regarded as control knowledge or domain knowledge.

There has been recent interest in specific type of HTN planning, called *ordered task decomposition* planning: actions are included into the plan in the same order in which they will be executed. This allows the planner to keep better track of the changes to the state of the world, and therefore more complex preconditions and effects can be used.

Simple Hierarchical Ordered Planner SHOP2 [98] is a hierarchical planner that creates the plans in the same order that they will be executed. This simplifies the implementation of the planner at the expense of performance but it has an important advantage: the information about the state of the world is better and more complex preconditions can be evaluated (even including external function calls).

Evren Sirin [87] presents HTN-DL formalism that combines HTN planning with terminological reasoning, enabling the automatic composition of Web Services described with OWL-S. This approach requires further development of optimization techniques for DL reasoning to deal with large number of individuals.

OWLS-XPlan [96] is a planning graphs based planner that includes ideas from hierarchical task network planners. It uses PDDL (Planning Domain Description Language) [38, 37] as its input language. With a mapping from OWL to PDDL the planner has enabled the use of XPlan in service composition reasoning. [59]

5.2.3 Composition of state transition systems

One approach is to model the behavior of a service in the Web as a state transition system. It defines a set of states and transitions are atomic actions of the service. A human or machine agent can use the service by sending appropiate messages. The description thus specifies a interactive mechanisms and not just a state transformation, as in the planning approaches describe above. This kind of service are closer to human-machine interaction than plain Web Services that are basically stateless.

Figure 5.2 shows a simple service that acts as a music store. The user can send three kinds of messages (init, search, listen, cart, and buy) to carry out a desired interaction with the service.

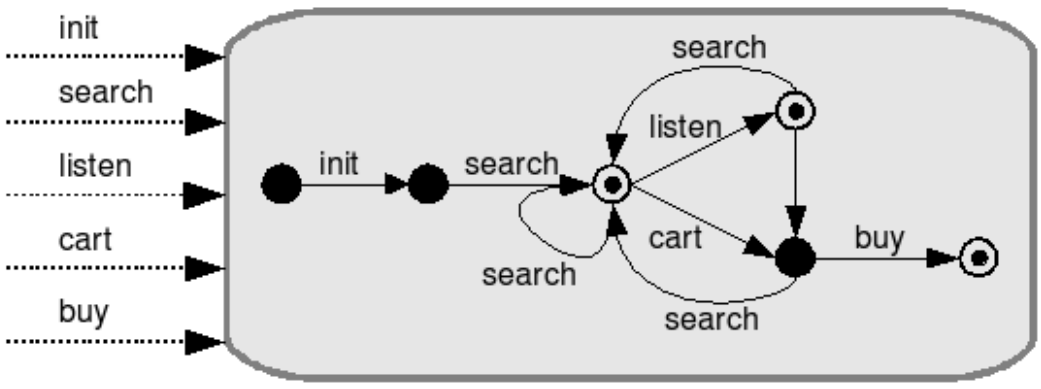

Figure 5.2: A music store service as a state transition system

The Colombo framework

Berardi et al [7, 9, 6] present an approach where the goal is to automatically produce reusable composite services out of existing services in the Web. The service composition problem is formulated in a completely different way than in AI Planning. It also differs from the Web Services philosophy in the focus is specifically on stateful services. However, the model is potentially applicable for modeling complex interactions between automated agents.

This approach is not based on a functional description of service, e.g. through their pre-conditions and effects. Rather, it is based on a *behavioral* description in which the internal actions and message passing actions of a service are represented using a *finite state transition system* [6]. After a service is enacted, the service requestor interacts with it by repeatedly sending and receiving messages, until a terminating situation is reached. Each service instance has a *local store* that captures the parameter values of incoming messages and output values of internal actions performed, and is used to produce the parameter values for outgoing messages and giving the input parameters to internal actions. A service can *access and modify the world state* that is modeled as a relational database.

The service composition tasks is specified through a *goal service*, a representation of the desired behavior as a transition system. The target is to build a *delegator* – also called a *mediator* e.g. in [9, 51] – that that interacts with existing services so that its overall behavior accurately simulates that of the goal service [6].

In a simple case, the delegator can be structurally similar finite state transition system than the goal service. The transitions in the delegator are just labeled with – and at the execution time delegated to – corresponding transitions in existing services. However, in more complex cases the delegator can be structurally different from the goal service.

The delegator can be be constructed by reducing the problem of composition existence to satisfiability of a concept in into description logic $ALCQ_{reg}$. This can be done in EXPTIME [8].

Planning as model checking

Cimatti, Giunchiglia, Pistero and Traverso [21, 40, 79] have proposed another approach of modeling planning problems as state transition systems. A verification technique called *model checking* – more precisely *symbolic model checking* – is used to create situated plans.

The advantages of the approach are its solve planning problems that can contain nondeterminism, extended goals, and partial observability. It is also possible to devise efficient algorithm to deal with large planning problems.

A planning domain can be considered as $D = (S, A, R)$ where S is a finite set of states, A a finite set of actions, and $R : S \times A \to 2^S$ is a nondeterministic state transition function. Model checking generates situation plans, or policies, $\pi = \{\langle s, a\rangle : s \in S, a \in A\}$ consisting of state-action pairs. The plans are executed in an reactive loop that repeatedly senses the state, selects ab appropriate action, and executes it:

$$\begin{array}{c}\text{while } s \in \{s : \langle s, a\rangle \in \pi\} \text{ do} \\ a \text{ such that } \langle s, a\rangle \in \pi\end{array}$$

The plans can be essentially generated as a side-product of verification that there exist a branch in a tree of states originating from the initial state where the goal conditions will be eventually true.

Plans generated with model checking are more general and robust than those in classical AI Planning. For example, plan can is some cases recover from unexpected action outcomes. plans can also be *goal preserving* which means that an agent can continue acting after achieving the goal without abandoning the set of goal states.

In the nondeterministic case there are different types of guarantees that a plan can give to the achievement of a goal. A *weak* plan may achieve the goal and a *strong* plan guarantees its achievement. A plan may also be *strong cyclic* if it guarantees the achievement of the goal if the execution terminates (which is guaranteed not to be impossible). Algorithm for producing different kinds of plans have been presented in the literature. The practical algorithms are based on *symbolic model checking* that allows for a less redundant representation of states and therefore lead to more efficient planners.

Planners based on model checking and that work in nondeterministic domains include MBP (Model Based Planner) [11] and UMOP (Universal Multi-Agent OBDD-based Planner) [54].

