**Zur Erlangung des akademischen Grades eines
Doktors der Wirtschaftswissenschaften (Dr. rer. pol.)
von der Fakultät für Wirtschaftswissenschaften
der Universität Karlsruhe (TH)
genehmigte Dissertation.**

Formal Description of Web Services for Expressive Matchmaking

Dipl.-Inform. Sudhir Agarwal

Tag der mündlichen Prüfung: 04.05.2007

Referent: Prof. Dr. Rudi Studer

Koreferent: Prof. Dr. Christof Weinhardt

Prüfer: Prof. Dr. Andreas Oberweis

2007 Karlsruhe

Abstract

Web services are programmable interfaces that can be invoked via standardized Web communication protocols, such as HTTP, in order to embed the functionality they offer in a software system. The necessity of outsourcing parts of business processes together with the wide acceptance of the Service Oriented Architecture (SOA) will lead to a large number of Web services. Such a big market of Web services demands automatic methods for finding suitable Web services. Existing industrial standards for finding Web services are keyword based and hence require a lot of manual effort to select Web services with desired functionality and desired interface.

Industrial standard UDDI allow only keyword based search that considers a fixed set of properties and methods known from research in the field of Semantic Web Services base on interoperable descriptions of Web service interfaces. However, in real business scenarios criteria for searching suitable Web services are much richer than just the types of input and output paremeters. In this work, our aim is to develop automatic methods for finding Web services based on the functional and non-functional properties of Web services. To be able to develop automatic reasoning algorithms about the functional and non-functional properties of Web services, they must be described formally. So, we first develop a formalism that allows to describe the involved resources, the credentials of the involved actors and the dynamic behavior of web services including access control policies in a unifying way.

We identify that in order to support a broader spectrum of use cases, two types of matchmaking approaches are needed, namely goal based and equivalence based. Goal based matchmaking deals with specifying constraints on desired Web services and then finding the Web services that satisfy the constraints, whereas equivalence based matchmaking supports the use cases in which an existing Web service should be replaced by another Web service without changing the overall behaviour of the system it is incorporated in. In this work, we develop an expressive formalism to specify desired constraints on a Web service and develop a sound and complete algorithm to check whether a Web service description fulfills a goal. Furthermore, we develop the notion of equivalence of Web service functionalities and show how two web service descriptions can be checked for equivalence. Finally, we present the implementation of our system and conclude by discussing some problems that may be of future research interest.

Contents

List of Figures

Part I.

Introduction

1. Motivation

From a historical perspective, Web Services represent the convergence between the Service-Oriented Architecture (SOA) and the Web. SOAs have evolved over the last 10 years to support high performance, scalability, reliability, and availability. To achieve the best performance, applications are designed as services that run on a cluster of centralized application servers. A service is an application that can be accessed through a programmable interface. In the past, clients accessed these services using a tightly coupled, distributed computing protocol, such as DCOM, CORBA, or RMI. While these protocols are very effective for building a specific application, they limit the flexibility of the system. The tight coupling used in this architecture limits the re-usability of individual services. Each of the protocols is constrained by dependencies on vendor implementations, platforms, languages, or data encoding schemes that severely limit interoperability. And none of these protocols operates effectively over the Web. The Web Services architecture takes all the best features of the service-oriented architecture and combines it with the Web. The Web supports universal communication using loosely coupled connections. Web protocols are completely vendor-, platform-, and language-independent. The resulting effect is an architecture that eliminates the usual constraints of DCOM, CORBA, or RMI. Web Services support Web based access, easy integration, and service re-usability.

1.1. Introduction to Web Services

The World Wide Web is more and more used for application to application communication. The programmatic interfaces made available are referred to as Web services [W3C02].

A Web Service is programmable application logic accessible using standard Internet protocols. Web Services combine the best aspects of component-based development and the Web. Like components, Web Services represent functionality that can be easily reused without knowing how the service is implemented. Unlike current component technologies which are accessed via proprietary protocols, Web Services are accessed via ubiquitous Web protocols, e.g. HTTP, using universally-accepted data formats, e.g. XML.

Web services are meant to be offered as building blocks of applications in Internet, intranet, or extranet. Web Services can support business-to-consumer, business-to- business, department-to-department, or peer-to-peer interactions. A Web Service consumer can be a human user accessing the service through a desktop or wireless browser; it can be an application program; or it can be another Web Service. Web Services support existing security frameworks.

Technically, Web services are programming language independent, platform independent remote procedures, that can be invoked using web protocols. Web services help in achieving increased automation across organizational boundaries but also within a large organization.

Existing definitions of Web services range from the very generic and all-inclusive to the very specific and restrictive. Often, a web service is seen as an application accessible to other applications over the Web. this is a very open definition, under which just about anything that has a URL is a Web service. It can, for instance, include a CGI script. It can also refer to a program accessible over the Web with a stable API, published with additional descriptive information on some service directory.

A more precise definition is provided by the UDDI consortium, which characterizes Web services as "self-contained, modular business applications that have open, Internet-oriented, standards-based interfaces" [UDD01].

A step further in refining the definition of Web services is the one provided by the World Wide Web consortium (W3C), and specifically the group involved in the Web Service Activity: "a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols" [ABFG04].

So, there is no single definition of a Web service. In the following, we list some more well known definitions[1].

- Web services is an esoteric data exchange technology that has mostly been used as a platform for connecting information infrastructures within companies [Ric04].
- Web Services are a new, standards-based approach to build integrated applications that run across an intranet, extranet, or the Internet. The approach represents a major evolution in how systems connect and interact with each other [CI01].
- A Service, any service, can be defined as a software component that can be (1) described in a formal language (2) published to a registry of services (3) discovered through standard mechanisms (4) invoked over a network (5) composed with other services [Tch04].
- Web Services are services, that can be reached/invoked via the Web(Dieter Fensel)
- The term Web Services refers to an architecture that allows applications to talk to each other [MCK03].

[1]Courtsey: Mario Jeckle http://www.jeckle.de/webServices/index.html

- A service that is accessible by means of messages sent using standard Web protocols, notations and naming conventions [FHBF+01].

1.1.1. Web Service Description Language (WSDL)

As communication protocols and message formats are standardized in the Web community, it becomes increasingly possible and important to be able to describe the communications in some structured way. WSDL addresses this need by defining an XML grammar for describing network services as collections of communication endpoints capable of exchanging messages [W3C03]. WSDL service definitions provide documentation for distributed systems and serve as a recipe for automating the details involved in applications communication.

WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate. However, it is mostly used in conjunction with SOAP 1.1, HTTP GET/POST, and MIME [W3C03].

A WSDL document defines services as collections of network endpoints, or ports. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: messages, which are abstract descriptions of the data being exchanged, and port types which are abstract collections of operations. The concrete protocol and data format specifications for a particular port type constitutes a reusable binding. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service. Hence, a WSDL document uses the following elements in the definition of network services [W3C03]:

- Types - a container for data type definitions using some type system (such as XSD).
- Message - an abstract, typed definition of the data being communicated.
- Operation - an abstract description of an action supported by the service.
- Port Type - an abstract set of operations supported by one or more endpoints.
- Binding - a concrete protocol and data format specification for a particular port type.
- Port - a single endpoint defined as a combination of a binding and a network address.

- Service - a collection of related endpoints.

In the following, we explain the various elements of a WSDL document with the help of an example.

```
<definitions name="StockQuote"
         targetNamespace="http://example.com/stockquote.wsdl"
         xmlns:tns="http://example.com/stockquote.wsdl"
         xmlns:xsd1="http://example.com/stockquote.xsd"
         xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
         xmlns="http://schemas.xmlsoap.org/wsdl/">

 <types>
      <schema targetNamespace="http://example.com/stockquote.xsd"
             xmlns="http://www.w3.org/2000/10/XMLSchema">
          <element name="TradePriceRequest">
             <complexType>
                 <all>
                     <element name="tickerSymbol" type="string"/>
                 </all>
             </complexType>
          </element>
          <element name="TradePrice">
             <complexType>
                 <all>
                     <element name="price" type="float"/>
                 </all>
             </complexType>
          </element>
      </schema>
   </types>
```

The `types` block defines the schema of the data objects that are expected or returned by the Web services defined in the WSDL document. These types can be primitive types like String, Integer or self defined complex types containing elements of primitive types or other complex types. In the above example, `TradePriceRequest` is a complex data type that contains an element of type String.

```
    <message name="GetLastTradePriceInput">
        <part name="body" element="xsd1:TradePriceRequest"/>
    </message>

    <message name="GetLastTradePriceOutput">
        <part name="body" element="xsd1:TradePrice"/>
    </message>

   <portType name="StockQuotePortType">
        <operation name="GetLastTradePrice">
           <input message="tns:GetLastTradePriceInput"/>
           <output message="tns:GetLastTradePriceOutput"/>
        </operation>
    </portType>
```

A `message` block defines a message type by listing the message parts and associating with each message a type. In our example, the message type `GetLastTradePriceInput` has only one part of type `TradePriceRequest`.

The element portType groups the set of operations that the web service can perform. The element <operation> inside <portType> defines an operation, in our example GetLastTradePrice. The order of occurrence of input and output messages inside an operation block describes the data flow. The order in our example means that the Web service first performs an input operation expecting a message of type GetLastTradePriceInput and then an output operation by transmitting a message of type GetLastTradePriceOutput to the client.

```
 <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
       <soap:binding style="document"
       transport="http://schemas.xmlsoap.org/soap/http"/>
       <operation name="GetLastTradePrice">
          <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
          <input>
              <soap:body use="literal"/>
          </input>
          <output>
              <soap:body use="literal"/>
          </output>
       </operation>
   </binding>

   <service name="StockQuoteService">
       <documentation>My first service</documentation>
       <port name="StockQuotePort" binding="tns:StockQuoteSoapBinding">
          <soap:address location="http://example.com/stockquote"/>
       </port>
   </service>
</definitions>
```

The binding elements describes the message serialization format and the communication protocol used during the execution of the web services. The protocol is described by the attribute transport, which is HTTP in our example. In addition to that, the binding element describes whether the messages are transmitted in ”RPC style” or in ”document style” and how the inputs and outputs should be encoded. In case of ”document style” only data is sent to the corresponding web service, whereas in case of ”RPC style” information about a particular method together with its parameters is sent, that is supposed to be called at the end point.

The element service defines end points (ports) and the addresses at which they can be contacted. In our example the port StockQuotePort is reachable at the address http://example.com/stockquote. The attribute binding refers to an already defined operation binding thus connecting the port type to an operation. In our example the value of the binding attribute for the port StockQuotePort is StockQuoteSoapBinding which is associated with the operation GetLastTradePrice.

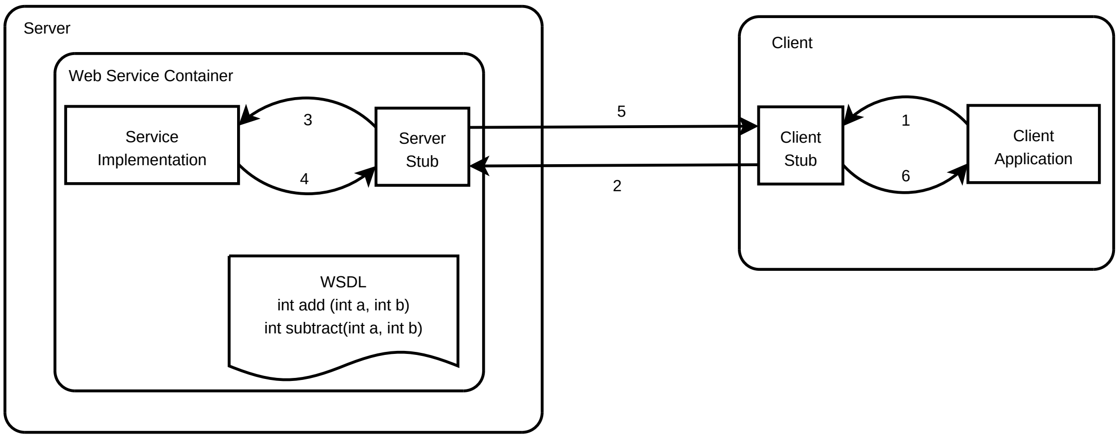


Figure 1.1.: Using Web Services in an Application

1.1.2. Simple Object Access Protocol (SOAP)

Simple Object Access Protocol (SOAP) is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics.

SOAP describes a message format along with a set of serialization rules for datatypes including structured types and arrays. In addition, it describes how to use the Hypertext Transfer Protocol (HTTP) as a transport for such messages. SOAP messages are effectively service requests sent to some end point on a network. The end point may be implemented in any number of ways (e.g. Remote Procedure Call (RPC) server, Component Object Model (COM) object, Java servlet, and Perl script) and may be running on any platform. Thus, SOAP is about interoperability between applications running on potentially disparate platforms using various implementation technologies in various programming languages.

Since execution of Web services is not our focus in this work, we do not introduce SOAP any further and refer to `http://www.w3.org/TR/soap12-part1/` for the official specification of SOAP Version 1.2.

1.1.3. Using Web Services

The steps involved in invoking a Web service are described in Figure 1.1. Let's suppose that the client has already discovered the Web service and generated the client stubs from the WSDL description. Furthermore, the server-side programmer will have generated the server stubs.

1. Whenever the client application needs to invoke the Web service, it will actually call the client stub. The client stub will turn this 'local invocation' into a proper SOAP request. This is often called the marshaling or serializing process.
2. The SOAP request is sent over a network using the HTTP protocol. The Web services container receives the SOAP requests and hands it to the server stub. The server stub will convert the SOAP request into something the service implementation can understand (this is usually called unmarshaling or deserializing).
3. The service implementation receives the request from the service stub, and carries out the work it has been asked to do. For example, if the client invokes the `int add(int a, int b)` method, the service implementation will perform an addition.
4. The result of the requested operation is handed to the server stub, which will turn it into a SOAP response.
5. The SOAP response is sent over a network using the HTTP protocol. The client stub receives the SOAP response and turns it into something the client application can understand.
6. Finally the application receives the result of the Web Service invocation and uses it.

1.2. B2B Integration and EAI with Web Services

The contribution of Web services toward resolving the limitations of conventional middleware involves three main aspects: service-oriented architectures, redesign of middleware protocols, and standardization [EF03]. Figure 1.2 depicts a typical B2B integration scenario. Organisations A,B,C and D offer their Web services to a B2B marketplace. They find potential business partners (more precisely their Web services) on the B2B marketplace and connect their businesses by integrating the corresponding Web services.

Service-oriented Paradigm

Web services work on the assumption that the functionality made available by a company will be exposed as service. In middleware terms, a service is a procedure, method, or object with a stable, published interface that can be invoked by a client. The invocation is made by a program. Thus, requesting and executing a service involves a program calling another program.

In terms of how they are used, Web services are no different from middleware services, with the exception that it should be possible to invoke them across the Web and across

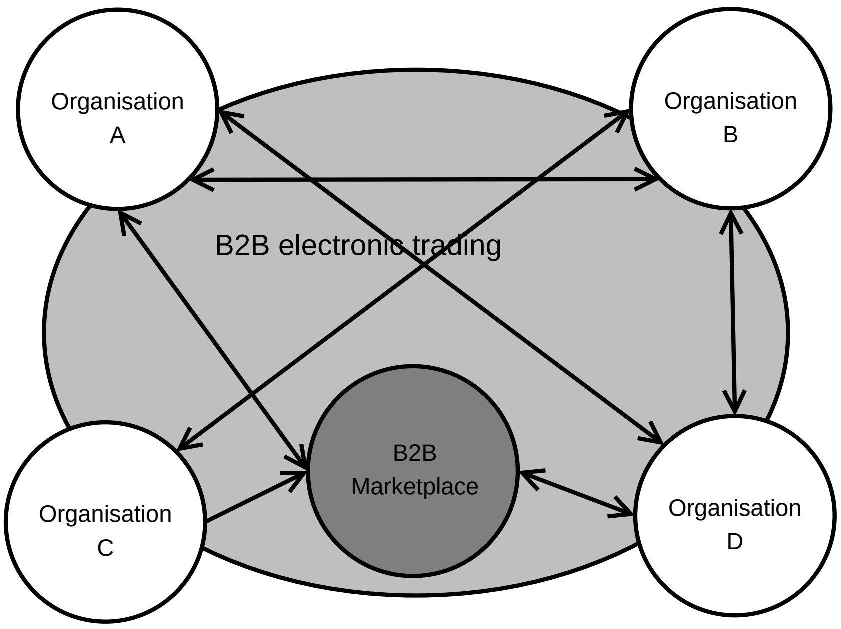


Figure 1.2.: B2B Integration

companies. As a consequence, web services assume that services are loosely-coupled, since in general they are defined, developed and managed by different companies. As web services become more popular and widely adopted, they are likely to lead to a scenario where service-oriented architectures, advocated for many years, finally become a reality. In fact, with Web services designers and developers are led to think in the direction that "everything is a service," and that different services are autonomous and independent (as opposed to being, for example, two CORBA objects developed by the same team). This interpretation has important implications in that it leads to decoupling applications and to making them more modular. Therefore, individual components can be reused and aggregated more easily and in different ways.

Middleware Protocols

The second aspect of the Web services approach is the redesign of the middleware protocols to work in a peer-to-peer fashion and across companies. Conventional middleware protocols, such as 2PC, were designed based on assumptions that do not hold in cross-organizational interactions. For example, they assumed a central transaction coordinator and the possibility for this coordinator to lock resources ad libitum. Lack of trust and confidentiality issues often make a case against a central coordinator, and therefor 2PC must now be redesigned to work in a fully distributed fashion and must be extended to allow more flexibility in terms of locking resources. Similar arguments can be made for all interaction and coordination protocols and, in general, for many for the other properties provided by conventional middleware, such as reliability and guaranteed delivery. What was then achieved by a centralized platform must be now redesigned in terms of protocols that can work in a decentralized setting and across trust domains.

Standardization

The final key ingredient of the Web services recipe is standardization. In conventional application integration, the presence of standards helped to address many problems. CORBA and Java, for example, have enabled the development of portable applications, have fostered the production of low cost middleware tools, and have considerably reduced the learning curves due to the widespread adoption of common models and abstractions. Whenever standardizations has failed or proved to be inapplicable due to the presence of legacy systems, the complexity and cost of the middleware has remained quite high and the effectiveness rather low. For Web services, where the interactions occur across companies and on a global scale, standardization is not only beneficial, but a necessity. Having a service-oriented architecture and redefining the middleware protocols is not sufficient to address the application integration problem in a general way, unless these languages and protocols become standardized and widely adopted.

Business to Business and in particular Enterprise Integration Systems represents some of the most mature service-oriented technology, but is largely based on custom solutions to large-scale e-commerce problems. It is one of the main targets of the current generation of Web service development. The key challenge in this area is semantic mediation between different organizational vocabularies, as service models and protocols have been coordinated among a few dominant organizations, although there are several competing standards.

Aim of *Enterprise Application Integration* (EAI) is to integrate applications within an organization in such a manner that they run and can be used as a homogeneous application even if the applications have been developed over a long period of time by different programmers and in different programming languages for different hardware and operating systems. This makes it possible to carry out business processes that not completely covered by a single application. The aims are significant cost reductions not only by simplified user interactions and hence faster processing, but also by unifying the data basis, that needs to be maintained only once. All application components are served by only one data source.

1.3. Web Service Markets

Increasing need for outsourcing certain steps of a business process together with wide acceptance of service oriented architectures (SOA) would lead to a large number of Web

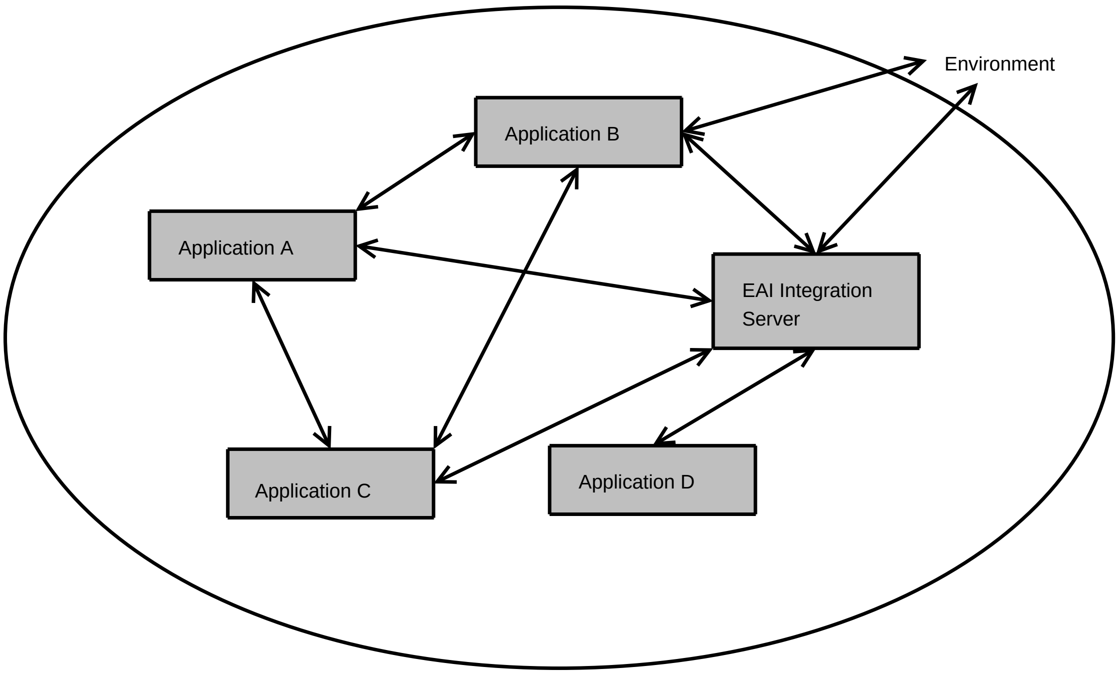


Figure 1.3.: Enterprise Application Integration

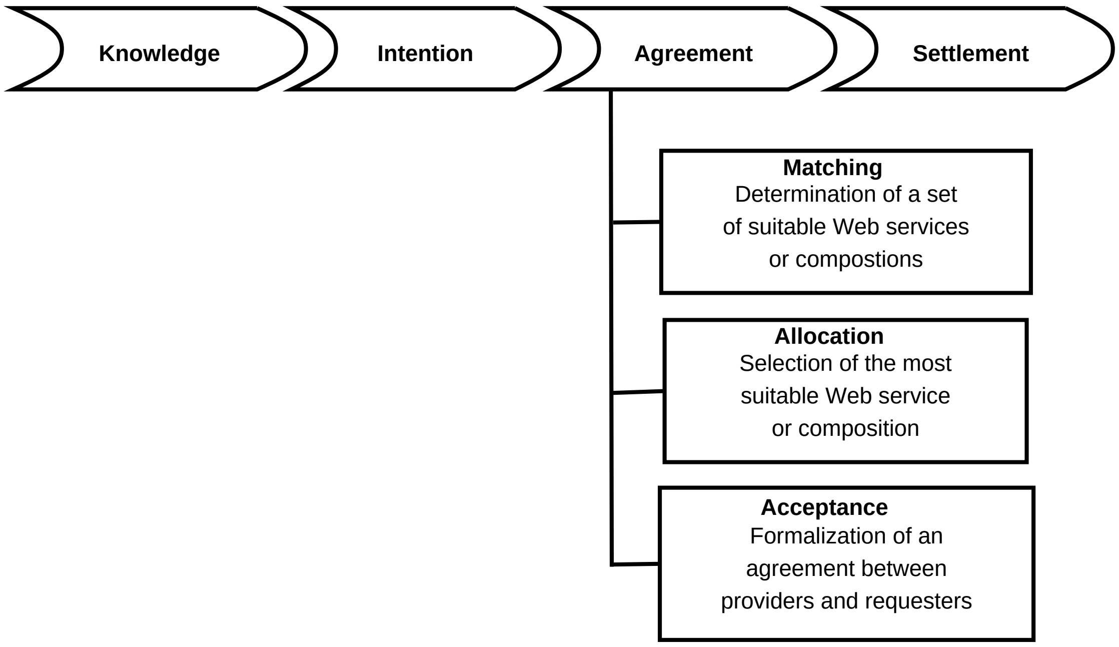

Knowledge
Intention
Agreement
Settlement
Matching
Determination of a set of suitable Web services or compostions
Allocation
Selection of the most suitable Web service or composition
Acceptance
Formalization of an agreement between providers and requesters

Figure 1.4.: Electronic Market Phases

services. Similar Web services will be offered by different actors; an actor will offer many different services; thus leading to a highly competitive market. On the other side actors will use Web services offered by other actors in order to offer their own, which leads to a cooperation based market. Markets, in which actors compete as well cooperate are often called copetitive markets. Web service markets are copetitive markets.

Electronic markets is a well studied topic and there is quite a few work done on defining and identifying the core elements of an electronic market. In this section, we give a short introduction of electronic markets and align Web services research to the phases of electronic markets.

A market transaction transfers the ownership of tangible or intangible objects (e.g. products, stocks, money etc.), or rights to services (e.g. insurance) from one market participant to another. As identified in [SW03], electronic market transactions go through four primary phases, namely *Knowledge*, *Intention*, *Agreement* and *Settlement* (refer to figure 1.4). To begin with, participants must gather information about products available in the market and about other participants. This takes place in the initial "Knowledge" phase. In the next phase, the "Intention" phase, individual market participants specify their purchase and sale offers within the market. In the following "Agreement" phase, market participants identify appropriate partners for a transaction, identify the terms and conditions of the transaction and finally, sign a contract. The agreed-upon contract is then executed in the "Settlement" phase, leading to the payment and potentially, post sale support.

The "Intention" phase is divided into three subphases, namely *Offer Specification*, *Offer Submission* and *Offer Analysis*. In this work, our main focus is on the "Agreement" phase of a Web service market. However, we will also deal with the "Offer Specification" phase, since the "Agreement Phase" builds on the offers.

Figure 1.4 shows the phases that can be identified in the agreement phase of an electronic market [SW03] in more detail, since this is mainly the phase that we address in this work. The "Agreement" is divided into steps *Matching*, *Allocation* and finally *Acceptance*.

In the "Matching" phase, the published Web service offers are filtered such that the resulting set of Web service offers are about those Web services that offer the functionality described by a requester in the request. A requester wishing to accomplish some task specifies a request for a Web service by describing the functionality needed to accomplish the task.

Even if there is no single service that directly fulfills a user's request, there may be a combination of services that do. Once user's goal is specified and it is known what the available Web services do, the idea of automatic composition of Web services is to compute possible combinations of Web services such that the execution of each such combination leads the user to the goal.

In the "Matching" phase suitable services are discovered. After having determined those services that are able to achieve a certain goal an optimal assignment of service

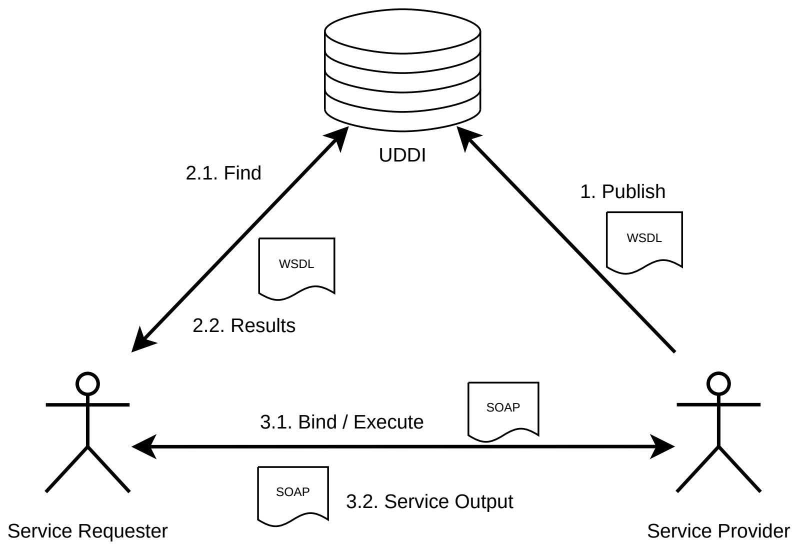


Figure 1.5.: Publish Find Bind

requests and offers with respect to the individual utility of the participants or to the overall welfare has to be found in the “Allocation” phase. To achieve this, negotiations between the participants might be required. For determining the allocation and price many different mechanisms are available ranging from simple selection approaches to complex negotiation or auction schemes. In the “Allocation” step, one of the Web services or Web service compositions is selected on the basis of requester preferences.

After this allocation, legally binding contracts are closed between the corresponding business partners in the “Acceptance” phase. These contracts have to be formalized in a machine-understandable way in order to allow automated execution and monitoring.

In this work, our aim is to address the “Matching” phase of Web service markets. However, since matchmaking of Web services depends on how Web services and search requests are described, we need to address the “Offer Specification” too.

1.4. UDDI, Current Standard for Web Service Discovery

Figure 1.5 shows the famous Web service triangle.

1. Web service providers publish the WSDL descriptions of their Web services at UDDI repository [UDD01].

2. Consumers or requesters search in the UDDI repository for desired services and can obtain WSDL descriptions of the services that fit their needs.

3. Having the WSDL document that contains information needed to execute a Web service, the requesters can invoke a Web service by sending an appropriate SOAP message and obtain the service output again as a SOAP message. For more details about this step refer to Figure 1.1.

UDDI is an acronym for Universal Description, Discovery, and Integration - A platform-independent, XML-based registry for businesses worldwide to list themselves on the internet. UDDI is an open industry initiative (sponsored by OASIS) enabling businesses to publish service listings and discover each other and define how the services or software applications interact over the Internet.

UDDI is one of the core Web Services standards. It is designed to be interrogated by SOAP messages and to provide access to Web Services Description Language documents describing the protocol bindings and message formats required to interact with the Web services listed in its directory.

UDDI was written in August, 2000, at a time when the authors had a vision of a world in which consumers of Web Services would be linked up with providers through a public or private dynamic brokerage system. In this vision, anyone needing a service such as credit card authentication, would go to their service broker and select one supporting the desired SOAP or other service interface and meeting other criteria. In such a world, the publicly operated UDDI node or broker would be critical for everyone. For the consumer, public or open brokers would only return services listed for public discovery by others, while for a service producer, getting a good placement, by relying on metadata of authoritative index categories, in the brokerage would be critical for effective placement.

The UDDI was integrated into the WS-I standard as a central pillar of Web services infrastructure. By the end of 2005, it was on the agenda for use by more than seventy percent of the Fortune 500 companies in either a public or private implementation, and particularly among those enterprises that seek to optimize software or service reuse. Many of these enterprises subscribe to some form of Service-oriented architecture (SOA), server programs or database software licensed by some of the professed founders of UDDI.org and OASIS.

The UDDI specification describes an online electronic registry that serves as electronic Yellow Pages, providing an information structure where various business entities register themselves and the services they offer through their WSDL definitions.

The UDDI specification defines a 4-tier hierarchical XML schema that provides a model for publishing, validating, and invoking information about Web Services. XML was chosen because it offers a platform-neutral view of data and allows hierarchical relationships to be described in a natural way. UDDI uses standards-based technologies, such as common Internet protocols (TCP/IP and HTTP), XML, and SOAP (a specification for using XML in simple message-based exchanges). UDDI is a standard Web Service description format and Web Service discovery protocol; a UDDI registry can contain metadata for any type of service, with best practices already defined for those described by Web Service Description Language (WSDL). There are two types of UDDI registries: public UDDI registries that serve as aggregation points for a variety of businesses to publish their services, and private UDDI registries that serve a similar role within organizations.

A UDDI registry consists of the following data structure types:

- businessEntity - The top-level XML element in a business UDDI entry, it captures the data, that the potential business partners require to find information about a business service, including its name, industry or product category, geographic location, and optional categorization and contact information. It includes support for "yellow pages" taxonomies to search for businesses by industry, product, or geography.
- businessService - The logical child of a businessEntity data structure as well as the logical parent of a bindingTemplate structure, it contains descriptive business service information about a group of related technical services including the group name, a brief description, technical service description information, and category information. By organizing Web Services into groups accociated with categories or business processes, UDDI allows more efficient search and discovery of Web Services.
- bindingTemplate - The logical child of a businessService data structure, it contains data that is relevant for applications that need to invoke or bind to a specific Web Service. This information includes the Web Service URL and other information describing hosted services, routing and load balancing facilities, and references to interface specifications.
- tModel - Descriptions of specifications for Web Services or taxonomies that form the basis for technical fingerprints; its role is to represent the technical specification of the Web Service, making it easier for Web Service consumers to find Web Services that are compatible with a particular technical specification. That is, based on the descriptions of the specifications for Web Services in the tModel structure, Web Service consumers can easily identify other compatible Web Services. For instance, to send a business partner's Web Service a purchase order, the invoking service must know not only the location/URL of the service, but what format the purchase order should be sent in, what protocols are appropriate, what security is required, an what form of a response will result after sending the purchase order.

A UDDI business registration consists of three components. The information provided in a UDDI registry can be used to perform three types of searches.

- White Pages - Basic contact information and identifiers about a company, including business name, address, contact information, and unique identifiers such as D-U-N-S numbers or tax IDs. This information allows others to discover a Web service based upon its business identification. For example, search for a business that a requester already knows something about, such as its name or some unique ID.

- Yellow Pages - industrial categorizations based on standard taxonomies such as the NAICS[2] and UNSPSC[3]. Information that describes a Web service using different categorizations (taxonomies) allows others to discover a Web service based upon its categorization (such as being in the manufacturing or car sales business).
- Green Pages - technical information about services exposed by the business including references to specifications of interfaces for Web Services, as well as support for pointers to various file and URL-based discovery mechanisms. Technical information that describes the behaviors and supported functions of a Web service hosted by its business. This information includes pointers to the grouping information of web services and where the Web services are located.

1.4.1. Limitations of UDDI

Even though the API provided by UDDI allows random searching for businesses, it's not feasible for a program to select new business partners dynamically. Realistically, it's more likely that business analysts with specific knowledge of the problem at hand will use UDDI portals to discover potentially interesting services and partners, and technologists will write programs to use the services from companies that have already been discovered. Most likely there will be programs that update the data in a UDDI registry, but most publicly available registries already have a user-friendly interface that allows human users to update information in a registry.

UDDI basically supports keyword based search, which though being fast and scalable[4] has certain limitations. Consider a Web service that sells books and the corresponding WSDL document that contains an output message of type "Book". The type "Book" is defined in the `types` block of the WSDL document. If a requester searching for a book selling web service, uses keywords, say "Buch", "hardcover", or "volume" instead of "Book" the book selling Web service will not be found as a match by the UDDI search algorithm. In other words, the keyword based search detects a Web service as match only if the requester uses the same terminology as the provider. Consider another Web service that sells books but not text books. If a requester searches for a web service that sells text books, keyword based search will detect this Web service as a match. So, keyword based search suffers from two main problems, namely lack of interoperability and lack of semantics.

Another limitation of UDDI search is the fixed terminology (cf. UDDI data set) for describing properties of Web services. Considering the idea of universality of UDDI, a

[2]North America Industry Classification System `http://www.census.gov/epcd/www/naics.html`
[3]United Nations Standard Products and Services Code `http://www.unspsc.org/`
[4]Probably the best example of scalable and efficient keyword based search is the Google search engine.

small fixed set of properties is not sufficient to describe all sorts of business services. Different business services have different properties that characterize them most appropriately. The properties of two business services may not be comparable with each other. For example, in case of an information providing Web service the response time may be interesting and there is no shipment period, whereas in case of a book selling service the inverse is more interesing.

1.5. Limitations of Existing Semantic Matchmaking Approaches

The main motivation behind semantic Web services research is to provide formalisms and tools to automate tasks needed in various phases of Web service markets. The basic idea is to describe properties and capabilities of Web services in unambiguous and machine interpretable way. There are already a few approaches available for describing Web services semantically, differentiating in the underlying formalism and expressivity. We will discuss them in detail in Section 5.3. However, there is not much new work done about matchmaking of Web services in the recent years.

Existing matchmaking approaches, though using expressive descriptions of Web service properties fail to exploit the expressiveness as they use only the interface description for the purpose of matchmaking. More precisely, most of the existing matchmaking approaches check whether the types of input and output parameters of a Web service are compatible with the desired types of input and output parameters, although the underlying description formalisms typically allow to model much more properties than just input and output parameters.

Such interface matching approaches have the drawback that they do not lead to automation, which is one of the primary goals of semantic web services. Consider for example a book selling Web service that on receiving an ISBN of the desired book and a delivery address, ships the ordered book to the delivery address. When this Web service is found on the basis of the type of output parameter (in this case "Book"), a software engineer can incorporate this Web service in a complex business process at a place, where a book is expected. However, the software engineer does not know which book the Web service actually delivers. In this case, obviously the software engineer wanted that the Web service delivers the order book, that is the book with the entered ISBN and not just any book.

In order to find out which book the found Web service actually delivers, whether he gets the electronic version of book per email, or a download link or hard copy of the buch via surface mail etc., the requester still needs to read description of each match manually. So, mere interface matching or black box views are actually bottlenecks for automation and scalability and consequently spontaneity in web service markets.

Another limitation of the existing semantic matchmaking approaches is concerning non-functional properties of Web services. Existing approaces suggest to use ontologies, in particular taxonomies of properties for describing various non-functional properties of Web services. This is an improvement over UDDI. However they do not deal with the trust aspect, which is very important while dealing with non-functional properties.

In order to achieve an infrastructure, in which clients can trust Web services on the basis of their non-functional properties (credentials), it is not practical to abstract from the issuer of such credentials. A Web service provider will obviously describe his Web service in such a way that it has credentials that the majority of clients prefer, so that his Web service is found and used often. However, when the clients know that the Web service providers can claim to have just any credentials, the credentials of the Web services will be hardly of any practical use. Rather, there is a need for a technique that allows parties different from the Web service providers to issue credentials to Web services and other parties to build trust in Web service on the basis of such credentials. That is, one needs a technique to determine which credentials of a Web service are worth considering while matchmaking and which not. The main challenge in this regard is that the technique should not base on central control.

1.6. Outline

The ultimate goal of this work is to show how semantically rich descriptions of Web services can improve Web service matchmaking. By improvement, we mean enabling or improving automation of certain tasks that a user has to accomplish in order to find a Web service that can be later incorporated in a system. The main contribution of this work is an expressive formalism for modeling resources, behavior and security related aspects of Web services in a unified way and algorithms for reasoning about such properties of Web services to enable expressive and automatic matchmaking of Web services.

In Chapter 2, we identify the need for more automation and expressivity as basic requirement for Web service discovery. **In Section 2.1, we present illustrating examples to identify criteria typically used in business scenarios to search for Web services. The examples illustrate the need for an expressive formalism that allows to specify constraints on dynamic behavior of Web services, on trustworthiness of the involved actors and on the objects involved in the execution of a web service.**

Existing matchmaking approaches on one side consider very simple kinds of constraints, on the other side have neglected the scenarios, in which a business process or a software system already uses web services and it is desired to replace a Web service by another web service, that offers equivalent functionality. In such scenarios, matchmaking based on constraint satisfaction is not efficient. Rather, there is a need for a method that directly checks two Web service descriptions for equivalence.

In order to perform expressive matchmaking, different aspects of web services must be described in a way that machines can reason about them. We consider *dynamic behavior* of Web services, *credentials* of the involved actors and *objects* involved in Web service execution. **In Section 2.2, we identify requirements for formal descriptions of Web services.**

In Part II, our focus is on developing a formalism for modeling Web services that can cover the requirements identified in Section 2.2.