In subsequent work these planning techniques are applied to Web Service composition problem. In [80] service composition in WS-BPEL is out of component services described as WS-BPEL process. In [66] this composition techniques are applied to create a composite process out of Amazon Web Services and the e-payment services of MPS bank.

5.2.4 Discussion

All the planners were too inefficient for any practical problems until 1990's when the increases in the available main memory of computers made it possible to use significantly more efficient solution approaches. A planning problem (usually specified in STRIPS-like format), is grounded by instantiating all formulas with all value combinations of all their variables (this assumes that domains are finite). Then the solution is searched using fast propositional search methods. This has dramatically expanded the scope of planning problems that can be solved. The original implementations of this approach were graph-based GraphPlan and SatPlan that encoded a planning problem to a boolean satisfiability problem.

While the planning algorithms are gradually becoming more efficient and more practical, there are some serious problems in the application of planners to service composition. The use of planners in this phase is still a *technological idea without any empirical studies of applications to large scale service composition tasks*. In fact, there is not even enough practical experience to say how the algorithms would behave in large service composition problems. In addition, there is no idea of how *complex or tedious it would be to develop the necessary representations to support planners*. Can services be described reliably enough in notation that planners understand? A more basic problem is that the current planning algorithms work on the assumption that the *only source of changes in the relevant parts of the world is the execution of the plan*. However, it is quite possible that outside changes to the world will make the plan break before its execution is completed. How serious a problem this is in practice, is an empirical question. It is still possible that a plan can be regarded as an approximation of the behavior and exceptional outside events can be taken care of with dynamic planning (reactive planning or replanning), or a user intervention. Dynamic planning, however, can not work under strict quality guarantees and a possibility of failure must be accepted. The *user interfaces that could support appropriate user interactions* still require new ideas and more research.

5.3 Mediation

The primary goal of semantic web services is to make it possible for a software agent to invoke services that it was not specially developed to invoke. It is still assumed, that the agent shares the same domain model (ontology) as the services that it uses [18]. In practice, most of the complications in information systems integration in general result from independently developed and thus incompatible domain models. The semantic web services would provide little help for the system integration work if they would not address the need to cope with heterogeneous ontologies. Thus it is essential to try to develop and use mappings between ontologies.

The incompatibility of a user agent and a web service can occur on two levels:

- on the data level where the syntax and models may differ
- on the process level where the sequence and functionality of operations may differ

If automatic planning is used for service composition, overcoming the differences on the data level may overcome the differences also on the process level [19].

The data level mediation has been studied as long as there has been data bases. The usual information system integration scenario is to query a set of databases through a single global schema. The schema is typically expressed as a query over the source schemas. This is called the global as view (GAV) architecture. In the semantic web such global schema or ontology is impossible to maintain, because there is no central authority that would keep track of the mapping between it and the various other ontologies that exists throughout the web. To overcome the requirement of a single definition of the global schema, it has been proposed that the mappings between the global schema and other schemas (local schemas) could be expressed so that each local schema is a query over the global schema. This is called local as view (LAV) architecture. In this setup, the mappings could be provided by various independent providers. A disadvantage is that answering queries over the global schema is the same as answering queries using views, which is a hard problem to solve [46]. The most advanced way of providing interoperability between ontologies in a dynamic environment may be provided by so called peer data management systems, in which the ontologies are mapped to each other to form a network of ontologies. Data can be transferred in such a network by composing and sequencing the various mappings [47].

Finding the correspondences between the elements of two or more schemas automatically is an active area of research, but the current approaches can only find simple one to one mappings between the elements and human operator is needed to validate and refine the results before their use [81].

Mediators are a central element in the WSMO framework in which they connect goals, services and ontologies together. Such mediators can be simple assertions that a certain goal may be achieved by a certain service, but they may optionally refer to a web service that does some mediation. Such mediation service is a black box from the WSMO point of view. This notion of mediator is close to the ”middle agents” that are used in the industry to integrate applications. Such mediators are developed in common procedural programming languages and the development is thus as laborious as any software development. In fact, any software module that provides some interface and uses other interfaces can be seen as a mediator between the interfaces. In general, mediators are thus simply software modules (or services) that provide some desired interface to services that do not directly implement that interface.

Transferring data bidirectionally between schemas is often impossible. For example when one schema represents a street address as a single string and other schema splits the address into various attributes such as the zip code and country, it may be impossible to automatically transfer the data from the monolith string to the more fine grained schema. Often the data stored in one schema simply does not exist in the other. Mediation between independently developed ontologies can thus usually succeed only partially. Mediation is only a way to alleviate the lack of coordinated ontology development, which is the ultimate but hard to achieve solution to system interoperability.

5.4 Ad hoc reasoning

There is a general understanding among the Semantic Web research community that the logic programming is a natural way to process the data represented as RDF triples. The background of the community in the knowledge representation research also makes that paradigm familiar to its members. However, this should not obscure the fact that rules are only one way to process the data and many practical reasoners, e.g. planners, may be implemented with other techniques. There can also be a range of less ambitious programs that simply use RDF data in their operations without attempting to do any advanced reasoning.

If rule paradigm is contrasted with the description logics, description logics are meant for reasoning about types while rules are more natural for *reasoning about instances*. Rules contain variables that are bound to individual object instances or properties. The condition of the firing of a rule may be that a complex relation between the properties of different individuals holds.

In the logic programming community the focus has been on languages with *default negation*: if p cannot be proved, it is assumed that its negation holds. Default negation makes the representation of the domain knowledge much easier when only the positive facts need to be taken into account. If only the classical negation is used (as in the first-order language), the volume of negative facts (what does not hold in the world) can obfuscate the representation task.

As noted above, languages with default negation are difficult to semantically combine with OWL which is based on first-order language and thus on classical negation. It is thus possible that the rule languages would sit more naturally on top of the RDF layer that on OWL in the Semantic Web language stack.

Prolog is the logic programming language where default negation is implemented as "negation as failure". In Prolog the evaluation of sentences can be controlled by the programmer by ordering the rules and the formulas in the rules. In addition, the programmer can also cut the further search of the failure. These mechanisms can be used to improve the performance of a program. However, they also mean that the program has an operational semantics in addition to declarative semantics.

Negation as failure can also be interpreted epistemically, as in the stable model semantics of answer set programming. In this interpretation not(p) means literally that p is not known or not believed. In answer set programming a program can have multiple stable models. The following program:

```
a :- not b.
b :- not a.
c :- a.
```

has two different answer sets a , c and b . The advantage of answer set programming over Prolog is its more declarative nature, since the order of the rules is not meaningful. It is therefore possible to combine rules from different sources into the same program and still produce meaningful results.