Different Web service providers use different terminologies to describe the objects involved in the execution of their web services. Typically, there is no shared understanding among the web service providers as they act autonomously and independent of each other. Currently, only users have to understand the terminologies used in the description of a Web service. However, when Web services are composed, there is a need for a mechanism that can handle the mappings between the ontologies of different Web service providers. Existing approaches for describing Web services semantically mainly focus on this aspect by using ontologies for describing the involved objects in a machine understandable way. We use description logics for modeling involved objects semantically. In Chapter 3, we first introduce description logics, ontologies, the Web Ontology Language (OWL), description logic query and rule languages. **Then, we show, how the the resource schema of an agent can be modeled as a description logic TBox.**

Existing approaches for describing Web services semantically do not deal with the security related aspects of the Web services, in particular, access control. Access control roughly means, that only eligible users are allowed to access a Web service. Resolving access control issues is extremely important, before Web services can be employed in realistic business settings. In open and distributed environments like the web, identity based access control is not appropriate, since there is no central instance to check the correctness of the identities. On the other hand, the identities of the actors are often unknown. In such environments capability or credential based access control is more suitable. In Chapter 4, we introduce credential based access control, especially the Simple Distributed Security Infrastructure (SDSI) and the Simple Public Key Infrastructure (SPKI) and their combination known as SPKI/SDSI. Even though credential based access control is the right paradigm for access control in distributed and open systems, the available technologies suffer from interoperability problems, since they do not allow to describe relationships among the certified credentials. For example, it is not possible to say that the property `above21` certified by actor A is a subset of the property `adult` certified by actor B. **In Section 4.3, we show how access control policies and certification policies in a credential bases access control system, can be modeled in an interoperable way with the help of ontologies.**

Invoking a Web service can trigger a complex business process that involves many independent and autonomous actors. Assuming a central instance that controls the execution of such a distributed process limits the application of Web services to only mediator like web services, which means a Web service coordinates all the interactions with its component Web services and appears as a black box to its user. However, in many cases a Web service is not a mediator but delegates the responsibility of certain subtasks to component Web services. For example, a book selling Web service delegates the task of shipping the book to the customer to some shipping service. For modeling behavioral aspects of Web services, we use polyadic π-calculus with typed channels, which we introduce in Chapter 5. Even though π-calculus is a very expressive formalism while still having a simple syntax, it is not practical to use it in its pure form. The reason is that it suggests to model everything as processes and hence makes it impossible to model certain steps as black-boxes. **In Section 5.2, we augment the formalism with a notion of *local operations* to overcome this problem.**

Having description logics, π-calculus and credential bases access control as basic formalisms at our disposal, we develop a formalism by combining the three formalisms for describing Web services semantically and illustrate the strength of the formalism by some examples in Chapter 5. Roughly, the main idea behind the combination can be summarized as follows: We describe the behavior of Web services using polyadic π-calculus. **Since, π-calculus abstracts from the meaning of the involved objects in a process by considering objects as strings without any structure, we use description logic ontologies to describe the meaning of the involved objects. π-calculus offers the notion of guarded processes (processes that are executed only if certain condition is fulfilled), and π-calculus allows equality check on objects as the only condition. We extend the notion of guarded processes by using using boolean predicate symbols as condition expressions, for example to embed access checks.**

Having an expressive formalism at our disposal for describing web services semantically, we deal with matchmaking of Web service in Part III. We identify the need for two types of matchmaking approaches, namely *Goal driven Matchmaking* and *Simulation based Matchmaking*.

In goal driven matchmaking, Web service discovery is triggered by user's goals. In business scenarios user's goals are derived by his business policies and strategies. **The very first problem that needs to be solved and often ignored in related literature is, how a user's goal can be specified. We deal with this problem in Chapter 6.**

Goal based matchmaking is handled in Chapter 7. **We develop an algorithm that selects from a given set of Web services those Web services that offer the functionality desired by the user (satisfy the formula describing user's goal).** Since in general, a Web service is a complex process that can take different execution paths, a Web service may yield desired functionality only if certain conditions are fulfilled. These conditions can be semantic constraints on the data involved in the web service execution or security constraints. Our matchmaking algorithm yields not only yes/no answers but also in case of a positive answer also the conditions under which a Web service is a match. This is one of the main differences of our matchmaking approach from existing approaches.

In some scenarios, in which a business process or a software system already uses Web services, it may be desired to replace the use of a Web service by another Web service, that offers equivalent functionality. In such scenarios, goal based matchmaking may not be efficient, since it would require to derive all the goals that the old Web service fulfills. **A better approach is to directly compare two web service descriptions and check for equivalence. We deal with this problem in Chapter 8.**

In Part IV *Implementations and Applications*, we introduce implementation of our algorithms in Chapter 9. We present the detailed architecture of the implemented prototype in Sections 9.1 through 9.4.

The main components of the implemented prototype are *graphical user interface*, *reasoner*, *semantic Web services API*, *knowledge base management* and *conversion tools* for generating semantic descriptions from WSDL documents and HTML forms. In Sections 9.1 through 9.4, we describe all the components in detail. In order to show the practicability of our prototype, we evaluated it: (1) to show that our formalism is expressive enough for describing real Web services (2) to show our reasoning algorithms are fast enough for practical application and (3) to show that our implementation scales for a large number of Web service descriptions and user interfaces are intuitive.

In Chapter 10, we discuss how the techniques developed in this work were used in SemIPort (Section 10.1) and SESAM (Section 10.2) research projects. We also discuss possible scenarios in which our techniques and tools can be used like reasoning about market mechanisms and model driven architecture.

Finally, we conclude in Chapter 11 by summarizing our contributions and presenting problems that can be of future research interest.

2. Requirements

In the previous chapter, we have introduced Web service markets and motivated our work by identifying the need for automation of various steps that need to be performed in such a market. In this work, we focus on the matchmaking of Web services, so we start by identifying requirements for matchmaking that would lead to automation while finding and binding Web services in an application.

Transparency of a business process is crucial for cooperation and hence automation in copetitive markets. Consider for example a real world scenario, in which a person, say A, walking down a shopping street is halted by an employee, say B, of a promotion company. The person B asks the person A to fill up a form that includes personal information like credit card details, date of birth, address etc. Typically, the very first question that arises in the mind of person A is what is he going to get in return and what happens with all his personal information that he is supposed to fill in. If the person B does not give him the answer to these questions, it is very unlikely that person A will cooperate by filling up the form.

Similar is the case with Web services. Web services are technical interfaces to offer business processes. Their intended use is to incorporate them in larger systems that need the functionality of the business processes offered by them. Since, we are following the aim of automation, that is a user must be able to decide automatically whether a Web service offers the functionality that he needs at a certain point in his business process, the transparency of the offered business process is necessary. Otherwise, a user will have to contact the Web service providers and ask them for details, which considering the large number of Web services, requires a lot of manual effort.

Because of lack of support for transparency, current semantic matchmaking approaches fall short, since they match the interfaces of a Web service. As a result, the matched Web services are those web services that a user can in principle execute but there is no guarantee whether the matched Web services solve the task the user needs to solve in his larger system.

Requirement 1. Matchmaking should be functionality based rather than interface based.

In Section 2.1, we will identify which aspects of Web services should be considered while dealing with matchmaking. That is, which type of conditions a user may wish to specify in order to find desired Web services. Since matchmaking operates on descriptions of Web services, the degree of automation of the web service discovery process depends on the expressivity and the formal nature of the language used for describing Web services. For example, the limited expressiveness of WSDL for describing the functionality of Web services together with the informal nature of the UDDI model is the main bottleneck from the point of view of automation. Therefore, the type of conditions that a user may wish to specify for finding desired Web services directly pose some requirements on the formalism for describing Web services. However there are also some additional requirements for describing Web services. We will deal with the requirements for Web service description formalism in Section 2.2.

2.1. Requirements for Matchmaking

We first consider the use case that a software engineer is designing a business process and needs Web services for performing certain tasks in the business process. In order to find the desired Web services, he specifies the conditions that a Web service has to fulfill. We call such an approach *goal based matchmaking*. Goal driven approach deals with specifying the conditions on a desired Web service description and checking whether a Web service can fulfill the conditions. In business scenarios, these conditions are derived from high level business strategies that are typically not formalized or often not formalizable at all since they often depend on experiences and intuitions of managers. In this work, we do not deal with the problem of deriving conditions on desired Web services from business strategies and assume that such conditions are available. Existing semantic matchmaking approaches fall into the category of goal based matchmaking even if, as we have discussed earlier, they support very simple kind conditions (types of input and output parameters).

Often there are cases, in which a business process is already designed and a software engineer needs to replace an already incorporated Web service by another one without changing the overall behavior of the whole business process. In such scenarios, one seeks Web services that offer the same functionality as the Web service that needs to be replaced. In EAI scenarios, the systems within a large organization with many departments are connected together via Web services. Since the number of available Web services can become quite large, humans easily lose overview which Web services offer which functionalities. This often leads to the problem that new Web services are programmed per demand although there is already a Web service that offers the same functionality. In such scenarios, it is desirable to find duplicates and remove them in order to achieve better management and avoid fragmentation. In both use cases, goal based matchmaking may not be feasible, since the conditions fulfilled by the Web service to be replaced are often not available. Rather there is a need for a method for comparing two Web service descriptions directly for similarity. We call such an approach for directly comparing two web service descriptions *simulation or equivalence based matchmaking.*

Requirement 2. In order to cover as many use cases as possible, two kinds of matchmaking approaches are needed, namely goal driven and simulation driven.

In the rest of this section, we motivate aspects of Web services one often wishes to define constraints on in order to find desired Web services.

2.1.1. Constraints on Resources

A Web service operates on resources that can be real world objects or information objects. A Web service expects resources from the client and delivers resources to the client. A requester searching for a web service typically wishes to define constraints on the resources expected from and delivered by the Web service.

Consider for example a library that maintains a wish list of books where the library members can enter the ISBN of books they would like to have. On the basis of some ranking (e.g. number of members that wish the book) of the desired books the library system needs to place book orders at regular intervals. At the time of designing and implementing the library system the software engineers need to find book selling Web services. Suitable book selling services are those that accept an ISBN and deliver a book.

Requirement 3. It must be possible to define constraints on the types of input and output parameters of a Web service.

Considering the library scenario a bit deeper, we identify that the software engineers are actually interested in Web services that do not sell just any book, but exactly the book that is ordered by the library software. In this example a suitable Web service is one that delivers a book with the same ISBN as the one it obtains as input.

Requirement 4. It must be possible to define constraints on the values of the properties of the resources. In other words, it must be possible to define relationships among resources.

In realistic scenarios we must consider that there are many book selling Web services available and there is no global vocabulary for the domain of books that every Web service can or wants to use for describing the resources of his book selling Web service. As a result, we have to assume that in general Web service providers describe their Web services with their respective vocabularies independently of other Web service providers. Similarly, a requester needs a vocabulary that he understands to be sure that the constraints that he specifies capture the intended meaning.

Consider for example, that the output type of a book selling service is `Book` and a requester searches for Web services that have output type `Buch`. If the type `Book` as well `Buch` mean the set of books, then the Web service must be detected as a match. Note, that problem we are addressing at this point is not the translation from English to German terms. We have used English and German words for books only to illustrate the problem and the web service should be detected as a match even if the offered and requested types are `KL651` and `T2Z7Q` provided both of them mean the set of books.

Requirement 5. The formalism for specifying constraints on resources must be able to consider mappings among the vocabularies while checking the satisfiability of constraints.

When a Web service executes, it may cause changes in the knowledge state of the participants. For example, charging requester's credit card as a consequence of the order he has placed is performed by means of an update (setting the available credit amount to a lower value) in the database of the corresponding bank. While searching for desired Web services, a requester may be interested in specifying which effects he wishes to take place and which not.

Requirement 6. The formalism for specifying constraints must support specification of desired and undesired changes in the resources.

2.1.2. Security and Trust Constraints

Because of the vast heterogeneity of the available information, Web service providers and users, security becomes extremely important. Security related aspects are mostly classified in three categories, namely confidentiality, integrity and availability [Bis03, SC01, Den82].

In some cases, the access to a Web service may be restricted. We will discuss this issue in the next section in more detail. For this moment, we identify the following requirement.

Requirement 7. The framework should allow end users to check and prove his eligibility for a Web service.

When a Web service uses other Web services in order to achieve its functionality, it may be possible that a requester trusts the composite Web service but not its component Web services.

Requirement 8. The formalism should allow a requester to specify constraints on the set of parties involved in the execution of a Web service.

2.1.3. Constraints on Behavior

In the above examples, we assumed that the book selling Web service performs input and output activities in a particular order; first receiving a book order (input activity) and then sending the book to the client (output activity). Similarly, we assumed that the client, the library system, performs the activities of opposite polarity in the order such that the communication between a book selling Web service and the library system is possible. That is, the library system first sends the book order (output activity) and then receives the book (input activity).

So, a suitable Web service for the library system, that first sends an book order and then expects a book, must have the matching (opposite pole) communication pattern.

If the Web services had only one input activity and only one output activity, one may assume that a web service always perform the output activity after the input activity. As a consequence matching the communication patterns of the client and a Web service would be trivial.

However, in general this is not the case. Even if Web services abstract from implementation details, the process triggered by invoking a Web service may be very complex involving multiple interactions with the client or other Web services. Consider a book selling Web service that after receiving a book order, sends a confirmation to the client and expects a back confirmation. Only after receiving the back confirmation from the client, it sends the ordered book to the client. Even if the Web service receives an ISBN in the first input activity and output the book with the same ISBN in the last activity, it may not be suitable for the library system, if the library system does not foresee to receive an order confirmation and sending a back confirmation. In other cases, an opposite situation may occur. That is, a client system is ready to receive a book only after it has received an order confirmation and sent a back confirmation, but the Web service neither sends any order confirmation nor it waits for a back confirmation before sending the book to the client.

Requirement 9. In order to find Web services that can be incorporated in the client system it must be possible for the requester to specify constraints on the communication pattern of a Web service.

While the above requirement concerns the public communication protocol of the Web services (choreography), there are use cases, in which a requester may be interested in specifying constraints on the internal communication structure of a Web service (orchestration). Consider, for example a company internal Web service exposing a complex order process. It may be desired that the ordered good must be approved by at least two managers before it is bought. In other words, this means that in order to check whether the Web service is compliant to company's policies, one must be able to reason about the information flow and the order of activities that transport information.

Requirement 10. In order to find Web services that can be incorporated in the client system it must be possible for the requester to specify constraints on the internal communication of a Web service with other services.

2.1.4. Combination of different Types of Constraints

Finally, a requester may wish to combine different types of constraints. In the following, we give some examples of the combination of different types of constraints.

- **Combination of Effects and Behaviour** A user may wish to search for services that charge the credit card after the shipment of the ordered book.

- **Combination of Resources, Behaviour and Trust** A Web service may send some user specific information, e.g. his credit card number or his date of birth, to its component Web services that a requester (potential user) may not trust. In order to make sure that a Web service acts in compliance with his privacy policies, he wishes to specify constraints like "user's date of birth should not be sent to a party that the user does not trust"
- **Combination of Trust and Effects** The bank Web service should check that the book selling Web service is authorized (by the user) to cause an effect in the user's bank account.

Requirement 11. The formalism should allow to specify different types of constraints in a unifying and composable manner. That is, it must be possible to specify complex constraints from simpler constraints that may be of different types.

2.2. Requirements for Semantic Description of Web Services

In order to enable automatic matchmaking based on complex goals, the necessary information about various aspects of Web services must be available in machine interpretable form. Web services are a technology to offer business processes. Every Web service has an underlying business process which represents the actual functionality of the Web service. Such a business process may be simple like querying a weather database and returning the weather information to the requester or it can be very complex, for example complete order processing. In general, invoking a Web service can trigger a complex business process that involves multiple parties and complex interactions among them. In order to describe the functionality of a Web service semantically, the underlying business process has to be described.

While identifying requirements for a formalism for describing web services, we need to differentiate among (1) which information about Web services should be available in machine understandable form such that expressive matchmaking is possible (2) which information web service providers typically do not wish to describe (3) which information providers have to describe, e.g. because of legal issues, (4) which information providers would like to describe formally if an appropriate formalism were available.

The primary motivation of Web service providers for publishing machine interpretable descriptions of their Web services is to achieve better visibility for their Web services. Web service providers wish that the descriptions of their Web services are found by the requesters and their Web services are invoked often. So, if a formalism were available which is expressive enough for modeling properties of web services that the Web service providers currently have to describe in natural language, the Web service providers would describe their web services more expressively provided that the expressive modeling does not require too much manual modeling effort compared to the added value it brings. Note, that the current Web service description standard WSDL is not expressive enough to describe many aspects of Web services that the providers may wish to publish. Therefore, providers currently have to describe those aspects in natural language inside the `documentation` tag of a WSDL document.

2.2.1. Requirements for Resource Descriptions

A Web service execution involves exchange of resources among the participants. Thus describing the involved resources is an essential part of a Web service description, so that the parties can reason about the resources other parties expect from them and vice versa. However, providers typically do not wish to disclose descriptions of all the resources that they possess. The reason for this is that in many cases the resources of a provider are an important asset (e.g. the database of all the books available at Amazon). On the other side, a requester obviously wishes to have as much information as possible about the Web service.

Requirement 12. The formalism should not force Web service providers to disclose descriptions of all their resources base while still allowing reasoning about the involved resources.

Web service are described by actors acting independently of each other. Assuming the existence of a global vocabulary for resources is not realistic. Resource descriptions communicated among different parties must be understandable by them.

Requirement 13. The formalism for describing resources should support semantic interoperability.

Consider for example a book selling Web service that inputs the ISBN of a book and outputs an order confirmation. In this case, it is not enough to model the type of the output, since the order confirmation should be about the book the requester has ordered and not about any book. In order to describe such an output, one must be able to refer to the values of properties of inputs. However, the concrete resources that are involved in the execution of a Web service are known only at run time and not the design time.

Requirement 14. The formalism should support a way for referring to individuals that become known at run time

In our example, the ISBN of the book ordered by a user is obviously not known at the time of describing the Web service, but is provided by the client as input.

Requirement 15. It must be possible to describe output resources together with their relationship with input resources.

2.2.2. Requirements for Credentials

In order to achieve an infrastructure, in which clients can trust Web services on the basis of their credentials, it is not practical to abstract from the issuer of such credentials. Every cloth shop obviously advertises that it has great atmosphere and very friendly salespersons etc. So, people typically do not give too much importance to such self advertisements. Rather, they look for credentials issued to the shop by some other party, e.g. Stiftung Warentest. Similarly, if only Web service providers describe their Web services, the credentials of the Web services will be hardly of any practical use. Rather, there is a need for techniques in which parties different from the Web service providers issue credentials Web services and other parties can build trust in Web service on the basis of such credentials.

Requirement 16. The Web service description formalism should allow to model non-functional properties in a way that users can build trust in them and there is no need for a central control.

2.2.3. Requirements for Behavior Description

Considering our example of a Web selling service more deeply, we see that the above identified requirements does not cover the participating agents. That is, the formalism just covering the above requirements does not allow to describe that the output (book) is sent to the same actor (user) that has sent the input (ordered the book).

Requirement 17. The formalism must allow to identify actors and communication channels between them.

A Web service is an interface that is invocable via standard web protocols like HTTP. In general, the process triggered by invoking a Web service may involve many actors that may or may not be Web services. So, behavior of a Web service can not be defined without defining the non-Web processes. So, we need a formalism that can also describe a non-Web process.

From the technical point of view, the only difference between Web and non-Web processes is that they use different communication protocols. The actors involved in a Web service process communicate by exchanging messages. In case of Web services, the messages are exchanged via Web protocols like HTTP and are serialized in some web standard syntax like SOAP, whereas in case of non-Web processes the communication protocols can be “Phone”, “Fax” “Surface mail”.

In order to describe Web services semantically, we need to describe any type of invocable actors, those that can be reached via Web protocols and those that can be reached via non-Web protocols. Web services can then be seen as a special type of invocable actors, namely those that can be invoked via Web protocols.

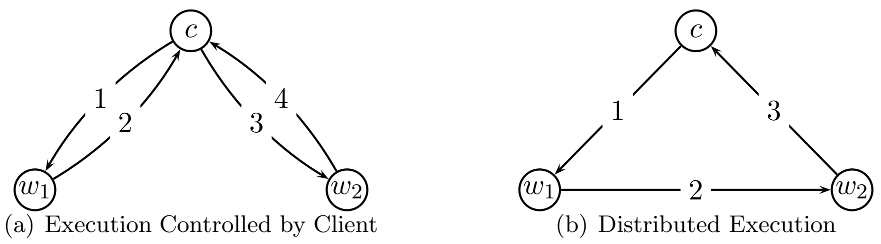


Figure 2.1.: Client-Server vs. Distributed Execution

Requirement 18. The formalism should support description of business processes that run in multiple environments.

We have already mentioned that providers typically do not wish to disclose the descriptions of all their resources. Similarly, they typically do not wish to disclose the identities of their business partners.

Requirement 19. The formalism should not force Web service providers to disclose the identities of their business partners while still allowing reasoning about the involved actors.

Centrally controlled vs. distributed execution

Consider a user who wishes to know the average temperature on the new year eve in Karlsruhe. Further consider, that there is no Web service that offers exactly the functionality the client is looking for. So, the client thinks of composing some appropriate Web services together and finds two Web services w_1 and w_2, where w_1 delivers a list of temperatures (one entry per year) for a given place and a given day and w_2 computes average of a given list of numbers.

To achieve his goal, the client c would specify a composite Web service having w_1 followed by w_2. In the current approaches, the data flow during the execution of the composite Web service is as follows (refer to Figure 2.1(a)).

1. The client inputs the values “Karlsruhe” and “31.12” for place and day respectively.
2. The client obtains the list of temperatures from w_1.
3. The client sends the list to w_2
4. The client obtains the average temperature from w_2 .

The execution of the newly composed Web service is controlled by the client. Although, the client is only interested in the end result (average temperature), it has to remember the list of temperatures returned by w_1 and send them unchanged to w_2. In other words, the Web services w_1 and w_2 do not communicate directly with each other, but via the client c. The problem becomes even clearer when w_1 and w_2 are provided by the same Web service provider and the client is physically far away from w_1 and w_2 and possesses only a narrow bandwidth internet connection or has only limited computing resources, e.g. if the client machine is a mobile phone.

A more efficient approach would be to specify and execute the composed Web service in such a way that w_1 sends the list of temperatures directly to w_2, which in turn calculates the average and sends it to the client. Such an approach would prevent the client from coordinating the Web services and having the overhead by compulsion of playing the role of a mediator unnecessarily. In such a distributed execution approach, that anyway reflects the autonomous functioning of the Web services more realistically, the communication between the parties will be as follows (refer to Figure 2.1(b)).

1. The client inputs the values “Karlsruhe” and “31.12” for place and day respectively for w_1.
2. The Web service w_1 sends the list of temperatures to w_2.
3. The Web service w_2 sends the average to the client c.

In order to enable scenarios like the one above, the Web service w_1 should allow the user to input the link to a Web service where the output of w_1 should be sent to (in our case w_2). Note, that technially it is not a problem to realize such Web services; in fact there are many Web services that allow such a flexibility. The challenge is rather to describe such a phenomenon formally.

Requirement 20. The formalism should support description of Web services that can communicate with other Web services as required by a user.

Updates

During the execution of a Web service, the resources of the involved parties may change. For example, when a book selling service charges the credit card of the user, the amount of money in the credit card account of the user is reduced. That is, there is an update in the data base of the corresponding bank. To model such updates, one needs to model the bank as another agent. The book selling Web service would send a request to the bank to reduce the amount from the credit card account of the user.

As another scenario, consider an English auction provided as web service. The Web service accepts a new bid only if it is higher than the current highest bid. The new bid then becomes the current highest bid. In order to guarantee the correct functionality of an English auction the value of the highest bid may not decrease. That is, the Web service may not update it to a smaller value. Furthermore, resources can be added to and deleted from the set of resources of a party.

Requirement 21. The formalism should allow to model the updates in the data bases of the involved parties.

Access Control

Not every Web service is intended to be used by everyone freely. In real business scenarios, providers need to restrict access to their Web services due to security reasons, economic reasons and legal reasons. Security related aspects are mostly classified in confidentiality, integrity and availability. Access control, which means the users must fulfill certain conditions in order to access certain functionality plays an important role in all three fields. For example, a student must show her library card to borrow a book from the university library. In context of confidentiality, it means that a student has access to the information relevant to only her own library account and thus can not know which other students have borrowed which books. In context of integrity, it means that a student may not change or cause a change in information relevant to the library account of another student. In context of availability, access control helps to prevent denial of service attacks that can take place if the access is uncontrolled. From the economic point of view, access control is necessary to for example ensure that only the users who have paid their annual membership are allowed to use the Web service. To understand why access control is important from legal perspective, consider again the example of our book selling services that on receiving ISBN of a book deliver the book to the client. Considering that there are books that are not allowed to be sold to minors by law, a book selling Web service is forced to check the age of the user in case a user orders such a book, before delivering him the book.

Requirement 22. The formalism should allow describing access control policies.

In order to achieve simulation based matchmaking we have the following requirement.

Requirement 23. Web services should be described in a way such that the descriptions are directly comparable.

Web service descriptions are often composed to form new Web services. In this work, our focus is not on developing algorithms for the automatic composition of Web services. However, in order to support matchmaking of composite Web services, there is a need for a formalism that allows to describe composite Web services. Furthermore, a formalism supporting the description of composite Web service would lend itself to automatic composition algorithms, since they need a way to “write down” the models they construct.

Requirement 24. The Web service description formalism should support the description of composite Web services.

2.3. Conclusion

In this chapter, we first motivated that for real business scenarios, the criteria for searching for suitable Web services are much richer than just their interfaces. We further identified that to cover a broader spectrum of use cases, there is a need for two types of matchmaking approaches, namely goal driven and simulation driven. We have identified that in order to achieve a functionality based matchmaking, three major aspects of Web services need to be dealt with namely, involved resources, credentials of the involved actors and the dynamic behaviour of a Web service. In Section 2.1 we identified various requirements for a formalism for specifying constraints concerning all the three aspects of Web services. These requirements are the basis of the goal specification formalism that we develop in Chapter 6. A matchmaking essentially operates on the description of Web services. In Section 2.2, we have identified requirements for a formalism for describing Web services. These requirments are the basis of the Web service description formalism that we develop in Chapter 5.

Part II.

Formal Description of Web Services

In this part, we present a formalism for describing Web services functionality. We have seen in Chapter 2, that in order to describe web service functionality we need to define invocable agents in general and view Web services as special type of invocable agents, namely those that can be invoked via Web protocols.

A web service is an interface that is invocable via standard Web protocols like HTTP. In general, the process triggered by invoking a web service may involve many actors, that may or may not be web services. The actors run concurrently to each other and communicate by exchanging resources. These resources can be real world objects or information objects. Through exchange of resources, the set of resources of the actors participating in an exchange (communication) may change. Furthermore, the actors may perform updates in their local set of resources. For example, they can add or remove resources to and from their respective set of resources or they can define relationships among the resources in their respective set of resources. The local operations performed by an actor are not observable by other actors, whereas the communication operations are observable for the actors that participate in the communication. Furthermore, we consider that the possiblity of communication between two actors may be restricted. That is, it is not necessarily the case, that every actor can communicate (exchange resources) with any other actor. Rather, only actors having certain credentials are allowed to communicate with another actor.

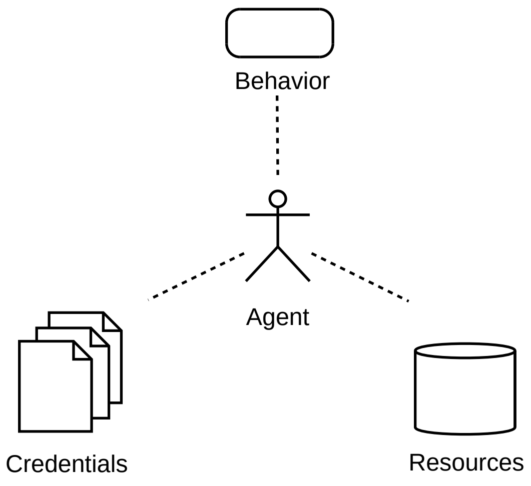


Figure 2.2.: Model of an Agent

In our formal model, we call such invocable actors *agents*. Figure 2.2 shows the model of an invocable agent that we consider in this work. Agents have a set of resources, a set of credentials and a dynamic behavior. Formally, we consider a finite set of agents $\mathcal{A}$.

- For each agent $A \in \mathcal{A}$, a finite set $\mathcal{R}_A$ describing the resources of agent A. We will deal with the description of resources in Chapter 3. In short, we will use expressive description logics to model resource schema of the agents as T-Box and the facts as A-Box.

- For each agent $A \in \mathcal{A}$, a finite set $\mathcal{C}_A$ of credentials that the agent A holds. We will deal with the modeling of credentials in Chapter 4. In short, we will use methods known from computer security to model non-functional properties of Web services as credentials such that users can build their trust in them and there is no need for a central control.

- For each agent $A \in \mathcal{A}$, a process $\mathcal{B}_A$ describing the dynamic behavior of A. We will deal with modeling of dynamic behaviour including access control policies and updates in Chapter 5. In short, we will use a process algebra to model the dynamic behaviour and introduce the notion of local operations to model the updates in the knowledge base (A-Box) of an agent. The objects communicated among different agents are given meaning by connecting them to description logic T-Boxes. Furthermore, we embed access control policies in the behaviour description by viewing them as conditions.

3. Modeling Resources

3.1. Introduction to Description Logics

Description logics are a family of knowledge representation formalisms closely related to first-order and modal logic. They are useful in various application fields, such as reasoning about database conceptual models, as the logical underpinning of ontology languages, for schema representation in information integration systems or for metadata management [BCM$^+$03].

In description logics, the important notions of a domain are described by concept descriptions that are built from concepts (unary predicates) and roles (binary predicates) by the use of various concept and role constructors. In addition to these concept descriptions, it is possible to state facts about the domain in the form of axioms. Terminological axioms make statements about how concepts or roles are related to each other, assertional facts make statements about the properties of individuals of the domain.

In the design of description logics, emphasis is put on retaining decidability of key reasoning problems and the provision of sound and complete reasoning algorithms. As the name suggests, Description Logics are logics, i.e. they are formal logics with well-defined semantics. Typically, the semantics of a description logics is specified via *model theoretic semantics*, which explicates the relationship between the language syntax and the models of a domain.

An interpretation consists of a domain and an interpretation function, which maps from individuals, concepts and roles to elements of the domain, subsets of the domain and binary relations on the domain, respectively. A description logic knowledge base consists of a set of axioms, which act as constraints on the interpretations. The meaning of a knowledge base derives from features and relationships that are common in all possible interpretations. An interpretation is said *to satisfy a knowledge base*, if it satisfies each axiom in the knowledge base. If there are no satisfying interpretations, the knowledge base is said to be *inconsistent*. If the relationship specified by an axiom holds in all possible interpretations of a knowledge base, the axiom is said to be *entailed* by the knowledge base.

Practical applications of DLs usually require concrete properties with values from a fixed domain, such as integers or strings, supporting built-in predicates. In [BH91], DLs were extended with concrete domains, where partial functions map objects of the abstract domain to values of the concrete domain, and can be used for building complex concepts. Further research on decidability, computational complexity, and reasoning

Name	Syntax	Semantics
Negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Conjunction	$(C \sqcap D)$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Existential Quantifier	$\exists R.C$	$\{d \in \Delta^{\mathcal{I}} \mid \exists e \in \Delta^{\mathcal{I}} \text{ with } (d,e) \in R^{\mathcal{I}} \wedge e \in C^{\mathcal{I}}\}$
Universal Quantifier	$\forall R.C$	$\{d \in \Delta^{\mathcal{I}} \mid \forall e \in \Delta^{\mathcal{I}} \text{ s.t. } (d,e) \in R^{\mathcal{I}} \Rightarrow e \in C^{\mathcal{I}}\}$

Figure 3.1.: $\mathcal{ALC}$−Constructors

algorithms for different DLs with concrete domains has influenced the design of the Ontology Web Language (OWL), which supports a basic form of concrete domains, so-called datatypes.

3.2. Basic Description Logic $\mathcal{ALC}$

$\mathcal{ALC}$ is the most widely accepted basic description logic, first introducted by Schmidt-Schauß and Smolka in [SSS91]. $\mathcal{ALC}$ stands for “Attributive Language with Complements”. Its semantics is based on an interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty set (the domain) and $\cdot^{\mathcal{I}}$ is an interpretation function. $\cdot^{\mathcal{I}}$ assigns each concept name C a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$ and each relation name R a subset $R^{\mathcal{I}}$ of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. It holds $\bot^{\mathcal{I}} = \emptyset$ and $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$. Figure 3.1 summarizes the syntax and semantics of the $\mathcal{ALC}$ constructors.

- For a concept C, the concept $\neg C$ denotes the set of all individuals that are in the domain $\Delta^{\mathcal{I}}$ but not in the set $C^{\mathcal{I}}$.
- For concepts C and D, the concept $C \sqcap D$ denotes the set of all individuals that are in the set $C^{\mathcal{I}}$ as well as in the set $D^{\mathcal{I}}$.
- For concepts C and D, the concept $C \sqcup D$ denotes the set of all individuals that are either in the set $C^{\mathcal{I}}$ or in the set $D^{\mathcal{I}}$ or in both the sets.
- For a relation R and a concept C, the concept $\exists R.C$ denotes the set of all individuals d for which there exists an individual e such that individuals d and e are related via the relation R, that is $(d,e) \in R^{\mathcal{I}}$, and the individual e belongs to the set $C^{\mathcal{I}}$.
- For a relation R and a concept C, the concept $\forall R.C$ denotes the set of all individuals d that are related via the relation R only to individuals e that are in the set $C^{\mathcal{I}}$.

Name	Syntax	Semantics
Concept Equivalence	$C \equiv D$	$C^{\mathcal{I}} = D^{\mathcal{I}}$
Concept Subsumption	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
Relation Equivalence	$P \equiv R$	$P^{\mathcal{I}} = R^{\mathcal{I}}$
Relation Subsumption	$P \sqsubseteq R$	$P^{\mathcal{I}} \subseteq R^{\mathcal{I}}$

Figure 3.2.: $\mathcal{ALC}$ T-Box Axioms

A description logic knowledge base usually consists of a set of axioms, which can be distinguished into terminological axioms (building the so-called TBox) and assertional axioms or assertions (building the so-called ABox).

A TBox is constituted by a finite set of terminological axioms which define subsumption and equivalence relations on concepts and relations. Figure 3.2 summarizes the $\mathcal{ALC}$ TBox axioms.

- An equivalence axiom of the form $C \equiv D$, whose left-hand side is an atomic concept is called a concept definition. The concept on the left-hand side of the equivalence axiom (C) is called defined concept. Semantically, $C \equiv D$ states that an individual either belongs to both the sets $C^{\mathcal{I}}$ and $D^{\mathcal{I}}$ or none of them.

- An equivalence axiom of the form $P \equiv R$ whose left-hand side is an atomic relation is called a relation definition. The relation on the left-hand side of the equivalence axiom (P) is called defined relation. Semantically, $P \equiv R$ states that a pair of individuals either belongs to both the sets $P^{\mathcal{I}}$ and $R^{\mathcal{I}}$ or none of them.

- A subsumption axiom of the form $C \sqsubseteq D$ for complex concept descriptions C and D are called (general) inclusion axiom. C is called a primitive concept, if it is atomic and occurs on the left-hand side of an inclusion axiom. A set of subsumption axioms of the form $C \sqsubseteq D$ where both C and D are atomic concepts is called a concept hierarchy. Semantically, $C \sqsubseteq D$ states that if an individual i belongs to the set $C^{\mathcal{I}}$ then i also belongs to the set $D^{\mathcal{I}}$.

- A set of subsumption axioms of the form $P \sqsubseteq R$ where both R and S are relations is called a relation or role hierarchy. Semantically $P \sqsubseteq R$ states that if a pair of individuals (d, e) belongs to the set $P^{\mathcal{I}}$ then (d, e) also belongs to the set $R^{\mathcal{I}}$.

Assertional axioms or Assertions introduce individuals, i.e. instances of a concept, into the knowledge base and relate individuals with each other and the introduced terminology. Figure 3.3 summarrizes the $\mathcal{ALC}$ ABox axioms. A finite set of such assertions is called an ABox.

- Individual Assertions express that an individual is member of a concept.

Name	Syntax	Semantics
Individual Assertion	$C(i)$	$i^{\mathcal{I}} \in C^{\mathcal{I}}$
Relation Filler	$R(a,b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$
Individual Equivalence	$i = j$	$i^{\mathcal{I}} = j^{\mathcal{I}}$
Individual Inequivalence	$i \neq j$	$i^{\mathcal{I}} \neq j^{\mathcal{I}}$

Figure 3.3.: $\mathcal{ALC}$ A-Box Axioms

- Relation Assertions express that two individuals are related with each other via a given relation.
- Individual equivalence assertions allows to express that two individuals are actually equivalent.
- Individual inequivalence axioms allow to express that two individuals are not the same.

Reasoning Tasks In the following, we summarize some of the most important reasoning tasks performed with $\mathcal{ALC}$.

- **Satisfiability:** A concept C is satisfiable wrt. a TBox $\mathcal{T}$, if there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that the interpretation of C is a non-empty set. That is, concept C is satisfiable if there is a model $\mathcal{I}$ of C such that $C^{\mathcal{I}} \neq \emptyset$.
- **Subsumption:** A concept C is subsumed by a concept D wrt. a TBox $\mathcal{T}$, if for every model $\mathcal{I}$ of $\mathcal{T}$ the interpretation of C is a subset of the interpretation of D. That is, $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in all models $\mathcal{I}$ of $\mathcal{T}$.
- **Equivalence:** Two concepts C and D are equivalent wrt. a TBox $\mathcal{T}$ if for every model $\mathcal{I}$ of $\mathcal{T}$ the interpretation of C is equal to the interpretation of D. That is $C \equiv D$, if $C^{\mathcal{I}} = D^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{T}$.
- **Disjointness:** Two concepts C and D are disjoint wrt. a TBox $\mathcal{T}$ if for every model $\mathcal{I}$ of $\mathcal{T}$ the interpretations of C and D are disjoint. That is, $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ for all models $\mathcal{I}$ of $\mathcal{T}$.
- **Instance check:** Given an individual a and a concept C, check whether a is an instance of C. That is $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for all models of the TBox $\mathcal{T}$ and ABox $\mathcal{A}$.

3.2.1. Description logics with Concrete Domains

One of the drawback of the description logic $\mathcal{ALC}$ is that all the terminological knowledge has to be defined on an abstract logical level. In many applications, one would like to be able to refer to concrete domains and predicates on these domains while defining concepts. For example the domain of integers or real numbers with predicates like equality, inequality etc.. In [BH91] a scheme for integrating such concrete domains into description logics is presented. In the following, we give the basic definitions regarding description logics with concrete domains and refer to [BH91] for more details.

Definition 1. A concrete domain $\mathcal{D} = (\mathsf{dom}(\mathcal{D}), \mathsf{pred}(\mathcal{D}))$ consists of a set $\mathsf{dom}(\mathcal{D})$ (the domain) and a set of predicate symbols $\mathsf{pred}(\mathcal{D})$. Each predicate symbol $P \in \mathsf{pred}(\mathcal{D})$ is associated with an arity n and an n-ary relation $P^{\mathcal{D}} \subseteq \mathsf{dom}(\mathcal{D})^{\mathsf{n}}$.

Definition 2. In addition to the concepts expressions in $\mathcal{ALC}$, $\mathcal{ALC}(\mathcal{D})$ allows expressions of the form $P(u_1, \ldots, u_n)$ as concept descriptions, where $P \in \mathsf{pred}(\mathcal{D})$ is an n-ary predicate and $u_1, \ldots u_n$ are feature chains. The semantics of such concepts expressions is defined as follows:

$$P(u_1, \ldots, u_n)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} | (u_1^{\mathcal{I}}(d), \ldots u_n^{\mathcal{I}}(d)) \in P^{\mathcal{D}}\},$$

where for $u = f_1 \ldots f_m$ a feature chain,

$$u^{\mathcal{I}}(a) = f_m^{\mathcal{I}}(f_{m-1}^{\mathcal{I}}(\ldots(f_1^{\mathcal{I}}(a)\ldots).$$

Definition 3. A concrete Domain $\mathcal{D}$ is called admissible iff (1) the set of its predicate names, $\mathsf{pred}(\mathcal{D})$ is closed under negation, (2) $\mathsf{pred}(\mathcal{D})$ contains a unary predicate $\top_{\mathcal{D}}$ for $\mathsf{dom}(\mathcal{D})$, and (3) the satisfiability problem for finite conjunctions over $\mathsf{pred}(\mathcal{D})$ is decidable.