The best known implementations for answer set programming are Smodels, an implementation of the stable model semantics for logic programs [92] from Teknillinen korkeakoulu and Dlv, a disjunctive datalog system [94] from TU Vienna.

If multiple models are returned, there is naturally a problem of choosing between the alternatives. This can be done with priorities or preferences after the models have been computed. There is a version of Smodels called Psmodels that can be used to compute preferred answer sets under the ordered disjunction semantics.

A different approach is used in *courteous logic programming* that maintains a consistent and unique answer set during the evaluation since it uses a prioritized conflict handling for rules. That is, it makes the choice between alternative models as soon as the multiple models appear. The formalism has been developed especially business rules in mind. SweetRules [85] is an Java-implementation of courteous logic programming.

Some of the influences of rule languages in the Semantic Web come from F-Logic (or frame logic) that is a logical language designed to represent complex objects in an object-oriented fashion. F-logic includes object-oriented concepts such as object identity, complex objects, inheritance, polymorphism, query methods, and encapsulation. One influence is HiLog that has higher-order and meta-programming features with computationally tractable first-order semantics.

Rule language specifications that are oriented towards the Semantic Web are

- Rule Markup Language (RuleML) [83]: An XML serialization syntax for rules, to promote rule interchange across rule systems, and
- Semantic Web Rule Language (SWRL) [49]: a rule language that combines RuleML and OWL

Chapter 6

Prototypes and case studies

Currently, there are no real-world working applications of Semantic Web Services. However, a recent EU project *Data, Information, and Process Integration with Semantic Web Services* (DIP) [84] developed several case studies of the semantic web services were applied to real world problems. In the following we describe some of these studies.

6.1 DIP case studies

6.1.1 The British Telecom case study

DIP project carried out a case study in which the British Telecom evaluated SWS technologies for managing the services that they provide for their customers [82]. The services constitute of ICT products ranging from desktop applications to the network infrastructure. They are provided by numerous subcontractors and their usage is constrained with complicated rules such as location and current usage. Each sub contractor has their own product catalog system that has its own ways to define constraints for the products. Each provider also has it's own order management system. The result was a prototype system called Contract Catalogue Prototype whose aims were:

- Representing a Contract Catalog as an ontology provides advantages over standard document or spreadsheet based approaches
- Integration of third party suppliers products (and associated rules) into the catalog can be done more efficiently with use of ontologies and Semantic Mediation.
- Using Semantic Web Services for functionality associated with products (such as ordering and querying) provides increased automation and efficiency over standard Web Services and other methods for B2B integration.

The prototype can classify product bundles, answer queries about the bundles, check the consistency of the bundles and produce a plan how the bundle can be ordered using the web services of the subcontractors. Mediation between incompatible product catalogs are defined as ontology mappings that are used to automatically mediate individual messages between the systems [82].

It is clear that if the product catalog is expressed in some structured format, it can be queried in ways that are not possible when using some text documents or spreadsheets. The

mediation between product catalogs is not well described in the prototype documentation. Also it seems that there is no effort on allowing the sub contractors to run their consistency checks on the product bundles. The constraints are only run in a single catalog instance. It is not evident whether the web service discovery and composition is easier with WSMO goal and service descriptions than by some more conventional service catalogue implementation approach. The web services and goals have to be compatible with each other after all. Thus the services can not be developed independently.

6.1.2 The social care eGovernment case study

DIP case study called Change of Circumstances Prototype was carried in the area of eGovernment [26]. In the case study semantic web services are applied in a situation in which two separate databases are used to manage equipment and services that are provided for citizens. One of the databases keeps track of the equipment like roller chairs while the other one keeps track of services like home cleaning. Total of 27 web services and their semantic descriptions were developed for these databases. 10 goal templates were then developed to provide the functionality of the integrated system. A WWW user interface was used to produce goals by filling these goal templates. Based on these goals, the IRS III SWS platform executed the appropriate web services.

The overall result of this study was that semantic web services can be used for system integration. It is clear though that this kind of integration is entirely possible without the usage of semantic web services because of the stability and simplicity of the integration setup. No comparison to some more conventional approach was done in the case study.

6.1.3 The emergency situation eGovernment case study

In DIP case study SWS Enhanced GIS prototype (eMerges) a set of information systems are combined to form an environment for emergency planning [45]. The integrated systems are a weather forecast service, a database of accommodations and an instant messaging chat service. The system is build on IRS III. The functionality of the underlying systems is provided in the form of web services and exposed to the user with a Google Maps based user interface. The functionality exposed to the user is specified as goal descriptions that are linked to web services with mediator descriptions.

A detailed description of the subgoals, web services and mediators related to one example goal is given in [44]. The used WSMO execution framework IRS III relies on mediators rather than planning to connect web services to goals. The mediator may optionally be related to a web service that does the needed mediation, but if no mediation is needed the mediator is simply a notion that a specific web service corresponds to a specific goal. The web services can define restrictions to the input parameters. If these restrictions are not met, the service is not executed. This kind of mediator usage reveals the framework from the need to do any sophisticated state space planning. Each goal is explicitly linked to corresponding web services. The result is a modular architecture in which new services may be introduced and bound to the existing goals easily.

When the goal is given for the IRS-III execution framework, the corresponding web service is executed. If there are more than one available web services for the given goal, one is arbitrarily picked. A web service may be implemented as an orchestration of goals. In the example a query

for accommodations on a specific area is implemented by satisfying three different goals in a sequence: translating of the area definition from a polygon to a circle, retrieving the information about the accommodations inside the circle and filtering the resulting set of accommodations according to the availability of heating in the building.

6.1.4 The mortgage comparator eBanking case study

DIP case study Mortgage Comparison Service allows customers to compare mortgage loan offers of multiple banks [63]. Such services are currently provided by manually collecting the data or by screen craping WWW pages. It is argued that a web services approach would allow each bank to provide accurate information about their offers. In the case study four ontologies and the related web services concerning mortgage loans were developed and the implemented prototype allowed to query them through a single set of WSMO goals. The details of the mediation that was needed to integrate the services are rather blurry, but it seems that all of the services implemented exactly the same functionality and only the naming of the parameters and the related ontological terms were different.

6.2 Printing case study

In this section we walk through a case study in which we examine how semantic web services described in WSMO could be applied in the description of device services. Specifically we study the task of printing a document in an environment where a printing is not free. The WSMO descriptions in this section are based on the examples in WSMO Primer [30].

6.2.1 Ontologies

There are two ontologies involved, one for describing the payments involved and one for printing.