Subsumption and satisfiability of $\mathcal{ALC}(\mathcal{D})$ is decidable iff the concrete domain $\mathcal{D}$ is admissible [BH91].

3.3. More Expressive Description Logic $\mathcal{SHOIN}(\mathbf{D})$

Before we formally define the syntax and semantics of the description logics underlying OWL, we here informally introduce the language constructs of the description logics $\mathcal{SHOIN}$. In particular we can build complex classes from atomic ones using the following constructors:

- $C \sqcap D$ (intersection), denoting the concept of individuals that belong to both C and D,
- $C \sqcup D$ (union), denoting the concept of individuals that belong to either C or D,
- $\neg C$ (complement), denoting the concept of individuals that do not belong to C,

- $\forall R.C$ (universal restriction), denoting the concept of individuals that are related via the role R only with individuals belonging to the concept C,
- $\exists R.C$ (existential restriction), denoting the concept of individuals that are related via the role R with some individual belonging to the concept C,
- $\geq n\,R$, $\leq n\,R$ (qualified number restriction), denoting the concept of individuals that are related with at least (at most) n individuals via the role R.
- $\{c_1, \ldots, c_n\}$ (enumeration), denoting the concept of individuals explicitly enumerated.

Based on these class descriptions, axioms of the following types can be formed:

- concept inclusion axioms $C \sqsubseteq D$, stating that the concept C is a subconcept of the concept D,
- transitivity axioms $\mathsf{Trans}(R)$, stating that the role R is transitive,
- role inclusion axioms $R \sqsubseteq S$ stating that the role R is a subrole of the role S,
- concept assertions $C(a)$ stating that the individual a is in the extension of the concept C,
- role assertions $R(a, b)$ stating that the individuals a, b are in the extension of the role R,
- individual (in)equalities $a = b$, and $a \neq b$, respectively, stating that a and b denote the same (different) individuals.

Example 1. In this example we build a small terminology about the bibliographic domain (based on a fragment of the SWRC ontology [SBH+05]. Consider the atomic concepts $Person$, $Publication$, $Article$ and $Book$, as well as a role $author$. We can build a terminology stating that

- *$Article \sqsubseteq Publication$, every article is a publication,*
- *$Book \sqsubseteq Publication$, every book is a publication,*
- *$Article \sqsubseteq \neg Book$, articles and books are disjoint, and*
- *$Publication \sqsubseteq \forall author.Person$, every author of a publication is a person.*

Further we can assert facts about individuals, *codd* and *relational_model*, stating that

- *$Article(relational\_model)$, the individual relational_model belongs to the concept Article,*

- $Person(codd)$, *the individual codd belongs to the concept* $Person$,
- $author(relational\_model, codd)$, *codd is the author of the relational_model,*
- *A_Relational_Model_of_Data_for_Large_Shared_Data_Banks* $\not\approx$ *codd, the individuals codd and relational_model are two different individuals*[1].

Datatypes In addition to the "abstract" classes considered so far, the $\mathcal{SHOIN}(\mathbf{D})$ description logic further supports reasoning with concrete datatypes, such as strings or integers. For example, it is possible to define a minor as a person whose age is less than or equal to 18 in the following way: $Minor \equiv Person \sqcap \exists age. \leq_{18}$. Instead of axiomatizing concrete datatypes in logic, $\mathcal{SHOIN}(\mathbf{D})$ employs an approach where the properties of concrete datatypes are encapsulated in so-called *concrete domains.*

3.3.1. The Family of OWL Languages

Traditionally, a number of different knowledge representation paradigms have competed to provide languages for representing ontologies, including most notably description logics and frame logics. With the advent of the Web Ontology Language (OWL), developed by the Web Ontology Working Group and recommended by World Wide Web Consortium (W3C), a standard for the representation of ontologies has been created. Adhering to this standard, we base our work on the OWL language (in particular OWL DL, as discussed below) and describe the developed formalisms in its terms.

In this section we introduce the OWL ontology language and Description Logics as its logical foundations. This introduction is to a great extent inspired by [HPSvH03] and [BHS04]. The OWL ontology language is based on a family of description logics languages. Within this family, three members have been standardized as sublanguages of OWL: OWL Full, OWL DL, and OWL Lite. These sublanguages differ in expressiveness, i.e. in their provided constructs and the allowed combinations of these constructs.

Several different syntaxes for OWL DL have been defined, including an abstract syntax as well as XML and RDF-based syntaxes. For our presentation, we use the traditional description logic notation since it is more compact. For the correspondence between this notation and various OWL DL syntaxes, see [HPS04b].

Before we discuss the description logic underlying OWL DL in detail, we provide a brief overview of description logics in general.

- OWL Full is the most expressive of the members of the OWL family. It has mainly been defined for compatibility with existing standards such as RDF (Resource Description Framework, [MM04]). Unfortunately, the satisfiability problem for OWL Full is undecidable and thus impractical for applications that require complete reasoning procedures.

[1]Please note that this fact is not trivial, as OWL does not take the unique names assumption.

- OWL DL is a sublanguage that was designed to regain computational decidability. It directly corresponds to the $\mathcal{SHOIN}(\mathbf{D})$ description logic, a decidable logic with NExpTime complexity. For $\mathcal{SHOIN}(\mathbf{D})$, practical reasoning algorithms are known, and increasingly more tools support $\mathcal{SHOIN}(\mathbf{D})$ or slightly less expressive languages.

- OWL Lite corresponds to the less expressive $\mathcal{SHIF}(\mathbf{D})$ description logic. Its complexity is known to be ExpTime. OWL Lite has received little interest, mainly due to the fact that even though its expressiveness is considerably restricted compared with OWL DL, reasoning with OWL Lite is still intractable.

- OWL DLP is a proper subspecies of OWL Lite [GHVD03, HHK+05]. Although it is only marginally less expressive then OWL Lite, it has polynomial complexity.

3.4. Query Languages

Query languages are an important tool in information management. They provide the means to access data that is managed using a particular data model. Surprisingly, despite the advance of OWL as an ontology language, the work on adequate query languages for OWL is still in its infancy. This is partly due to the fact that traditionally the focus of Description Logics has not been on managing large sets of assertional facts, but rather on reasoning over the terminological knowledge. Recently, there have been proposals for query answering over description logics that are based on a reduction to standard description logic reasoning problems. While being theoretically elegant it is impractical, as effectively for every individual in the knowledge base one needs to check whether it satisfies the query. On the other hand, there is a large body of work on query languages in database management. Some attempts have been made to build on this work in the context of query languages for RDF.

In [HBEV04] a comparative analysis for six RDF query languages has been presented . While some of these query languages are used in many systems as OWL query languages (simply treating OWL as an extension of RDF), such an approach has major drawbacks. The main problem lies in the fact that the semantics of RDF query languages is typically defined via a pattern matching over a particular RDF graph or a set of RDF triples, which corresponds to *one* particular model of the RDF knowledge base. Such a semantics is incompatible with that of description logics in the sense that an answer over a description logic knowledge base is supposed to hold in *any* model, of which there may be infinitely many.

A query language of particular importance is SPARQL, which is currently being standardized by the W3C as the standard query language for the Semantic Web. While SPARQL in principle follows the same pattern matching approach as other RDF query languages, in essence it only standardizes the *syntax* of a query language, but leaves the *semantics* up to the implementation. Although the lack of a well-defined semantics can be seen as a deficiency, it can also be seen as a feature, as it allows us to use SPARQL merely as a syntax carrier and to separately assign it a well-defined meaning based on grounding in descriptions logics. In the following we thus present the semantics of conjunctive queries over OWL DL ontologies and afterwards show how SPARQL can be used to encode such conjunctive queries. In our discussion we will further analyze how these formalisms meet generic criteria desired for any query language. We here review these criteria from [HBEV04]:

- *Expressiveness.* Expressiveness indicates how powerful queries can be formulated in a given language. Typically, a language should at least provide the means offered by relational algebra, i.e. be relationally complete. Usually, expressiveness is restricted to maintain properties such as safety and to allow an efficient (and optimizable) execution of queries.
- *Closure.* The closure property requires that the results of an operation are again elements of the data model. This means that if a query language operates on the graph data model, the query results would again have to be graphs.
- *Adequacy.* A query language is called adequate if it uses all concepts of the underlying data model. This property therefore complements the closure property: For the closure, a query result must not be outside the data model, whereas for adequacy the entire data model needs to be exploited.
- *Orthogonality.* The orthogonality of a query language requires that all operations may be used independently of the usage context.
- *Safety.* A query language is considered safe, if every query that is syntactically correct returns a finite set of results (on a finite data set). Typical concepts that cause query languages to be unsafe are recursion, negation and built-in functions.

Conjunctive Queries

Conjunctive queries are a popular formalism, well explored in database theory, capable of expressing the class of selection/projection/join/renaming relational queries. The vast majority of query languages for many data models used in practice fall into this fragment. Because conjunctive queries have been found useful in diverse practical applications, it is natural to use them as an expressive formalism for querying ontologies.

When talking about conjunctive queries, we need to distinguish between the query as a syntactic object and the *query mapping*, i.e. the function defined by a query under a specific semantics. Informally, a *conjunctive query* $Q(\mathbf{x}, \mathbf{y})$, is a conjunction of DL-atoms, i.e. atoms over DL-concepts and roles. $\mathbf{x}$ and $\mathbf{y}$ denote sets of *distinguished* and *non-distinguished* variables, respectively. Intuitively, the semantics of a conjunctive query is to ask for concrete individuals that are valid fillers for the distinguished variables, with the non-distinguished variables existentially bound.

Example 2. The following conjunctive query asks for the title of articles written by the individual codd:
$Q(t,x) : \mathit{Publication}(x) \wedge \mathit{title}(x,t) \wedge \mathit{author}(x, \mathit{codd})$

SPARQL

SPARQL was originally designed as RDF query language. As every OWL ontology can be encoded in an RDF graph, SPARQL can serve – at least syntactically – as an OWL query language.

The SPARQL query language is based on matching graph patterns. The simplest graph pattern is the triple pattern. In terms of OWL, every unary assertion $C(a)$ can be represented as a triple $(a\, \mathit{rdf{:}type}\, C)$ and every role assertion $R(a,b)$ can be represented as a triple $(a\, R\, b)$. Triple patterns in queries additionally may contain variables in the subject, predicate or object positions. Combining triples gives a basic graph pattern, where an exact match to a graph is needed to fulfill a pattern.

Queries are composed of two main building blocks, the SELECT or CONSTRUCT clause and the WHERE clause. The SELECT or CONSTRUCT clause identifies the variables to appear in the query results, they thus correspond to the distinguished variables of a conjunctive query. The WHERE clause identifies the triple pattern to match. Let us look at an example:

```
PREFIX  swrc: <http://swrc.ontoware.org/ontology#>

SELECT ?book WHERE {
  ?book rdf:type swrc:Book
}
```

The PREFIX clause simply allows to declare namespace abbreviations, in this case for the SWRC ontology. The SELECT clause specifies that the ?book variable should be returned, and the FROM clause states how an optional FILTER clause further allows to filter variable bindings, as in the following query that restricts the bindings of books to those that were published after the year 2000:

```
PREFIX  swrc: <http://swrc.ontoware.org/ontology#>

CONSTRUCT { ?book swrc:year ?year } WHERE {
  ?book rdf:type swrc:Book .
  ?book swrc:year ?year . FILTER (?year > 2000)
}
```

This query also demonstrates the use of the CONSTRUCT clause, which unlike the SELECT queries does not return tuples of variable bindings, but allows to specify triples that should be returned.

Multiple patterns can be joined with the `UNION` clause, as in the following query that asks for all publications that are either a book or a part of a book:

```
PREFIX  swrc: <http://swrc.ontoware.org/ontology#>

SELECT ?x WHERE {
  ?x rdf:type swrc:Book
  UNION
  ?x rdf:type swrc:InBook
}
```

Expressiveness. In terms of expressiveness, an important criterion is relational completeness, which requires that certain basic algebraic operations are supported, i.e. (i) selection, (ii) projection, (iii) cartesian product, (iv) difference and (v) union. These operations can be combined to express other operations such as set intersection, several forms of joins, etc.. SPARQL is not relationally complete. It does support the relational operations of selection (`FILTER` clause), projection (`SELECT` clause) and cartesian product (combination of triple patterns in the `WHERE` clause) as well as union (joining patterns with the `UNION` clause). However, it does not support difference, as there is no notion of negation in SPARQL. For example, it is thus not possible to ask for all publications that are not books.

Closure. The SPARQL language is closed, if one considers `CONSTRUCT` queries: The queries operate in triples, and also return triples as results. This is an important feature that allows to compose queries on the one hand, and to directly exchange query results as ontologies on the other hand.

Adequacy. As mentioned above, SPARQL is only partially adequate for querying OWL DL ontologies: SPARQL supports all elements of the OWL DL data model that are relevant for querying, but it also provides many features, which have not been presented above, that are not applicable to OWL DL ontologies (e.g. reification).

Orthogonality. Unfortunately, SPARQL is orthogonal only to a limited extent, as most of the operators have a very narrow scope of usage. For example, the `FILTER` operator can only be used inside of a `WHERE` clause.

Safety. The safety of the language is obviously depends on the semantics that is given to it. So far, SPARQL only defines the syntax of a query language. For the semantics that we define in the following, a sound and complete decision procedure exists, which implies safety.

Concluding, one can say that SPARQL at this time is an acceptable compromise for querying OWL ontologies, provided that is used as a syntax carrier for conjunctive queries, whose semantics is adequately defined separately using model theory.

3.5. Rule Languages for OWL

Although the description logics underlying OWL are very expressive ontology languages, they cannot be used for modeling particular forms of axioms. Specifically, description logic axioms are restricted to a tree-structure that disallows to model rule-like axioms commonly needed for complex data integration tasks that involve queries, views and transformations. This situation has lead to several proposals for the combination of ontology languages such as OWL with rule-based languages, which are currently controversially discussed [W3C05]. Just recently the W3C has chartered a working group for the definition of a standardized Rule Interchange Format (RIF)[2]. The Semantic Web Rule Language (SWRL, [HPS04a]) is currently the most prominent proposal for an extension of OWL DL with rules. From the Description Logics perspective, a main goal in the definition of a rule language for OWL is to retain decidability of the main reasoning problems. Unfortunately, SWRL is known to be undecidable. Recently, DL-safe rules, which can be seen as a syntactic fragment of SWRL, have been proposed as a decidable rule extension[MSS04].

SWRL - Semantic Web Rule Language

SWRL allows the use of datalog-like horn rules together with OWL axioms. These rules are of the form of an implication between an antecedent (body) and consequent (head). The intuitive meaning of a rule is that whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold. Both the antecedent (body) and consequent (head) of a rule consist of zero or more atoms. Atoms can be of the form $C(x)$, $R(x,y)$, $x \approx y$ or $x \not\approx y$, where C is an OWL DL concept, R is a role, and x,y are either variables, individuals or data values.

Example 3. The fact that a person is an expert on a topic if he has authored a publication on that topic, can be asserted with the following rule:
$expertOn(z,y) \leftarrow Publication(x) \wedge isAbout(x,y) \wedge author(x,z)$

A great advantage of SWRL is the tight integration with the existing OWL standards: A high-level abstract syntax is provided that directly extends the OWL abstract syntax. Further, an XML based syntax has been defined An extension of the OWL model-theoretic semantics provides a formal meaning for SWRL ontologies.

Example 4. The rule from Example 3 stating that a person is an expert on a topic if he has authored a publication on that topic, can be expressed in SWRL syntax in the following way:

```
<swrlx:Ontology swrlx:name=""
```

[2]http://www.w3.org/2005/rules/wg/charter

```
    xmlns:owlx="http://www.w3.org/2003/05/owl-xml#"
    xmlns:ruleml="http://www.w3.org/2003/11/ruleml#"
    xmlns:swrlx="http://www.w3.org/2003/11/swrlx#">

<ruleml:imp>
    <ruleml:_head>
       <swrlx:individualPropertyAtom  swrlx:property="expertOn">
           <ruleml:var>Z</ruleml:var>
           <ruleml:var>Y</ruleml:var>
       </swrlx:individualPropertyAtom>
    </ruleml:_head>
    <ruleml:_body>
       <swrlx:classAtom>
        <owlx:Class owlx:name="Publication" />
        <ruleml:var>X</ruleml:var>
       </swrlx:classAtom>
        <swrlx:individualPropertyAtom swrlx:property="isAbout">
            <ruleml:var>X</ruleml:var>
            <ruleml:var>Y</ruleml:var>
        </swrlx:individualPropertyAtom>
       <swrlx:individualPropertyAtom  swrlx:property="author">
           <ruleml:var>X</ruleml:var>
           <ruleml:var>Z</ruleml:var>
       </swrlx:individualPropertyAtom>
    </ruleml:_body>
</ruleml:imp>
</swrlx:Ontology>
```

DL-safe Rules

An unrestricted combination of description logics with rules leads to undecidability. Intuitively, the undecidability arises because adding rules to description logics causes the loss of any form of tree model property. DL-safe rules [MSS04] have been proposed as a subset of SWRL that allows to regain decidability. In DL-safe rules, the decidability is retained by restricting the interchange of consequences between the component languages, without restricting the component languages themselves. Specifically, concepts (roles) are allowed to occur in both rule bodies and heads as unary (binary) predicates in atoms, but each variable in a rule is required to occur in a body literal whose predicate is neither a DL-concept nor a DL-role. Intuitively, this restriction makes the logic decidable because the rules are applicable only to individuals explicitly introduced in the ABox.

3.6. Modeling Resources

For modeling resources, we use the description logic $\mathcal{SHOIN}(\mathcal{D})$ [BCM+03] with DL-safe rules [MSS04]. We associate with each agent $A \in \mathcal{A}$ a namespace NS_A. The set of resources $\mathcal{R}_A$ of the agent A are described as description logic individuals with names in the namespace NS_A. That is, we associate with each agent $A \in \mathcal{A}$ a description logic A-Box that represents the knowledge base of the agent A. The A-Box axioms $i = j$ (sameIndividual) and $i \neq j$ (differentIndividual) allow to describe explicitly whether two individuals i and j having different names describe the same object or different objects. Note, that in our setting two individuals belonging to different agents automatically have different names since they belong to different namespaces.

Now, from Requirement 12, we have that the agents typically do not wish to disclose their complete database. That is, they do not wish to disclose the description of each and every resource that they have. However, it is realistic to assume that they are not reluntant to disclose the schema for their resources. Note, that WSDL also makes this assumption and proposes to use XML Schema inside the `types` block.

From the Requirement 13 we have that the schemas must be interoperable. So, we describe the schemas as description logic T-Boxes, since description logics provide the $\sqsubseteq$ (subClassOf) relation to relate the concepts. A concept denotes a set of individuals and an individual i can be classified into a concept C either by directly specifying an axiom $C(i)$ or with the help of DL-safe rules.

So, in our formal model we consider a set of T-Boxes $\mathcal{T}$ and a T-Box association function $T : \mathcal{A} \to \mathcal{T}$ that associates with each agent $A \in \mathcal{A}$ a T-Box $T(A) \in \mathcal{T}$. Note, that $T(A)$ may have different namespace than NS_A. The reason for this is that it is not necessary to force an agent to use only his own vocabulary. Rather, the agents should be able to share vocabularies, which is not only more efficient for the agents, since they do not need to model the ontologies that already exist, but also leads to better interoperability of the information exchanged among the agents.

Requirement 14 demands the notion of variables for referring to individuals. The semantics of a variable cannot be captured with description logics, since description logics cannot reason about changing things. As a result, description logics do not direclty provide variables. However, as we will also see later, variables of an agent A can be described as individuals in the namespace NS_A and when a variable v is supposed to be bound to a value, another individual i , then we simply add an axiom $v = i$. Note, that even if modeling variables this way is rather easy and captures the correct semantics of a variable, well known semantic Web service approaches like OWL-S still cannot model variables correctly.

Having the notion of variables as individuals and that individuals can be related with each other with A-Box axioms, Requirement 15 is automatically fulfilled.