The payment ontology defines three concepts: payment, product, and creditCard. Payment refers to a product, which specifics the total price, receiving account and the creditCard that is charged. CreditCard could be replaced with any payment instrument that can be used over the network.

```
namespace {_"http://www.example.org/paymentOntology#"}
ontology _"http://www.example.org/paymentOntology#"

concept creditCard
  number ofType _integer

concept product
  total ofType _integer
  receivingAccount of type _string
  creditCard ofType creditCard

concept payment
  product ofType product
```

The printing ontology defines the concepts of printer, printing job, and status of printing. There is threee instances of status: waiting, processing, and done. Printing job refers to a printer, a document, and a status.

```
namespace {_"http://example.org/printingOntology#"
    payo _"http://www.example.org/paymentOntology#"}
```

```
ontology _"http://example.org/printingOntology#"

concept status

instance inqueue memberOf status
instance processing memberOf status
instance done memberOf status

concept printer

concept printingJob sybConceptOf payo#product
  status ofType status
  document ofType _string
  printer ofType printer
```

6.2.2 Goal definition

The goal is to print the document in the file "presentation.ps" with the printer that is chosen by a printer selection service and to pay for the printing with the credit card "myCreditCard". The following WSMO goal description and the related ontology corresponds to this informal description:

```
namespace {_"http://example.org/goals#",
    pro    _"http://example.org/printingOntology#"}

goal  _"http://example.org/printingGloal"

  importsOntology _"http://example.org/printingJob#"

  capability
    postcondition
      definedBy
        myPrintingJob [
           status hasValue pro#done
        ]
        .
```

The above goal imports the following ontology that describes the printing job:

```
namespace {_"http://example.org/goals#",
    pro   _"http://example.org/ptintingOntology#"
    payo  _"http://www.example.org/paymentOntology#"}

ontology _"http://example.org/printingJob#"

  instance myCreditCard memberOf payo#creditCard
    number 123456

  instance myPrintingJob memberOf pro#printingJob
    document hasValue "presentation.ps"
    creditCard hasValue myCreditCard
```

6.2.3 Service descriptions

The following services are needed:

- *printing service* that queues a given printing job to the specified printer assuming that it has been paid for
- *payment service* for handling the payment for the printing

- *printer selection service* that chooses the printer according to the preferences in the print job description and the information it has about the available printers

Below we describe the preconditions, postconditions, assumptions and effects of each service. Preconditions represent the information that is required by the service and assumptions are the requirements for the worlds state before the service can be executed. Postconditions are the information that is produced by the service and effects represent the new state of the world after the service is executed.

- Printing service
  - *preconditions*: a printing job description that describes what document is to be printed and what printer is to be used
  - *postconditions*: -
  - *assumptions*: there exists a payment that pays for the printing job
  - *effects*: the state of the printing job is marked as "done".
- Payment service
  - *preconditions*: a service description that specifies a credit card, total and the receivers account number
  - *postconditions*: -
  - *assumptions*: -
  - *effects*: a payment that pays for the specified product
- Printer selection service
  - *preconditions*: printing job that does not have a selected printer
  - *postconditions*: printer, the total cost of the printing, and the account to which the printing must be paid are specified for the printing job
  - *assumptions*: -
  - *effects*: -

```
namespace {_"http://example.org/printerSelection#",
   pro      _"http://example.org/printingOntology#"}

webService _"http://example.org/printerSelectionService"

capability

  sharedVariables {?printingJob}

  precondition
      definedBy
        ?printingJob [
          printer hasValue wsml#false
        ] memberOf pro#printingJob
   .

  postcondition
      definedBy
        ?printingJob [
```

```
            printer hasValue ?printer
            total hasValue ?price
            receivingAccount ?receivingAccount
        ]
        .


namespace {_"http://example.org/printing#",
   pro      _"http://example.org/printingOntology#",
   payo    _"http://www.example.org/paymentOntology#"}

webService _"http://example.org/printingService"

capability

  sharedVariables {?printingJob}

  precondition
      definedBy
        ?printingJob[
          printer ?printer
          document hasValue ?document
          state hasValue ?state
        ] memberOf pro#printingJob
        and
        neg ?state hasValue pro#done
        and
        ?payment[
     paidService hasValue ?printingJob
        ] memberOf payo#payment
        .

  effect
     definedBy
       ?printingJob[
     state hasValue pro#done
       ] memberOf pro#printingJob
       .


namespace {_"http://example.org/payment#",
    pro    _"http://www.example.org/printingOntology#",
    payo    _"http://www.example.org/paymentOntology#"}

webService _"http://example.org/PaymentService"

capability PaymentCapability

  sharedVariables {?service}

  precondition
      definedBy
        ?product [
          creditCard hasValue ?creditCard
          total ?total
          receivingAccount ?receivingAccount
        ] memberOf payo#product
        .

  effect
    definedBy
       ?payment[
         product hasValue ?product
       ] memberOf payo#payment
       .
```

6.2.4 Reasoning

The planning process goes as follows:

1. The goal includes a printing job that has the state "done". The planner notices that the printing service can provide this.
2. The printing service requires that the printing job specifies the printer. The printer is not specified in the goal but the printer selection service can do the selection.
3. The printing service also requires that there exists a payment for the printing job. The payment service can provide this but only after the price and receivers account are specified, which is done by the printer selection service.
4. Now the planner has all the needed information for composing and executing the needed services. It runs the services in the order: printer selection, payment and printing.

6.2.5 Discussion

The benefit of automatic discovery and composition of services provided by WSMO would show up in a scenario where there are many different kinds of services that could fulfill the given goal. There could for example be a printing service that does not require a payment. The same goal as used here could also be used to invoke that service. Here the credit card is explicitly specified in the goal, but it could be left unspecified and a general credit card selection service could be provided to select which credit card the user wants to use when paying for specific facilities or products. That way the goal to print a document would not have to refer to the payment details at all.

It is not clear whether there the distinction between the information space and the physical world, i.e. between preconditions and assumptions and between postconditions and effects is significant for the planning. It seems that the choice between these is rather arbitrary.

There has to be made a decision about how much of the business logic is implemented using the goals and service descriptions and how much is left for the services themselves. In this case study the printer selection could have been achieved by describing each printer as a distinct service with differing preconditions. Here this decision was left to he printer selection service.

Chapter 7

Conclusions

The research around Web Services, service oriented computing, service composition, and Semantic Web Services is intense. There is a lot of interest in finding ways to create an infrastructure where services could be described in a manner that allows their dynamic discovery, composition, and invocation.

However, the underlying infrastructure – various Web Service standards and recommendations – is constantly evolving and becoming very complex. Many concepts and methods from artificial intelligence research have been brought into the work. The development of the Semantic Web has been characterized by the input from the knowledge representation community (e.g., description logics and rule based systems), while in the Semantic Web Services area the AI Planning research has had a big impact.

Large proposals for Semantic Web Services frameworks have been presented (OWL-S, SWSF, and WSMO) but it is already obvious that none of them is going to be "accepted" as such. They will, however, affect the future work in Web Services such as the development of Semantic Annotations for WSDL. It is possible that new problems and ideas may also emerge over the time. The ideas with IOPEs and planning have so far not been tested in large scale service composition problems. Practical experiences are likely to bring new insights.

No real Semantic Web Services applications or large scale service-specific ontologies exist at the moment. There are, however, some demonstration systems and tools that are encouraging.

There are several questions that are still open or require experience or empirical research:

- What are the right representation primitives for services? How should the functional, behavioral, and non-functional properties of services be described?
- What should the service interaction process be like? Will some form of the publish-find-bind model be sufficient, or will the focus in the future research turn to negotiation phase or monitoring and replanning of the execution of a service?
- Are the assumptions behind planning techniques realistic? Will their performance be sufficient in typical reasoning tasks?
- Is general purpose planners the right approach or would goal specific reasoners (travel planning, eCommerce) be more realistic? What are the tradeoffs with respect to performance, the requirements for uniform representations (least common denominator or possibly too strict requirements), the treatment of multiple goal interactions with specialized reasoners?

Bibliography

[1] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[2] Steve Battle, Abraham Bernstein, Harold Boley, Benjamin Grosof, Michael Gruninger, Richard Hull, Michael Kifer, David Martin, Sheila McIlraith, Deborah McGuinness, Jianwen Su, and Said Tabet. *Semantic Web Services Framework (SWSF) Overview*. W3C, 2005. URL: http://www.w3.org/Submission/SWSF/.

[3] Steve Battle, Abraham Bernstein, Harold Boley, Benjamin Grosof, Michael Gruninger, Richard Hull, Michael Kifer, David Martin, Sheila McIlraith, Deborah McGuinness, Jianwen Su, and Said Tabet. *Semantic Web Services Language (SWSL)*. W3C, 2005. URL: http://www.w3.org/Submission/SWSF-SWSL/.

[4] Steve Battle, Abraham Bernstein, Harold Boley, Benjamin Grosof, Michael Gruninger, Richard Hull, Michael Kifer, David Martin, Sheila McIlraith, Deborah McGuinness, Jianwen Su, and Said Tabet. *Semantic Web Services Ontology (SWSO)*. W3C, 2005. URL: http://www.w3.org/Submission/SWSF-SWSO/.

[5] Sean Bechhofer. *OWL Reasoning Examples*. University of Manchester, 2003. URL: http://owl.man.ac.uk/2003/why/latest/.

[6] D. Berardi, D. Calvanese, G. De Giacomo, R. Hull, and M. Mecella. Automatic composition of web services in Colombo. *Proc. of 13th Itallian Symp. on Advanced Database Systems*, 2005. URL: http://www.inf.unibz.it/*sim*calvanese/papers/bera-etal-SEBD-2005.pdf.

[7] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of e-services that export their behavior. *Proc. 1st Int. Conf. on Service Oriented Computing (ICSOC*, 2910:43–58, 2003. URL: http://www.dis.uniroma1.it/*sim*berardi/publications/conferences/BCDLM-ICSOC03.pdf.

[8] D. Berardi, G. De Giacomo, M. Lenzerini, M. Mecella, and D. Calvanese. Synthesis of underspecified composite e-services based on automated reasoning. *Proceedings of the 2nd international conference on Service oriented computing*, pages 105–114, 2004. URL: http://www.dis.uniroma1.it/*sim*berardi/publications/conferences/BCDLM_P4WGS2004.pdf.

[9] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Rick Hull, and Massimo Mecella. Automatic Composition of Transition-based Semantic Web Services with Messaging.

In *Proc. of the 31st Int. Conf. on Very Large Data Bases (VLDB 2005)*, pages 613–624, 2005. URL: http://www.inf.unibz.it/∼calvanese/papers-html/VLDB-2005.html.

[10] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic Web. *Scientific American*, 284(5):28–37, 2001.

[11] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. MBP: a Model Based Planner. *Proc. of the IJCAI 01 Workshop on Planning under Uncertainty and Incomplete Information*, 2001. URL: http://citeseer.ist.psu.edu/695227.html.

[12] A. Blum and M. Furst. Fast Planning through Plan-graph Analysis. *Proceedings of IJCAI-95*, 1995.

[13] A. Blum and M.L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1):281–300, 1997.

[14] B. Bonet and H. Geffner. HSP: Heuristic search planner. *AIPS-98 Planning Competition Pittsburgh, PA*, 1998. URL: http://citeseer.ist.psu.edu/137014.html.

[15] B. Bonet and H. Geffner. Planning as heuristic search: New results. *Proceedings of ECP*, 99:360–372, 1999. URL: http://www.isi.edu/*sim*blythe/cs541/2000/Readings/hsp-r.ps.

[16] B. Bonet and H. Geffner. Heuristic Search Planner 2.0. *AI Magazine*, 22(3):77–80, 2001.

[17] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1):5–33, 2001. URL: http://www.isi.edu/*sim*blythe/cs541/Readings/hsp-aij.ps.

[18] M. Burstein, C. Bussler, T. Finin, M.N. Huhns, M. Paolucci, A.P. Sheth, S. Williams, and M. Zaremba. A semantic Web services architecture. *IEEE Internet Computing*, 9(5):72–81, 2005. URL: http://ebiquity.umbc.edu/_file_directory_/papers/208.pdf.

[19] M.H. Burstein. Dynamic invocation of semantic Web services that use unfamiliar ontologies. *Intelligent Systems*, 19(4):67–73, 2004. URL: http://portal.acm.org/citation.cfm?id=1024986.

[20] Chalmers - Göteborg University. *MiniSat - Minimalist open-source SAT solver*. URL: http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/.

[21] A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso. Planning via Model Checking: A Decision Procedure for AR. *Proceeding of the Fourth European Conference on Planning*, pages 130–142, 1997. URL: http://sra.itc.it/people/traverso/papers/ecp97.ps.gz.

[22] Flávio Soares Corrêa da Silva. On the Meaning of Meaning (position paper). SemGrail 2007 Workshop online proceedings, June 2007. URL: http://research.microsoft.com/workshops/SemGrail2007/Papers/FlavioS_Position.doc.