When resources are transported from one agent to another they need to be serialized. We assume a set of serialization types $\mathcal{S}$. A serialization type is similar to MIME types. We assume, that the sets of resources described by the concepts in the T-Boxes $\mathcal{T}$ are associated with a serialization type. That is, for every concept C in a T-Box $T \in \mathcal{T}$, there is a searialization type $S_C \in \mathcal{S}$.

4. Modeling Credentials

In Web the separation of roles in clients and providers is becoming narrower and one day it might completely disappear. People will use other Web services to provide their own, hence becoming client and provider at the same time. On one side they will compete with each other, on the other side they will collaborate with other.

In such a large and copetitive Web service market, where billions of Web services are traded, there will be more than one Web services that provide same functionality. This is analogous to having more than one shop where one can buy clothes. So, the non-functional aspects of Web services become very important, so that the potential users can decide which of the many alternatives they should chose. This is analogous to the phenomenon that despite having a large offer of clothes shops, most people usually go to their favourite shops because they like the atmosphere in the shop, or they find the salespersons friendly or they go to a shop someone who they trust in matter of clothes has recommended to them. Therefore, the credentials of the Web services, in the literature often referred to as non-functional properties will play an important role.

However, in order to achieve an infrastructure, in which clients can trust Web services on the basis of their credentials, it is not practical to abstract from the issuer of such credentials. Comparing with the clothes shops example, in which every cloth shop obviously advertises that the respective shop has great atmosphere and very friendly salespersons etc., if only Web service providers describe their Web services, the credentials of the Web services will be hardly of any practical use. Rather, there is a need for techniques in which parties different from the Web service providers issue credentials to Web services and parties can build trust in Web services on the basis of such credentials.

4.1. Role Based Access Control

Because of the vast heterogeneity of the available information, information providers and users, security becomes extremely important. Security related aspects are mostly classified in three categories, namely confidentiality, integrity, availability and accountability [Bis03, SC01, Den82]. Access control, which means the users must fulfill certain conditions in order to access certain functionality plays an important role in all three fields of security.

For example, a student must show his library card to borrow a book from the university library. In context of confidentiality, it means that a student has access to the information relevant to only his own library account and thus can not know which other students have borrowed which books. In context of integrity, it means that a student may not change or cause a change in information relevant to the library account of another student. In context of availability, access control helps to prevent denial of service attacks that can take place if the access is uncontrolled.

In an access control model, the entities that can perform actions in a system are called *subjects*, and the entities representing resources to which access may need to be controlled are called *objects*. Access control models used by current systems tend to fall into one of two classes: those based on *capabilities* and those based on *access control lists (ACLs)*. In a capability-based model, holding an unforgeable reference or *capability* to an object provides access to the object. For example, the possession of keys of a house grants access to the house to the holder. In an ACL-based model, a subject's access to an object depends on whether the identity of the subject is on a list associated with the object. For example, a bouncer checks the ID of a potential guest to see if his name is on the guest list, before allowing him to join a private party.

The concept of Role-Based Access Control (RBAC) began with multi-user and multi-application online systems pioneered in the 1970s. The central notion of RBAC is that permissions are associated with roles, and users are assigned to appropriate roles. This greatly simplifies management of permissions. Roles are created for the various job functions in an organisation and users are assigned roles based on their responsibilities and qualifications. Users can be easily reassigned from one role to another. Roles can be granted new permissions as new applications and systems are incorporated, and permissions can be revoked from roles as needed.

A role is properly viewed as a semantic construct around which access control policy is formulated. The particular collection of users and permissions brought together by a role is transitory. The role is more stable because an organisation's activities or functions usually change less frequently.

There are several distinct motivations for constructing a role. A role can represent competency to do specific tasks, such as a physician or a pharmacist. A role can embody authority and responsibility, e.g., project supervisor. Authority and responsibility are distinct from competency, e.g., a person can be competent to head several departments, but is assigned to head one of them. Roles can reflect specific duty assignments that are rotated through multiple users, e.g., a duty physician or shift manager. RBAC models and implementations should be able to conveniently accomodate all of these manifestations of the role concept.

The central notion of RBAC is that users do not have discretionary access to enterprise objects. Instead, access permissions are administratively associated with roles, and users are administratively made members of appropriate roles. This idea greatly simplifies management of authorization while providing an opportunity for great flexibility in specifying and enforcing enterprise- specific protection policies. Users can be made members of roles as determined by their responsibilities and qualifications and can be easily reassigned from one role to another without modifying the underlying access structure. Roles can be granted new permissions as new applications and actions are incorporated, and permissions can be revoked from roles as needed.

The principal motivations behind RBAC are the ability to articulate and enforce enterprise-specific security policies and to streamline the typically burdensome process of security management. RBAC represents a major advancement in flexibility and detail of control from the present-day standards of discretionary and mandatory access control. In many enterprises within industry and civilian government, end users do not "own" the information for which they are allowed access as is often assumed by traditional discretionary access control schemes. For these organizations, the corporation or agency is the actual "owner" of system objects and discretion on the part of the users may not be appropriate. With role-based access control, access decisions are based on the roles individual users have as part of an organization. As such, RBAC is often described as a form of nondiscretionary access control in the sense that users are unavoidably constrained by the organization's protection policies. In non-classified environments, such policies are not focused on solving the multi-level security problem as is assumed within the existing standard for nondiscretionary access control. In contrast, RBAC allows for the specification and enforcement of a variety of protection policies which can be tailored on an enterprise-by- enterprise basis. The policies enforced in a particular stand-alone or distributed system are the net result of the precise configuration of the various components of RBAC. This RBAC framework provides administrators with the capability to regulate who can perform what actions, when, from where, in what order, and in some cases under what relational circumstances.

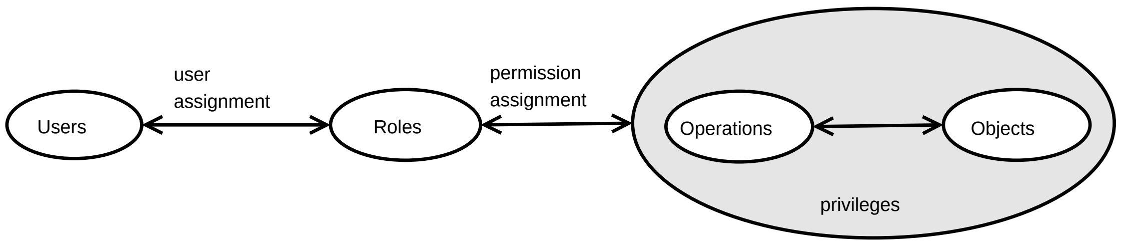


Figure 4.1.: Core Model of Role Based Access Control

From a functional perspective, RBAC's central notion is that of operations representing actions associated with roles and users that are appropriately made members of roles. The relationships between users, roles, and operations is depicted in Figure 4.1. As shown in Figure 4.1, the use of double arrows indicate a many-to-many relationship. For example, a single user can be associated with one or more roles, and a single role can have one or more user members. Roles can be created for various job positions in an organization. For example, a role can include Teller or Loan Officer in a bank, or Doctor, Nurse, or Clinician in a hospital. The operations that are associated with roles constrain members of the role to a specified set of actions. For example, within a hospital system the role of Doctor can include operations to perform diagnosis, prescribe medication, and order laboratory tests; the role of Researcher can be limited to gathering anonymous clinical information for studies; and the role of Social Worker may be to review patient profiles to flag possible suicidal patients or determine possible abuse cases.

The association of operations with roles within an enterprise can be in compliance with rules that are selfimposed. For example, a health care provider may decide that the role of Clinician must be constrained to post only the results of certain tests rather than distribute them where routing and human errors can result in a violation of a patient's right to privacy. Operations can also be specified in a manner that can be used in the demonstration and enforcement of laws or regulations. For example, a nurse can be constrained to adding a new entry to a patient's history of treatments rather than being generally able to modify a patient record. A pharmacist can be provided with operations to dispense, but not to prescribe, medication.

The core model of RBAC consists of following entities:

- A set of *users* U, a set of *roles* R, a set of *operations* A and a set of *objects* O
- A relation *user assignment* UA, $UA \subseteq U \times R$
- A set of *privileges* $P = 2^{A \times O}$
- A relation *permission assignment* PA, $PA \subseteq P \times R$
- Role hierarchy $RH \subseteq R \times R$, a partial order on the set of roles R
- Inhertitance relation $r_1 \succeq r_2$ if and only if
  - all privileges of r_2 are also privileges of r_1
  - all users of r_1 are also users of r_2

4.2. Authorization based Access Control with SPKI/SDSI

Current access control is mostly based on authentication, which requires central control (registration) and proof of identity. Thus identity based authentication leads to certain limitations regarding the spontaneity and privacy and hence is not always desired. Therefore, we propose authorization based access control rather than authentication based access control of Web services. Authorization based access control also includes authentication, but here the authentication is based on public keys and not on identities.

In most systems access control is based on identity based authentication, which means that the users must be known to the provider, for example via registration.

Since Web is an open, distributed, decentralized, dynamic and interoperable environment, in which Web services must be offerable and usable by anyone spontaneously and

centralized access control	**distributed access control**
closed organization	open environment
immediate inspection	trust evaluation whether digital document presumably capture properties in real world
established procedures	protocols under development
trust based on personal acquintances, law enforcement, ...	trust based on appropricate protocols
identity based	public key based
authentication by "personal peculiarities" (passwords, biometrics, ...)	authentication by "proof of secret key" (challenge response, biometrics, ...
real world directly observable	real world "hidden", only documents visible

Table 4.1.: Central vs. distributed access control

dynamically and users do not always wish to disclose their identities, security infrastructures that require registrations or any other central controlling components are not suitable. For instance, a server may receive request not just from the local community of users, but also from remote, previously unknown users. The server may not be able to authenticate these users or to specify authorizations for them (with respect to their identity). The traditional separation between authentication and access control cannot be applied in this context, and alternative access control methods should be devised. Early approaches departing from this assumption proposed associating authorizations with keys rather than users' identities. This family of trust management systems (e.g. PolicyMaker [BFL96], Keynote [BFIK99], REFEREE [CFL+97]) use credentials to describe specific delegation of trusts among keys and to bind public keys to authorizations. While these approaches provide an interesting framework for reasoning about trust among unknown parties, assigining authorizations to keys may make authorization specifications difficult to manage.

An alternative promising approach is represented by the use of digital certificates (or credentials), representing statements certified by given entities (e.g. certification authorities), which can be used to establish properties of their holder (such as identity, accreditation, or authorizations) [HFPS99]. Credential based access control makes the access decision of whether or not a party may execute an access dependent on properties that the party may have, and can prove by presenting one or more certificates (authorization certificates [BFL96] being a special kind of them). Several researchers have addressed the problem of establishing a Public Key Infrastrucure (which is at the basis of credential-management); managing credentials and credential chains and developing strategies for automated trust-negotiation, that is for determining the credentials to be required and released when interacting with other parties.

Credentials are digitally signed documents, which can be transmitted over untrusted channels like the Web, see e.g., [BK02, Cha85]. Credentials assert a binding between a principal and some property. A principal represents a user and is identified by his public key. The meaning of a stated property may be a granted capability for a service, an identity or a non-identifying characteristic of a user like e.g., a skill. For further related work refer to [Bra00, BFIK99, Sam02]. The credential-based public key infrastructure SPKI/SDSI [EFL$^+$99a, EFL$^+$99b] allows each principal to issue credentials. Unlike other public key infrastructures, SPKI/SDSI requires no central certification authority. Thus, each Web service provider can issue and trust credentials independent of other service providers and may even define her own trust structure. A Web service provider, acting as a verifier, can locally and autonomously decide whether access to her service should be granted or not. Access decisions are based on the provider's interpretation of a user's capabilities or characteristics given by shown SPKI/SDSI certificates. Furthermore, users can request Web services spontaneously without registering themselves with the individual Web service providers. Therefore SPKI/SDSI credentials are more suitable than the classical authentication based systems for specifying access control policies in the Semantic Web.

In 1996 Lampson and Rivest [RL96] proposed a new public-key infrastructure, called "a Simple Distributed Security Infrastructure", abbreviated "SDSI". Its most interesting feature is probably its decentralized name space. In SDSI the owner of each public key can create a local name space relative to that key. These name spaces can be linked together in a flexible and powerful manner to enable chains of authorization and define groups of authorized principals. Concurrently, Carl Ellison, Bill Frantz, Brian Thomas an, Tatu Ylonen and others developed a "Simple Public Key Infrastructure" abbreviated "SPKI" which emphasized exceptional simplicity of design and a flexible means of specifying authorizations. The SDSI and SPKI efforts were both motivated in part by the preceived complexity of the X.509 public-key infrastructure, and also by its perceived lack of power and flexibility. In 1997 the SDSI and SPKI efforts were merged: the resulting synthesis has been called "SPKI/SDSI".

In SPKI/SDSI there is a *local name space* associated with every public key. There are no *global names* in SPKI/SDSI. A local name is a pair consisting of a public key and an arbitrary identifier. A public key can sign statements (certificates) binding one of its local names to a value. Values can be specified indirectly in terms of other names, so the name spaces can become linked and interdependent in a flexible and powerful manner.

In SPKI/SDSI, all pricipals are represented by their public keys. A principal is an individual, process, or active entity whose messages are distintively recognizable because they are digitally signed by the public key that represents them. Let $\mathcal{K}$ denote the set of public keys and K, K_A, K_B, K_1, K_2 etc. specific public keys. Because the most important function of a name is to serve as a mnemonic handle of some user, it is important that users be able to create names rather than freely using well-chosen identifiers. An identifier is a word over some given standard alphabet, for examples, `A`, `B`, `Alice`, `Bob`.

Each public key has its own associated name space; there is no global name space or even a hierarchy of name spaces. SPKI/SDSI does not require a "root" or "root key"; it can be built "bottom-up" in a distributed manner from a collection of local name spaces.

Definition 4. A local name is a sequence of length two consisting of a public key K followed by a single identifier

Typical local names might be "K `Alice`" or "K `project-team`". Here, K represents an actual public key.

The local name "K `A`" belongs to the local name space of key K. Let $\mathcal{N}_L$ denote the set of all local names and $\mathcal{N}_L(K)$ denote the local name space of key K.

Definition 5. An extended name is a sequence consisting of a key followed by two or more identifiers.

Typical extended names might me "K `Alice mother`", "K `microsoft engineering windows project-mgr`" or "K `UNIKA personnel-committee`".

Let $\mathcal{N}_E$ denote the set of all extended names and $\mathcal{N} = \mathcal{N}_L \cup \mathcal{N}_E$ denote the set of all names. Furthermore, let $\mathcal{N}_E(K)$ denote the set of extended names beginning with key K and let $\mathcal{N}(K)$, called name space of key K, denote the set of all names (local or extended) beginning with key K.

The SPKI/SDSI expressions are called "terms"; intuitively, a term is something that may have a value. In SPKI/SDSI values are always set of keys.

Definition 6. A term is either a key or a name. Let $\mathcal{T} = \mathcal{K} \cup \mathcal{N}$ denote the set of all terms.

SPKI/SDSI has two types of certificates *name certificates*, which provide a definition for a local name, and *authorization certificates*, which confer authorization on a key or a name. Compared to X.509 public-key infrastructure schemes [FB97], an SPKI/SDSI name certificate is comparable to an "ID certificate" and to some forms of "attribute certificate" that conveys authorization.

Name Certificates

A name certificate provides a definition of a local name (e.g. K `A`) belonging to the issuer's (e.g. K's) local name space. Only key K may issue (that is, sign) certificates for names in the local name space $\mathcal{N}_L(K)$. A name certificate C is a signed four-tuple $(K, \mathtt{A}, S, V)$

- The *issuer* K is a public key; the certificate is signed by K.

- The *identifier* $\mathtt{A}$ (together with the issuer) determines the local name "$K\ \mathtt{A}$" that is being defined; this name belongs to the local name space $\mathcal{N}_L(K)$ of key K. It should be noted, that name certificates only define local names (with one identifier); extended names are never defined directly, only indirectly.

- The *subject* S is a term in $\mathcal{T}$. Intuitively, the subject S specifies a new additional meaning for the local name "$K\ \mathtt{A}$".

- The *validity specification* V provides additional information allowing anyone to ascertain if the certificate is currently valid, beyond the obvious verification of the certificate signature. Normally, the validity takes the form of a validity period (t_1, t_2): the certificate is valid from time t_1 to time t_2, inclusive.

In SPKI/SDSI the value of a term T is defined relative to a set $\mathcal{C}$ of certificates. Let $\mathcal{V}_\mathcal{C}(T)$ denote the value of a term T with respect to a set $\mathcal{C}$ of certificates (cf. Definition 6). The value of a term is a set of public keys, possibly empty. For any public key K the value of K $\mathcal{V}_\mathcal{C}(K) = \{K\}$ for any set $\mathcal{C}$ of certificates. A local name has value that is a set of public keys; this value may be the empty set, a set containing a single key, or a set containing many keys. This value is determined by one or more name certificates. A local name, such as K `Alice`, need not have the same meaning as the local name K' `Alice` when $K \neq K'$; the owner of key K may define K `Alice` however he wishes, while the owner of key K' may similarly but independently define K' `Alice` in an arbitrary manner. A name certificate $C = (K, \mathtt{A}, S, V)$ (intuitively, defining local name $K\mathtt{A}$ in terms of subject S) should be understood as a signed statement by the issuer asserting that

$$\mathcal{V}_\mathcal{C}(K\mathtt{A}) \supseteq \mathcal{V}_\mathcal{C}(S) \qquad (4.1)$$

that is, every key in the value $\mathcal{V}_\mathcal{C}(S)$ of subject S is also a key in the value $\mathcal{V}_\mathcal{C}(K\mathtt{A})$. One name certificate does not invalidate others for the same local name; their effect is cumulative. That is why the above equation says $\mathcal{V}_\mathcal{C}(K\mathtt{A}) \supseteq \mathcal{V}_\mathcal{C}(S)$ and not $\mathcal{V}_\mathcal{C}(K\mathtt{A}) = \mathcal{V}_\mathcal{C}(S)$. Each additional name certificate for $K\mathtt{A}$ may add new elements to $\mathcal{V}_\mathcal{C}(K\mathtt{A})$. A local name in SPKI/SDSI thus without any special fanfare, represent a *group* of public keys.

Although a name certificate $C = (K, \mathtt{A}, S, V)$ has the explizit function of providing a definition for the local name $K\mathtt{A}$, it also gives meaning to related extended names. If the subject S of a name certificate is an extended name, then it is necessary to have a definition for the value $\mathcal{V}_\mathcal{C}(S)$ in order to interpret Equation 4.1.

The value of an extended name is implied by the values of various related local names as follows:

Definition 7. The value of an extended name $K\mathtt{A}_1\mathtt{A}_2 \ldots \mathtt{A}_n$ is defined recursively for $n \geq 2$ as:

$$\mathcal{V}_\mathcal{C}(K\mathtt{A}_1\mathtt{A}_2 \ldots \mathtt{A}_n) = \{K'' : K'' \in \mathcal{V}_\mathcal{C}(K'\mathtt{A}_n) \qquad (4.2)$$

for some $K' \in \mathcal{V}_{\mathcal{C}}(K\mathtt{A}_1\mathtt{A}_2 \ldots \mathtt{A}_{n-1})$. An equivalent definition is:

$$K\mathtt{A}_1\mathtt{A}_2 \ldots \mathtt{A}_n = \bigcup_{K' \in \mathcal{V}_{\mathcal{C}}(K\mathtt{A}_1)} K'\mathtt{A}_2\mathtt{A}_3 \ldots \mathtt{A}_n$$

Definition 8. For any term T, $\mathcal{V}_{\mathcal{C}}(T)$ is defined to be the smallest set of public keys that is consistent with any constraints for the form of equation 4.1 and 4.2 implied by the name certificates in $\mathcal{C}$.