[23] Digital Enterprise Research Institute (DERI). *Triple Space Computing*. URL: http://tsc.deri.at/.

[24] Digital Enterprise Research Institute (DERI). *Web Service Execution Environment*. URL: http://www.wsmx.org/.

[25] Jean Dollimore, Tim Kindberg, and George Coulouris. *Distributed Systems – Concepts and Design*. Addison Wesley/Pearson Education, 4th edition, 2005. URL: http://cdk4.net.

[26] Christian Drumm, Liliana Cabral, John Domingue, Harald Fuchs, Leticia Gutierrez, Mary Rowlatt, and Robert Davies. DIP WP9: Case Study eGovernment – D9.4 Change of circumstances prototype. Technical report, SemanticWeb.Org, 2005. URL: http://dip.semanticweb.org/documents/D9-4.pdf.

[27] Dublin Core Metadata Initiative. *Dublin Core Metadata Element Set*. URL: http://dublincore.org/documents/dces/.

[28] Simon Duff, James Harland, and John Thangarajah. On proactivity and maintenance goals. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1033–1040, New York, NY, USA, 2006. ACM Press.

[29] Marlon Dumas, Justin O'Sullivan, Mitra Hervizadeh, David Edmond, and Arthur ter Hofstede. Towards A Semantic Framework for Service Description. *Proceedings of the IFIP TC2/WG2. 6 Ninth Working Conference on Database Semantics: Semantic Issues in E-Commerce Systems*, pages 277–291, 2001. URL: http://www.fit.qut.edu.au/~edmond/papers/ds9.pdf.

[30] Cristina Feier and John Domingue. *D3.1v0.1 WSMO Primer*. DERI, 2005. URL: http://www.wsmo.org/TR/d3/d3.1/v0.1/.

[31] Dieter Fensel and Christoph Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1:113–137(25), Summer 2002. URL: http://www.swsi.org/resources/wsmf-paper.pdf.

[32] Dieter Fensel, Enrico Motta, V. Richard Benjamins, Monica Crubezy, Stefan Decker, Mauro Gaspari, Rix Groenboom, William Grosso, Frank van Harmelen, Mark Musen, Enric Plaza, Guus Schreiber, Rudi Studer, and Bob Wielinga. The unified problem-solving method development language upml. *Knowledge and Information Systems*, 5(1):83–131, 2002. URL: http://www.cs.vu.nl/~frankh/abstracts/KAIS02.html.

[33] R. Fikes and N.J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3/4):189–208, 1971. URL: http://ai.stanford.edu/users/nilsson/OnlinePubs-Nils/PublishedPapers/strips.pdf.

[34] J.A. Fitzsimmons and M.J. Fitzsimmons. *Service management*. McGraw-Hill Boston, 2001.

[35] Fujitsu. *Task Computing*. URL: http://taskcomputing.org/.

[36] Michael R. Genesereth and Richard E. Fikes. *Knowledge Interchange Format (KIF)*. Standford University, version 3.0 edition, 1992. URL: http://www-ksl.stanford.edu/knowledge-sharing/papers/kif.ps.

[37] Alfonso Gerevini and Derek Long. *BNF Description of PDDL3.0*. International Planning Competition, International Conference on Automated Planning and Scheduling, October 2005. URL: http://zeus.ing.unibs.it/ipc-5/bnf.pdf.

[38] Alfonso Gerevini and Derek Long. Plan Constraints and Preferences in PDDL3 – The Language of the Fifth International Planning Competition. Technical Report RT 2005-08-47, Department of Electronics for Automation, University of Brescia, 2005. URL: http://zeus.ing.unibs.it/ipc-5/pddl-ipc5.pdf.

[39] Y. Gil and J. Blythe. How Can a Structured Representation of Capabilities Help in Planning. *Proceedings of the AAAI–Workshop on Representational Issues for Real-world Planning Systems*, 2000.

[40] F. Giunchiglia and P. Traverso. Planning as model checking. *Proc. The 5th European Conf. on Planning (ECP 99)*, pages 1–20, 1999. URL: http://www.informatik.uni-ulm.de/ki/biundo/ECP-Papers/invited-giunchiglia.ps.gz.

[41] The Globus Alliance. *Open Grid Services Architecture*. URL: http://www.globus.org/ogsa/.

[42] Bernardo Cuenca Grau. A possible simplification of the semantic web architecture. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 704–713, New York, NY, USA, 2004. ACM Press. URL: http://doi.acm.org/10.1145/988672.988769.

[43] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proc. of the Twelfth International World Wide Web Conference (WWW 2003)*, pages 48–57. ACM, 2003. URL: http://www.cs.man.ac.uk/∼horrocks/Publications/download/2003/p117-grosof.pdf.

[44] Alessio Gugliotta, John Domingue, Vlad Tanasescu, Leticia Gutierrez, Mary Rowlatt, and Robert Davies. DIP WP9: Case Study eGovernment – D9.10 GIS WSMO descriptions. Technical report, SemanticWeb.Org, 2006. URL: http://dip.semanticweb.org/documents/D9.10WSMODescriptionsv1.pdf.

[45] Alessio Gugliotta, Vlad Tanasescu, John Domingue, Leticia Gutierrez, Rob Davies, Mary Rowlatt, Jon Bryant, Sandra Stincic, and Marc Richardson. DIP WP9: Case Study eGovernment – D9.12 SWS Enhanced GIS prototype (IRSIII) v2.0. Technical report, SemanticWeb.Org, 2006. URL: http://dip.semanticweb.org/documents/D9.12SWSEnhancedGISPrototypeIRSIII-v2.pdf.

[46] Alon Y. Halevy. Answering queries using views: A survey. *VLDB Journal: Very Large Data Bases*, 10(4):270–294, 2001. URL: http://citeseer.ist.psu.edu/halevy00answering.html.

[47] Alon Y. Halevy, Zachary G. Ives, Dan Suciu, and Igor Tatarinov. Schema Mediation for Large-Scale Semantic Data Sharing. *VLDB Journal : the International Journal on Very*

Large Data Bases, 14(1):68–83, 2005. URL: http://dx.doi.org/10.1007/s00778-003-0116-y.

[48] J. Hoffmann. FF: The fast-forward planning system. *AI MAG*, 22(3):57–62, 2001.

[49] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, , Said Tabet, Benjamin Grosof, and Mike Dean. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. W3C, 2004. URL: http://www.w3.org/Submission/SWRL/.

[50] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003. URL: http://www.cs.man.ac.uk/∼horrocks/Publications/download/2003/HoPH03a.pdf.