Authorization Certificates

The function of an authorization certificate is to grant or delegate a specific authorization form the issuer to the subject. An authorization certificate C is a signed five-tuple (K, S, d, T, V):

- The *issuer* K is a public key, which signs the certificate. The issuer is the one granting a specific authorization.
- The *subject* S is a term in $\mathcal{T}$. The public keys in $\mathcal{V}_{\mathcal{C}}(S)$ are receving the grant of authorization.
- The *delegation bit* d, if true, grants each key in $\mathcal{V}_{\mathcal{C}}(S)$ permission to further delegate to others the authorization it is receiving via this certificate.
- The *authorization specification* or *authorization tag* T specifies permission or permissions being granted. For example it may specify right to read a certain file, or login to a particular machine.
- The *validity specification* for an authorization certificate is the same as that of a name certificate.

SPKI/SDSI authorization certificates integrate smoothly with SPKI/SDSI name certificates. The name certificates are used to give useful symbolic names to individual keys or groups of keys, and the authorization certificates can be used to authorize those keys or groups of keys for specific operations.

4.3. Modeling Credentials

Web service providers specify access control policies to restrict access to their Web services. It turned out, that since the Web is an open, distributed and dynamic environment, in which a central controlling instance cannot be assumed, capability based access control is most suitable for this purpose. However, since practically every participant can certify capabilities defined in his/her own terminology, determining the semantics of certified capabilities and the trustworthiness of certification authorities are two major challenges in such a setting. In this section we show,

- how certification authorities and their certification policies can be modeled semantically
- how Web service providers can specify and check the consistency of their access control policies and
- how end users can check automatically, whether they can be granted access to a Web service.

Semantic Web services promise dynamic business that can be offered and carried out in the Web. In such an open environment, access control plays an important role. Roughly, access control means that the users must fulfill certain conditions in order to gain access to certain functionality. Access control is not only important from the security point of view, but also from a legal point of view. In some cases, even if a Web service provider does not wish to restrict access to his Web service, he may be forced by law to do so. As we have discussed earlier capability based access control is much more suitable for semantic Web services than identity based access control methods. In such systems, users prove their legitimacy for access by showing an appropriate set of credentials stating their capabilities. A Web service provider can verify, whether the shown set of proved credentials satisfy all the required constraints. In case, it does not, a Web service provider will not allow the user to access his Web service. However, there are still several open issues that need to be resolved in capability based access control in open environments. We identify three major roles a participant can play in such a setting, namely *user*, *Web service provider* and *certification authority*.

Users

Users want to access Web services. In case of restricted access they must prove their eligibility, e.g. by showing appropriate certificates (in a capability based setting). In some cases a user wishes to automatically infer whether he/she can fulfill the conditions imposed by the access control policy of a Web service. In another scenario, a user may compose some Web services that may belong to different administrative domains. He then wishes to know the access control requirements of the composed system. To support such use cases, the access control policies as well as the credentials have to be specified formally.

Web Service Providers

Web service providers want to restrict access to their services to only eligible users. For this, they specify and enforce access control policies. If these are specified in terms of capabilities that need to be proved by certificates, then the following questions need to be addressed:

- What is the meaning of the terminology used by CAs in their certificates?
- What is the certification policy of the CA that has issued the certificate?
- Which credentials of which certification authorities shall be trusted? On what grounds are the CAs authorized to certify the respective property?
- Are the credentials still valid or have they been revoked?
- Considering the certification policies of the CAs, is the specified ACP consistent with certain conditions and laws?
- Is the ACP satisfiable? That is, is it possible for any user at all to fulfill all the conditions and thus gain access to the Web service?

Certification Authorities

Certification authorities certify users certain properties by issuing certificates. Each certification authority defines its own terminology that it uses in its certificates, for example, the names of certifiable properties and the relation between certifiable properties.

Currently, certification authorities specify their certification policies explicitly in documents that are readable only for humans (see e.g. [Kel04] for an extensive list of certification authorities). These documents are meant to be read by the service providers before they define the access control policies for their Web services. In this section we show how certification authorities can specify their certification policies in a machine readable form and how they can publish the policy as well as their certification context, e.g. which properties have been certified to them. This approach has several advantages: When specifying their access control policies, Web service providers can use these certification policies to automatically check, whether the specified ACP fulfills certain (legal as well as self imposed) requirements. When verifying the access eligibility of a particular user on the grounds of a presented set of credentials, the Web service provider can make use of the published certification context to check the validity of the presented credential chains. Though revocation of certificates has been addressed in several papers, it is still a controversial issue (e.g. in the context of delegation) and important real life applications such as *tls* or *ssl* simply ignore the possibility of revocation. Our approach can support the CAs in the implementation and enforcement of revocation of certificates.

We begin with a description of an example scenario that serves as a running example in Section 4.3.1. In Section 5.2.3, we introduce a simple and novel approach, how a certification authority can specify explicitly and with machine understandable semantics which terminology it uses in the certificates that it issues and which relationships exist among the certified properties. In Section 5.2.3, we show how Web service providers can specify the access control policies of their Web services based on their knowledge about certification authorities.

4.3.1. Scenario

We now describe an example scenario that will be used throughout this chapter as a running example. We consider the following organizational entities: (1) Outdoor Shop (*OutShop*), (2) Wildlife foundation (*WLF*), (3) Forest Department (*FD*), (4) Local Trekking Club (*LTC*).

The Outdoor Shop *OutShop* acts as Web service that maintains a list of approved trekking guides. To register as a guide, a user must be above 25 years old, have good trekking experience as well as knowledge in first aid. *OutShop* trusts the Wildlife foundation *WLF* and the Forest Department *FD* as well as their delegates to certify guidance experience.

The Wildlife Foundation *WLF* issues certificates about guidance experience *app_guide*. In its certification policy it does not relate this properties to any other properties. Furthermore, it allows the local Trekking Club *LTC* to act on its behalf and issue guidance experience certificates.

The Forest Department *FD*, too, issues certificates about guidance experience. However, it issues such certificates only to people who are above 25 years old and have knowledge in first aid (certified e.g. by the Red Cross). This restriction is specified in *FD*'s certification policy.

4.3.2. Specification of Certification Policies

In the Web practically every participant can act as *Certification Authority* (CA) autonomously and independent of other CAs. It is therefore unrealistic to assume that there is a global vocabulary of properties that every CA uses (in our example scenario for instance, the meaning of `app_guide` certified by the Forest Department is different from the meaning of `app_guide` certified by the Wildlife Foundation). Further, there may exist logical relationships among certification authorities (e.g. delegation) and among the properties they certify (e.g. inclusion/exclusion of other properties)(refer to figure 4.2). For example the certification of the property `app_guide` is delegated from *WLF* to the local trekking club *LTC*, and the certification of `app_guide` by the Forest Department implies, that the grantee is above 25 years old and is experienced in first aid.

A signed set of properties that a certification authority certifies together with their relationships and dependencies with other properties (possibly certified by other CAs) is called a *certification policy* of the certification authority. A CA can specify its certification policy and publish its trustworthiness in the Semantic Web with machine understandable semantics. To do so, the CA specifies

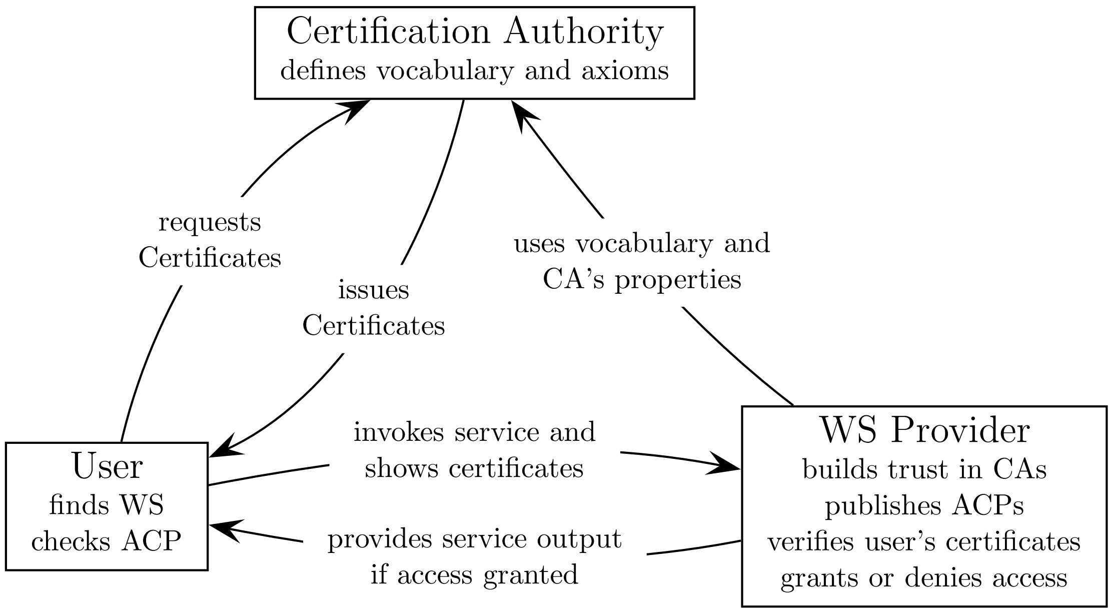

Certification Authority
defines vocabulary and axioms
requests Certificates
issues Certificates
uses vocabulary and CA's properties
User
finds WS
checks ACP
invokes service and shows certificates
provides service output if access granted
WS Provider
builds trust in CAs
publishes ACPs
verifies user's certificates
grants or denies access

Figure 4.2.: Interaction among Parties (logical roles)

- its certification context in terms of certified properties. This ground can help a Web service provider to decide about the trustworthiness of the certification authority.
- the terminology it uses in the certificates that it issues. These certificates are referenced by a Web service provider in the specification of an access control policy.
- relationships among the properties, that it certifies and any axioms about them.

A certification authority is associated with a set of properties that it possesses and a set of properties that it certifies. In our example, the certification authority *LTC* possesses the property that it is delegated to certifiy property *app_guide* to users. This property is certified by the *WLF*. In order to talk about such properties and certification authorities more formally, we model the following concepts.

$$\begin{aligned}\texttt{Property} &\sqsubseteq \top\\ \texttt{CA} &\sqsubseteq \top \sqcap \exists\texttt{possesses.Property} \sqcap \exists\texttt{certifies.Property}\end{aligned}$$

Delegation

Logical relationships between various CAs and the properties they certify can be specified through axioms, in particular class hierarchies over relevant instances of `Property`. Note, that the concrete properties used in an axiom by a certification authority C do not necessarily need to be the properties that are certified by C. Subclass relationship between two classes of different certification authorities can be used for specifying delegation structures. Consider for example, certification authorities C_1 and C_2 that certify properties P_1 and P_2, respectively. By defining the axiom $P_1 \sqsubseteq P_2$, the certification authority C_2 states, that anyone possessing the property P_1 also possesses the property P_2. In other words, the certification authority C_2 delegates the certification of the property P_2 to the certification authority C_1.

Enforcement

We propose to use two kinds of certificates for the enforcement of certification policies, namely *delegation certificates* and *property certificates*. Delegation certificates are meant for realizing the delegation. A delegation certificate is issued by a certification authority to another certification authority to allow the latter to issue certificates (delegation or property) about a property that is defined by the former. This certificate will be signed by the issuer and contain the public key of the recipient. A property certificate is a certificate that is issued by a certification authority to an agent certifying that this agent has a certain property. A property certificate is signed by the certification authority and contains the public key of the recipient.

Certification Policy - Example

Recalling our example, we consider the certification authorities, namely the WLF, FD, LTC, GOV and $REDCROSS$ that we model as instances of the concept `CA` as `CA(WLF)`, `CA(LTC)`, `CA(FD)`, `CA(GOV)`, `CA(REDCROSS)`.

WLF defines and certifies the property “appguide” to approved guides, which can be modeled with the following axioms.

```
Property(WLF.org;#appguide)
certifies(WLF,WLF.org;#appguide)
```

Further, WLF defines

$$\texttt{LTC.org;\#appguide} \sqsubseteq \texttt{WLF.org;\#appguide} \tag{4.3}$$

to specify the delegation structure which means that the guides approved by LTC are also approved guides from the point of view of WLF. By specifying such an axiom, WLF delegates the certification of the property “`WLF.org;#appguide`” to LTC.

LTC and FD also issue certificates to approved guides. Government GOV certifies the property `state.gov;#above25` to people who are above 25 years of age. Similarly, $REDCROSS$ issues certifies the property `RedCross.org;#firstaid` to persons who have experience in first aid.

$$\begin{aligned}\texttt{certifies}(\texttt{LTC}, \texttt{LTC.org;\#appguide})\\ \texttt{Property}(\texttt{FD.org;\#appguide})\\ \texttt{certifies}(\texttt{FD}, \texttt{FD.org;\#appguide})\\ \texttt{certifies}(\texttt{GOV}, \texttt{state.gov;\#above25})\\ \texttt{certifies}(\texttt{REDCROSS}, \texttt{RedCross.org;\#firstaid})\end{aligned}$$

Though on first sight the properties `WLF.org;#appguide` and `FD.org;#appguide` seem to be equivalent because of their names, they are quite different in their certification policies: The Forest Department certifies this propety only to guides who are at least 25 years old (certified by the government) and who are knowledgable in first aid (certified by Red Cross). This restriction can be expressed in the certification policy by stating the following axiom:

$$\texttt{FD.org;\#appguide} \sqsubseteq \texttt{state.gov;\#above25} \sqcap \texttt{RedCross.org;\#firstaid} \qquad (4.4)$$

Note the difference between the two Axioms (4.3) and (4.4): While in Axiom (4.3) the conclusion of the axiom lies in the namespace of specifying CA, it is the assumption of Axiom (4.4). In this sense, Axiom (4.3) has more the character of a definition (of delegation) and may substitute a delegation credential. Axiom (4.4) is more a promise or an actual certification policy which says, that the CA *FD* "promises" to check the age and the knowledge about first aid before certifying a user that he is an approved guide. In case there is no axiom in the government's namespace delegating the certification of the property *age above 25* and in the namespace of the Red Cross delegating the certification of the property knowledgeable in First Aid, a potential Web service provider now has to decide whether to trust this policy or not.

4.3.3. Users

We now turn our attention to the third logical role, namely *users*. Users are mainly interested in accessing Web services. In case of secure semantic Web services, which is our main concern in this section, any Web service discovery component must consider the user's certificates as well as the access control policies of Web services. Syntactical certificates description schemas, such as X509, KeyNote or SPKI/SDSI certificates make it difficult for a client side discovery component to perform matching based on functional as well as security aspects, because such a discovery component cannot know the meaning of the certificates that the user has and hence can not know which properties has been certified to the user.

In our setting, an end user possesses a set of certificates meta-data about each certificate. The meta-data contains information about the properties that a certificate actually certifies and hence describes the semantics of the certificate. Note, that in many cases, a certificate certifies more than one property. The end users

- have their goals in mind and want to discover and compose Web services,
- want to access Web services that offer required functionality,
- have certain properties certified to them and can use the certificates to prove their eligibility if access to a Web service is restricted.

Example

Consider an end user, who is an experienced wildlife and trekking guide and holds a certificate certifying him the property `LTC.org;#appguide`. The user wishes to register himself as an approved trekking guide in the Outdoor Shop *OutShop*.

In our example, the set of Web service descriptions that our end user obtains from an appropriate discovery component will contain the description of the Web service *OutShop*, since the matching software can infer from the certification policy of the wildlife foundation WLF that `LTC.org;#appguide` is subset of `WLF.org;#appguide` and hence a user possessing the property `LTC.org;#appguide` has access to the Web service *OutShop*.

Now, our end user wants to register as a approved trekking guide. He finds out, that this functionality is accessible only for approved guides from the forest department FD and from the Wildlife foundation WLF, that are above 25 years old and are knowledgeable in first aid. By looking at the certification policies of WLF and FD he finds out automatically, that WLF has delegated the certification of property *app_guide* to the local trekking club LTC, which means, that it will be enough to hold the certificate `LTC.org;#appguide` rather than the certificate `WLF.org;#appguide`.

The resources of an agent describe his knowledge or assets, the things that he owns and can exchange with things owned by other agents. Apart from the resources, we consider the properties of an agent in our formal model to enable access control. For example, *agent i is a university* or *agent j is above 18 years*. Now, in order to realize systems that consider the properties of agents there must be a way for an agent to prove that he possesses some property. Note, that if he cannot prove a certain property, it does not mean that he does not have that property. Consider for example, that an agent is above 18 years old but he does not have his ID to prove this. In this case, he should not be allowed to access the service, but the service should not treat him as a minor.

In distributed and open systems like the Web, there is no central certification authority, i.e. there is no one single actor that alone certifies all sorts of properties to all agents and whom all other agents can trust. Rather, any agent can act as a certification authority and certify other agents some properties. So, in distributed and open environment, the certified properties are relative to the issuer.

In capability based access control system, any agent can act as a certification authority and certify to other agents some property. Such a certification takes place by issuing certificates. We denote with $\mathcal{P}$ the set of all such properties.

Definition 9 (Certificate)**.** A certificate is a tuple (i, r, p), with $i \in \mathcal{A}$ is the issuing agent, $r \in \mathcal{A}$ the recipient and p is a property, means that the agent i certifies the agent r to possess the property $p \in \mathcal{P}$. We denote with $\mathcal{C}$, the set of all certificates.

4.4. Conclusion

In this chapter, we have presented an approach how properties and certificates can be modeled as description logics to enable reasoning, interoperability and flexibility in distributed systems that incorporate trust based access control. The basic idea was to specify the properties that are certified to agents as description logic concepts and then use concept subsumption to specify the mapping between different properties. For example, by defining the axiom $\mathsf{above21} \sqsubseteq \mathsf{adult}$, an agent acting as a certification authority, states, that anyone possessing the property $\mathsf{above21}$ also possesses the property adult. Note, that the concepts $\mathsf{above21}$ and adult may have different name spaces, i.e. they do not have be issuable by the same agent. This leads to interoperability and flexibility of access control, since an access control policy requiring an agent to prove the property adult can grant access to the agent, even if he proves the property $\mathsf{above21}$.

5. Modeling Behavior

Today there is an emerging shift in the area of Business Process Management. People are used to state-based Workflow Management Systems (WfMS). In these, a workflow consists of several activities or tasks guided by explicit control flow that defines the state the workflow is in [vdAvK02]. The workflow itself often resembles some kind of office process that is enacted in a well defined and closed environment like the department of an enterprise. Structural change occurs seldom and can be handled in a pre-defined manne [vdA01].

However, things are changing to more flexible, distributed workflows. These "new" workflows still have a state and are guided by control flow constraints. But the "new state" is made up of events instead of documents in certain places. These events are consumed and produced by activities that have no static connections but event-connectors. Events are used as preconditions to trigger activities and produced as an outcome. Some activities are still executed in closed environments, but most are distributed as services over open environments like the internet. There is no absolute control of the workflow by a single engine, or by any engine, as activities are outsourced and enforced not by technical issues but rather by legal contracts. The event-based model allows the flexible integration of activities and sub-workflows into other workflows, wherever they are distributed, and however they are executed, as long as their interaction behavior is matching.

5.1. Introduction to π-calculus

5.1.1. Relation to Other Formalisms

Sequential systems are described in the λ-calculus [Bar85] as well as Turing machines [Tur36]. Both theories have been developed in the 1930s. The science evolved to the description of parallel systems by the use of Petri nets, developed by Carl Adam Petri in the 1960s. Another thirty years later, at around 1990, a theory of mobile systems, called the π-calculus, has been developed by Robin Milner, Joachim Parrow, and David Walker. Sequential systems are based on the concept of executing one step after the other. Parallel systems as represented by Petri nets explicitly describe static structures for concurrently executed steps. Mobile systems are made up of parallel components which communicate and change their structure – thereby overcoming the limitations of static structures.

Sequential Systems

Sequential systems can be formally described by λ-calculus as well as Turing machines. The λ-calculus is a formal system designed to investigate the definition of functions, which are used for sequential computing. It brought the ideas of recursion and the precise definition of a computable function into discussion even before the first computers have been constructed. In the view of computer science, the λ-calculus can be seen as the smallest universal programming language as any computable function can be expressed and evaluated using this formalism. A different approach, computational equal to the λ-calculus, are Turing machines that form the foundation for imperative programming languages. Both, the λ-calculus as well as Turing machines, can be used to represent business processes at a very low level of abstraction, already making concepts like parallelism a complexity overwhelm.