[51] Richard Hull and Jianwen Su. Tools for design of composite Web services. *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 958–961, 2004. URL: http://doi.acm.org/10.1145/1007568.1007722.

[52] David Hyatt, Ben Goodger, Ian Hickson, David Hyatt, and Chris Waterson. *XML User Interface Language (XUL) 1.0*. Mozilla.org, 2001. URL: http://www.mozilla.org/projects/xul/xul.html.

[53] N. Jennings, T. Norman, P. Faratin, P. O'Brien, and B. Odgers. Autonomous agents for business process management. *Journal of Applied Artificial Intelligence*, 14(2):145–189, 2000.

[54] R.M. Jensen and M. M. Veloso. Obdd-based universal planning for synchronized agents in non-deterministic domains. *Journal of Artificial Intelligence Research*, 13:189–226, 2000. URL: http://www.itu.dk/people/rmj/data/papers/JV00A.pdf.

[55] Steve Jones. Toward an accepatble definition of service. *IEEE Software*, 22(3):87–93, 2005. URL: http://doi.ieeecomputersociety.org/10.1109/MS.2005.80.

[56] H. Kautz and B. Selman. Planning as satisfiability. *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 359–363, 1992. URL: http://www.cs.washington.edu/homes/kautz/papers/satplan.ps.

[57] Henry Kautz. Deconstructing Planning as Satisfiability. In *Proceedings of the Twenty-first National Conference on Artificial Intelligence (AAAI-06)*, Boston, MA, 2006. URL: http://www.cs.rochester.edu/u/kautz/papers/AAAI0609KautzA.pdf.

[58] Michael Kifer, Jos de Bruijn, Harold Boley, and Dieter Fensel. A Realistic Architecture for the Semantic Web. In *Proceedings of the International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML-2005)*, number 3791 in Lecture Notes in Computer Science, pages 17–29, Ireland, Galway, November 2005. Springer. URL: http://www.debruijn.net/publications/msa-ruleml05.pdf.

[59] Mathias Klusch, Andreas Gerber, and Marcus Schmidt. Semantic Web Service Composition Planning with OWLS-XPlan. In *Proceedings of the 1st Intl. AAAI Fall Symposium on Agents and the Semantic Web*, Arlington VA, USA, 2005. AAAI Press. URL: http://www-ags.dfki.uni-sb.de/∼klusch/owls-xplan/owlsxplan-3.pdf.

[60] Knowledge Media Institute, The Open University. *IRS - Internet Reasoning Service*. URL: http://kmi.open.ac.uk/projects/irs/.

[61] J. Koehler. Handling of conditional effects and negative goals in IPP. Technical report, Tech. rep. 128, Institute for Computer Science, Albert Ludwigs University, Freiburg, Germany, 1999. URL: http://historical.ncstrl.org/litesite-data/ifi_de/report00128.ps.gz.

[62] D. Long and M. Fox. Efficient Implementation of the Plan Graph in STAN. *Journal of Artificial IntelligenceResearch*, 10:87–115, 1999. URL: http://www.cs.cmu.edu/afs/cs.cmu.edu/project/jair/OldFiles/OldFiles/pub/volume10/long99

[63] Silvestre Losada, Jordi Ribas, Jesús Contreras, José Luís Bas, Sergio Bellido, José Manuel Gómez, Richard Benjamins, and Ignacio González. DIP WP10: Case study eBanking – D 10.5 Mortgage Comparison Service. Technical report, SemanticWeb.Org, 2005. URL: http://dip.semanticweb.org/documents/10.5_Final.pdf.

[64] Christopher Lovelock, Sandra Vandermerwe, and Barbara Lewis. *Services Marketing*. Prentice Hall, 1996.

[65] Christopher Lovelock and Jochen Wirtz. *Services Marketing: People, Technology, Strategy*. Prentice Hall, 5th edition, 2004.

[66] A. Marconi, M. Pistore, P. Poccianti, and P. Traverso. AutomatedWeb Service Composition at Work: the Amazon/MPS Case Study. *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 767–774, 2007. URL: http://doi.ieeecomputersociety.org/10.1109/ICWS.2007.50.

[67] David Martin, Massimo Paolucci, Sheila McIlraith, Mark Burstein, Drew McDermott, Deborah McGuinness, Bijan Parsia, Terry Payne, Marta Sabou, Monika Solanki, Naveen Srinivasan, and Katia Sycara. Bringing Semantics to Web Services: The OWL-S Approach. In *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, California, USA, 2004. URL: http://www.daml.org/services/owl-s/OWL-S-SWSWPC2004-CameraReady.doc.

[68] Maryland University. *Mindswap - Maryland Information and Network Dynamics Lab Semantic Web Agents Project*. URL: http://www.mindswap.org/.

[69] Sheila McIlraith, Tran Cao Son, and Honglei Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46–53, 2001. URL: http://dx.doi.org/10.1109/5254.920599.

[70] NIST. *PSL Ontology – Current Theories and Extensions (version 2.5)*. URL: http://www.mel.nist.gov/psl/ontology.html.

[71] P. Oaks and A. ter Hofstede. Guided interaction: A mechanism to enable ad hoc service interaction. *Information Systems Frontiers*, 9(1):29–51, 2007.

[72] OASIS. *EbXML Registry, Version 3.0.1*. URL: http://www.ebxml.org/.

[73] OASIS. *Universal Description Discovery and Integration, Version 3.0.2*, 2004. URL: http://uddi.org/pubs/uddi_v3.htm.

[74] OASIS. *Web Services Business Process Execution Language, Version 2.0*, 2007. URL: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html.

[75] Object Management Group. *Common Object Request Broker Architecture (CORBA)*. URL: http://www.omg.org/corba.

[76] Justin O'Sullivan, David Edmond, and Arthur Ter Hofstede. What's in a Service? Towards accurate description of non-functional service properties. *Distributed and Parallel Databases*, 12(2-3):117–133, 2002. URL: http://citeseer.ist.psu.edu/663154.html.

[77] Alison Pease, Simon Colton, Alan Smaill, and John Lee. Semantic Negotiation: Modeling Ambiguity in Dialogue. In *EDILOG 2002 - 6th workshop on the semantics and pragmatics of dialogue*, Edinburgh, UK, 2002. Institute for Communicating and Collaborative Systems at The University of Edinburgh. URL: http://www.ltg.ed.ac.uk/edilog/papers/125.pdf.

[78] Joachim Peer. Web Service Composition as AI Planning – a Survey. Technical report, University of St. Gallen, Switzerland, 2005. URL: http://elektra.mcm.unisg.ch/pbwsc/docs/pfwsc.pdf.

[79] M. Pistore and P. Traverso. Planning as model checking for extended goals in non-deterministic domains. *Proc. IJCAI*, 1:479–484, 2001. URL: http://sra.itc.it/tr/PT01.pdf.

[80] M. Pistore, P. Traverso, P. Bertoli, and A. Marconi. Automated synthesis of composite BPEL4WS Web services. *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*, pages 293–301, 2005. URL: http://sra.itc.it/people/marconi/links/paper/ICWS05.pdf.

[81] Erhard Rahm and Phil Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 10(4):334–350, 2001. URL: http://research.microsoft.com/∼philbe/VLDBJ-Dec2001.pdf.

[82] Marc Richardson. DIP WP8: B2B in Telecommunications – D8.6b Contract Catalogue Prototype v2. Technical report, SemanticWeb.Org, 2007. URL: http://dip.semanticweb.org/documents/D8.5bContractCataloguePrototypeV2.pdf.

[83] RuleML Initiative. *RuleML - The Rule Markup Initiative*. URL: http://www.ruleml.org/.

[84] SemanticWeb.Org. *Data, Information, and Process Integration with Semantic Web Services (DIP)*. URL: http://dip.semanticweb.org/.

[85] SemWebCentral.org. *SweetRules: Tools for Semantic Web Rules and Ontologies, including Translation, Inferencing, Analysis, and Authoring*, 2005. URL: http://sweetrules.projects.semwebcentral.org/.

[86] M.P. Singh and M.N. Huhns. *Service-oriented Computing: Semantics, Processes, Agents*. Wiley, West Sussex, England, 2005.

[87] Evren Sirin. *Combining Description Logic Reasoning with AI Planning for Composition of Web Services*. PhD thesis, University of Maryland, Department of Computer Science, 2006. URL: http://hdl.handle.net/1903/4070.

[88] Evren Sirin, Bijan Parsia, Dan Wu, James A. Hendler, and Dana S. Nau. HTN planning for Web Service composition using SHOP2. *Journal of Web Semantics*, 1(4):377–396, 2004. URL: http://citeseer.ist.psu.edu/sirin04htn.html.

[89] SRI. *OWL Web Ontology Language, Version 1.2 Pre-Release*, 2006. URL: http://www.ai.sri.com/daml/services/owl-s/1.2/.

[90] N. Srinivasan, M. Paolucci, and K. Sycara. An efficient algorithm for OWL-S based semantic search in UDDI. *Lecture Notes in Computer Science*, 3387, 2005. URL: http://www.daml.ri.cmu.edu/matchmaker/download/cr-swsws-paper.pdf.

[91] Biplav Srivastava and Jana Koehler. Web Service Composition - Current Solutions and Open Problems. In *ICAPS 2003 Workshop on Planning for Web Services, Online Proceedings*, Trento, Italy, 2003. The 13th International Conference on Automated Planning and Scheduling. URL: http://www.isi.edu/info-agents/workshops/icaps2003-p4ws/papers/srivastava-icaps2003-p4ws.pdf.

[92] TKK. *Smodels*. URL: http://www.tcs.hut.fi/Software/smodels/.

[93] Ioan Toma and Douglas Foxvog. Non-functional properties in Web services. Technical report, DERI, 2006. URL: http://www.wsmo.org/TR/d28/d28.4/v0.1/.

[94] TU Vienna. *The DLV Project - A Disjunctive Datalog System*. URL: http://www.dbai.tuwien.ac.at/proj/dlv/.

[95] M. Turner, D. Budgen, and P. Brereton. Turning software into a service. *Computer*, 36(10):38–44, 2003.

[96] Universitat des Saarlandes. *OWL-S Service Composition Planner*. URL: http://www-ags.dfki.uni-sb.de/~klusch/owls-xplan/.

[97] University of Manchester. *FaCT++*. URL: http://owl.man.ac.uk/factplusplus/.

[98] University of Maryland, MA, USA. *Simple Hierarchical Ordered Planner*. URL: http://www.cs.umd.edu/projects/shop/.

[99] University of Rochester. *SatPlan - Planning as Satisfiability*. URL: http://www.cs.rochester.edu/u/kautz/satplan/.

[100] W.M.P. van der Aalst. Don't go with the flow: Web services composition standards exposed. *IEEE Intelligent Systems*, 18(1):72–76, 2003.

[101] W3C. *Resource Description Framework (RDF)*. URL: http://www.w3.org/RDF/.

[102] W3C. *XML Schema*. URL: http://www.w3.org/XML/Schema.

[103] W3C. *Extensible Markup Language (XML)*, 1998. URL: http://www.w3.org/XML/.

[104] W3C. *Scalable Vector Graphics (SVG)*, 2003. URL: http://www.w3.org/Graphics/SVG/.

[105] W3C. *OWL-S: Semantic Markup for Web Services*, 2004. URL: http://www.w3.org/Submission/OWL-S/.

[106] W3C. *OWL Web Ontology Language*, 2004. URL: http://www.w3.org/TR/owl-features/.

[107] W3C. *RDF Test Cases*, 2004. URL: http://www.w3.org/TR/rdf-testcases/.

[108] W3C. *RDF Vocabulary Description Language 1.0: RDF Schema*, 2004. URL: http://www.w3.org/TR/rdf-schema/.

[109] W3C. *Web Services Architecture*, 2004. URL: http://www.w3.org/TR/ws-arch/.

[110] W3C. *Web Services Choreography Description Language, Version 1.0*, 2004. URL: http://www.w3.org/TR/ws-cdl-10/.

[111] W3C. *Web Services Description Language (WSDL), Version 2.0, Part 0: Primer*, 2004. URL: http://www.w3.org/TR/2004/WD-wsdl20-primer-20041221/.

[112] W3C. *Primer: Getting into RDF and Semantic Web using N3*, 2005. URL: http://www.w3.org/2000/10/swap/Primer.

[113] W3C. *SOAP version 1.2*, 2007. URL: http://www.w3.org/TR/soap12.

[114] W3C. *Web Services Description Language (WSDL), Version 2.0, Part 1: Core Language*, 2007. URL: http://www.w3.org/TR/wsdl20/.

[115] P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Analysis of Web Services Composition Languages: The Case of BPEL4WS. *22nd International Conference on Conceptual Modeling (ER 2003)*, 2813:200–215, 2003.

[116] WSMO.org. *The Web Service Modeling Language WSML*, 2005. URL: http://www.wsmo.org/wsml/wsml-syntax.