Parallel Systems

While the λ-calculus formed the foundation for many computer science related topics like programming languages, the description of workflows required a different approach. In a typical workflow tasks are not only executed in sequential order, rather tasks are executed in parallel by different employees to speed up the processing. These different – then again sequential – processing paths have to be created and joined at some points in the business process. Even further, parallel processing tasks could depend on each other. The optimization of business processes usually adds parallelism and dependencies as this is an effective way to reduce the throughput time.

These kinds of parallel processes are hard to describe in terms of the λ-calculus. To overcome the limitations of sequential systems, an approach to represent parallel systems called Petri nets has been adapted for workflow representation. Petri nets have a simple but yet powerful mathematical foundation as well as a strong visual representation. They use the concept of an explicit state representation for parallel systems. Each Petri net is always in a precisely defined state denoted by the distribution of tokens over places contained in the net. The state of the system could then be changed by firing transitions which relocate the token distribution over the places. Petri nets have been adapted by many systems that are used in the business process management domain to describe business processes.

Beside the advantages of Petri nets for the business process management domain, that include strong visualization capabilities, mathematical foundations, as well as their main purpose, the description of parallel systems, Petri nets also have some drawbacks. The main drawbacks are the static structure of the nets (that do not support dynamic process structures) as well as the missing capabilities for advanced composition as for instance recursion. Of course, Petri have been extended with support for dynamic structure, like self modifying Petri nets, recursion, and objects. However, these enhancements also complicate the theory of the nets and thus have reached restricted usage only. A broad research on the capabilities of Petri nets regarding common patterns of behavior found in business processes showed that they fulfill basic tasks like splitting and merging process paths easily, while they fail at advanced patterns like multiple instances of a task with dynamic boundaries. Whereas there exist approaches to overcome some or all of the limitations regarding the behavior, the static structure and limited composition of Petri nets remains [vdAvK02].

Mobile Systems

To overcome the limitations of Petri nets, theories of mobile systems have been developed. Thereby a mobile system is made up of entities that move in a certain space. The space consists of processes, and the entities that move are either links between the processes (link passing mobility) or the processes themselves (process passing mobility). A theory for mobile systems, the π-calculus, overcomes the limitations of Petri nets regarding the static structure and limited composition capabilities at the cost of a more complex representation. The π-calculus represents mobility by directly expressing movements of links in an abstract space of linked processes (i.e. link passing mobility). Practical examples are hypertext links that can be created, passed around, and disappear. The π-calculus does not, however, support another kind of mobility that represents the movement of processes. An example is code that is sent across a network and executed at its destination. The π-calculus uses the concept of names with a certain scope for interaction between different parallel processes. Names are a collective term for concepts like channels, links, pointers, and so on. As the mobile system evolves, names are communicated between processes and extrude or intrude their scope regarding to certain processes. As the synchronization between processes is based on interaction and received names are also used as communication channels, the link structure is changed dynamically all the time the mobile system evolves.

In this section, we give a short introduction to π-calculus, a calculus of communicating systems in which one can naturally express processes which have changing nature. The π-calculus is a process algebra whose processes interact by sending communication links to each other. π-calculus was first introduced in [MPW92]. Details about various extensions of π-calculus can be found in [SW01].

The π-calculus is a mathematical model of processes whose interconnections change as they interact. The basic computational step is the transfer of a communication link between two processes; the recipient can then use the link for further interaction with other parties. This makes the calculus suitable for modelling systems where the accessible resources vary over times. It also provides a significant expressive power since the notions of access and resouce underlie much of the theory of concurrent computation, in the same way as the more abstract and mathematically tractable concept of a function underlies functional computation.

Let us consider an example. Suppose a server controls access to a printer and a client wishes to access it. In the original state only the server itself has access to the printer, represented by a communication link a in Figure 5.1. After an interaction with the client along some other link b this access to the printer has been transferred.

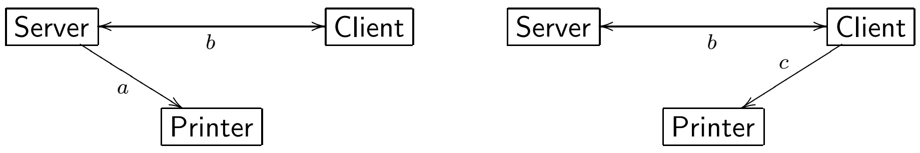


Figure 5.1.: Link passing in π-calculus: before and after interaction

In the π-calculus this is expressed as follows: the server that sends a along b is $\overline{b}a.S$; the client that receives some link along b and then uses it to send data along it is $b(c).\overline{c}d.P$. In the process expression describing the client, the name c denotes the variable that is used to remember the link received along b ($b(c)$) such that it can be used later ($\overline{c}d.P$). The interaction depicted in Figure 5.1 is formulated

$$\overline{b}a.S \mid b(c).\overline{c}d.P \xrightarrow{\tau} S \mid \overline{a}d.P$$

We see that a plays two different roles. In the interaction between the server and the client it is an object transferred from one to the other. In a further interaction between the client and the printer it is the name of the communication link. The idea that the names of the links belong to the same category as the transferred objects is one the cornerstones of the calculus, and is one way in which it is different from other process algebras.

5.1.2. Syntax

Assume an infinite set $\mathcal{N}$ of names and x, y, z, w, v as names. The syntax of an agent can be summarized as follows:

$$
\begin{array}{rcl}
P & ::= & \mathbf{0} \\
 & | & \tau.P \\
 & | & y(x).P \\
 & | & \overline{y}x.P \\
 & | & P_1 \parallel P_2 \\
 & | & P_1 + P_2 \\
 & | & \omega?Q{:}R \\
 & | & (\mathtt{new}\ x)P \\
 & | & A(y_1, \ldots, y_n)
\end{array}
$$

- *Null Process* **0** is a process that does nothing.
- A *prefix form* $\tau.P$, $y(x).P$ or $\overline{y}x.P$.

  τ is called a *silent prefix.* $\tau.P$ performs the silent action τ and then behaves like P.

  $y(x)$ is called a *positive prefix.* A name y may be thought of as an input port of an agent; $y(x).P$ inputs an arbitrary name z at port y and then behaves like $P\{z/x\}$ (refer to the definition of substitution below). The name x is bound by the positive prefix '$y(x)$'.

  $\overline{y}x$ is called a *negative prefix.* $\overline{y}$ may be thought of as an output port of an agent which contains it; $\overline{y}x.P$ outputs the name x at port y and then behaves like P. Note, that a negative prefix does not bind a name.
- A *Composition* $P_1 \parallel P_2$.

  This agent consists of P_1 and P_2 acting in parallel. The components may act independently; also, an output action of P_1 (resp. P_2) at any output port x may synchronize with an input action of P_2 (resp. P_1) at x, to create a silent (τ) action of the composite agent $P_1 \parallel P_2$.
- A *Summation* $P_1 + P_2$

  This agent behaves like either P_1 or P_2.
- A *Match* $\omega?Q{:}R$ behaves like Q if the condition ω is true, and otherwise like R.
- *Restriction* $(\mathtt{new}\ x)P$ This agent behaves as P but the name x is local, meaning it cannot immediately be used as a port for communication between P and its environment. However, it can be used for communication between components of P.

- A *defined agent* $A(y_1, \ldots, y_n)$.

  For any agent identifier A (with arity n) used thus, there must be a unique defining equation $A(x_1, \ldots, x_n) \overset{\text{def}}{=} P$, where the names $x_1, \ldots, x_n$ are distinct and are the only names which may occur free in P. Then $A(y_1, \ldots, y_n)$ behaves like $P\{y_1/x_1, \ldots, y_n/x_n\}$ (see below for the definition of substitution). Defining equations provide recursion, since P may contain any agent identifier, even A itself.

Definition 10. In each agent of one of the forms $x(y).P$ and $(\texttt{new}\ y)P$ the occurence of y within parentheses is a *binding* occurence, and in each case the *scope* of occurence is P. An occurence of y in an agent is said to be *free* if it does not lie within the scope of a binding occurrence of y. The set of names occurring free in P is denoted by $fn(P)$. $fn(P, Q, \ldots, x, y, \ldots)$ is used as an abbreviation for $fn(P) \cup fn(Q) \cup \ldots \cup \{x, y, \ldots\}$.

Definition 11. A defining equation of an agent identifier A of arity n is of the form

$$A(x_1, \ldots, x_n) \overset{\text{def}}{=} P$$

where the x_i are pairwise distinct and $fn(P) \subseteq \{x_1, \ldots, x_n\}$.

Definition 12. An occurrence of a name in an agent is said to be *bound* if it is not free. We assume that the set of bound names of P, $bn(P)$, is defined in such a way that it contains all names which occur bound in P and that if $A(\tilde{x}) \overset{\text{def}}{=} Q$ then $bn(A(\tilde{x})) = bn(Q)$, where $\tilde{x} = x_1, \ldots, x_n$. We write $n(P)$ for the set $fn(P) \cup bn(P)$ of *names* of P.

Definition 13. A substitution is a function σ from $\mathcal{N}$ to $\mathcal{N}$ which is almost everywhere identity. If $x_i\sigma = y_i$ for all i with $1 \leq i \leq n$ (and $x\sigma = x$ for all other names x).

Definition 14. Processes P and Q are α-convertible, denoted as $P \equiv Q$, if Q can be obtained from P by a finite number of changes of bound names. α-convertibility can be seen as syntactic identity between processes.

Let $P\sigma$ denote the process obtained by simultaneously substituting $z\sigma$ for each free occurrence of z in P for each z, with change of bound names to avoid captures. In particular the following hold, when the bound name y is replaced by the name y'.

$$\begin{aligned} (x(y).P)\sigma &\equiv x\sigma(y').P\{y'/y\}\sigma \\ ((\texttt{new}\ y)P)\sigma &\equiv (\texttt{new}\ y')P\{y'/y\}\sigma \end{aligned}$$

We shall write $P\{y_1/x_1, \ldots, y_n/x_n\}$ or $P\{y_i/x_i\}_{1 \leq i \leq n}$ or $P\{\tilde{y}/\tilde{x}\}$ for the simulatenous substitution of y_i for all free occurrences of x_i (for $1 \leq i \leq n$) in P, with change of bound names if necessary to prevent any of the new names y_i from becoming bound in P.

α	Kind	Free/Bound	Polarity	$fn(\alpha)$	$bn(\alpha)$
τ	Silent	f	0	$\emptyset$	$\emptyset$
$\overline{y}x$	Free Output	f	$-$	$\{x, y\}$	$\emptyset$
$y(x)$	Input	b	$+$	$\{y\}$	$\{x\}$
$\overline{y}(x)$	Bound Output	b	$-$	$\{y\}$	$\{x\}$

Table 5.1.: The actions

5.1.3. Semantics

The standard way to give an operational semantics to a process algebra is through a labelled transition system, where transitions are of kind $P \xrightarrow{\alpha} Q$ for some set of actions ranged over by α. The π-calculus follows this norm and most of the rules of transitions are similar to other algebras. For example, for an agent $\alpha.P$ there will be a transition labelled α leading to P and the restriction operator will not permit an action with the restricted name as subject, so $(\texttt{new}\ a)\overline{a}u.P$ has no transitions and is therefore semantically equivalent to $\mathbf{0}$.

However the process $(\texttt{new}\ u)\overline{a}u.P$ must have some action. Inserted in a context $a(x).Q \parallel (\texttt{new}\ u)\overline{a}u.P$ it will enable an interaction since, assuming $u \notin fn(Q)$, this term is syntactically congruent to $(\texttt{new}\ u)a(x).Q \parallel \overline{a}u.P$ and there is an interaction between the components. So $(\texttt{new}\ u)\overline{a}u.P$ is not, intuitively, something that behaves as $\mathbf{0}$. On the other hand it is clearly distinct from $\overline{a}u$. For example,

$$a(x).(x = u)?Q{:}\mathbf{0} \parallel \overline{a}u \xrightarrow{\tau} (u = u)?Q{:}\mathbf{0}$$

which can continue as Q, while

$$(a(x).(x = u)?Q{:}\mathbf{0}) \parallel (\texttt{new}\ u)\overline{a}u \equiv (\texttt{new}\ v)(a(x).(x = u)?Q{:}\mathbf{0} \parallel \overline{a}v)$$

and

$$(\texttt{new}\ v)(a(x).(x = u)?Q{:}\mathbf{0} \parallel \overline{a}v) \xrightarrow{\tau} (\texttt{new}\ v)(v = u)?Q{:}\mathbf{0}$$

and there are no further actions.

The solution is to give $(\texttt{new}\ u)\overline{a}u$ a new kind of action called *bound output* written $\overline{a}(u)$. The intuition is that a local name represented by u is transmitted along a, extending the scope of u to the recipient. In summaray, the actions ranged over by α consists of the following four classes as shown in Table 5.1

1. The *silent* action τ. $P \xrightarrow{\tau} Q$ means that P can evolve into Q, and in doing so requires no interaction with the environment. Silent actions can naturally arise from agents of form $\tau.P$, but also from communications within an agent.

2. A *free* output action $\overline{x}y$. The transition $P \xrightarrow{\overline{x}y} Q$ implies that P can emit the free

name y on the port $\overline{x}$. Free output arise from the output prefix form $\overline{x}y.P$.

3. An *input* action $x(y)$. Intuitively, $P \xrightarrow{x(y)} Q$ means that P can receive any name w on the port x, and then evolve into $Q\{w/y\}$. Here, (y) represents a reference to the place where the received value will go; y is enclosed in brackets to emphasize this fact. Input actions arise from the input prefix for $x(y).P$.

4. A *bound output* action $\overline{x}(y)$. Intuitively, $P \xrightarrow{\overline{x}(y)} Q$ means that P emits a private name (i.e. a name bound in P) on port $\overline{x}$, and (y) is a reference to where this private name occurs. As in the input action above, y is enclosed in brackets to emphasize that it is a reference and does not represent a free name. Bound output actions arise from free output actions which carry name out of their scope, as e.g. in the agent $(\texttt{new}\ y)\overline{x}y.P$.

The silent action and free output actions are collectively called *free* actions, while input actions and bound output actions are called *bound* actions. Thus, the bound actions carry "references" rather than values; these references are in the form of names within brackets.

The free output and bound output actions are collectively called *output* actions, or *negative* actions (actions of negative polarity). Similarly, the input actions are called *positive* actions (actions of positive polarity). Two actions must be of opposite polarity in order to combine into an internal communication.

In the output and input actions mentioned above, x ist the *subject* and y is the *object* or *parameter*. The object is said to be *bound* in the bound actions and *free* in free actions. The set of *bound* name $bn(\alpha)$ of an action α is the empty set if α is a free action; otherwise it contains just the bound object of α. The set of *free* names $fn(\alpha)$ of α contains the subject and free object (if any) of α, and the names $n(\alpha)$ of α is the union $bn(\alpha)$ and $fn(\alpha)$. Note, that $n(\tau) = \emptyset$.

The operational semantics of π-calculus is given in Figure 5.2. The operational semantics of π-calculus maps a π-calculus process expression to a labeled transition system by viewing operations (communication operations and silent operation) as transitions and process expressions as states [MPW92].

5.1.4. The Polyadic π-calculus

A straightforward extension of the π-calculus is to allow multiple objects in communications: outputs of type $\overline{a}\langle y_1, \ldots, y_n\rangle.P$ and inputs of type $a(x_1, \ldots, x_n).Q$ where x_i are pairwise distinct. In polyadic calculus, the case $n = 0$ is also admitted. Though allowing multiple name does not increase the expressiveness of the calculus since polyadic interactions can be emulated by sequences of monadic interactions over a private link, the question arises how to treat agents such $a(xy).P \parallel \overline{a}\langle u\rangle.Q$ where the arity of the output

$$\text{TAU-ACT:} \quad \frac{-}{\tau.P \xrightarrow{\tau} P}$$

$$\text{OUTPUT-ACT:} \quad \frac{-}{\overline{x}y.P \xrightarrow{\overline{x}y} P}$$

$$\text{INPUT-ACT:} \quad \frac{-}{x(z).P \xrightarrow{x(w)} P\{w/z\}} \quad w \in fn((z)P)$$

$$\text{SUM:} \quad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

$$\text{MATCH:} \quad \frac{P \xrightarrow{\alpha} P'}{[x = x]P \xrightarrow{\alpha} P'}$$

$$\text{IDE:} \quad \frac{P\{\tilde{y}/\tilde{x}\} \xrightarrow{\alpha} P'}{A(\tilde{y}) \xrightarrow{\alpha} P'} \quad A(\tilde{x}) \overset{\text{def}}{=} P$$

$$\text{PAR:} \quad \frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q} \quad bn(\alpha) \cap fn(Q) = \emptyset$$

$$\text{PAR:} \quad \frac{Q \xrightarrow{\alpha} Q'}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'} \quad bn(\alpha) \cap fn(P) = \emptyset$$

$$\text{COM:} \quad \frac{P \xrightarrow{\overline{x}y} P' \quad Q \xrightarrow{x(z)} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'\{y/z\}}$$

$$\text{CLOSE:} \quad \frac{P \xrightarrow{\overline{x}(w)} P' \quad Q \xrightarrow{x(w)} Q'}{P \parallel Q \xrightarrow{\tau} (w)(P' \parallel Q')}$$

$$\text{RES:} \quad \frac{P \xrightarrow{\alpha} P'}{(y)P \xrightarrow{\alpha} (y)P'} \quad y \in n(\alpha)$$

$$\text{OPEN:} \quad \frac{P \xrightarrow{\overline{x}y} P'}{(y)P \xrightarrow{\overline{x}(w)} P'\{w/y\}} \quad y \neq x, w \notin fn((y)P')$$

Figure 5.2.: Operational Semantics

is not the same as the arity of the input. Such an incompatibility should be caught by a type system. The idea is that each name is assigned a *sort*, containing information about the objects that can be passed along the name. If $\mathcal{S}$ is a set of sorts, a *sort context* Δ is a function from $\mathcal{N}$ to $\mathcal{S}$, in other words $\Delta(a)$ is the sort of a.

In the simplest system a sort would just be a natural number, $\mathcal{S} = \mathbb{N}$, such that $\Delta(a)$ denotes the arity of a, i.e. the number of objects in any prefix where a is the subject. Formally we write $\Delta \vdash P$ to mean that P conforms to Δ, and rules for inferring $\Delta \vdash P$ can easily be given by induction over the structure of P. For example,

$$\frac{\Delta \vdash P, \Delta(a) = n}{\Delta \vdash \overline{a}\langle x_1 \dots x_n\rangle.P} \qquad \frac{\Delta \vdash P, \Delta \vdash Q}{\Delta \vdash P \parallel Q}$$

With this idea the agent $a(xy).P \parallel \overline{a}\langle u\rangle.Q$ is ill-formed in the sense that it cannot conform to any sort context: $\Delta(a)$ is required to be 2 in one component and 1 in the other component. However, although this simple scheme works for this particular example, it is not in general sufficient to catch all incompatible arities that may arise when an agent executes. Consider:

$$a(u).u(z).\mathbf{0} \parallel \overline{a}\langle x\rangle.\overline{x}\langle vv\rangle.\mathbf{0}$$

A sorting assining 2 to x and 1 to all other names works fine here. Yet the agent can evolve through an interaction along a to

$$x(z).\mathbf{0} \parallel \overline{x}\langle vv\rangle.\mathbf{0}$$

which is ill-formed. To be able to catch not only the immediately obvious arity conflicts but also any such conflicts that can aries during execution more information must be added to the sorts. For each name, the number of objects passed along that name is not enough, also the sort each such object must be included. In the example above, the left component requires the sort of a to be one object (corresponding to u) which has sort 1 because of the subterm $u(z)$. The right component requires the sort of a to be one object (corresponding to x) of sort 2 because of the subterm $\overline{x}\langle vv\rangle$. With this refined notion of sort an agent such as $a(u).u(z).\mathbf{0} \parallel \overline{a}\langle x\rangle.\overline{x}\langle vv\rangle.\mathbf{0}$ is ill-formed.

Of course arity conflicts can lie arbitrarily deep, meaning that the sort of a must contain information about the objects which are passed along the objects which are passed along ... which are passed along a to arbitrary depth. One way to set this up is the following. To each sort $S \in \mathcal{S}$ associate a fixed *object sort* $ob(S)$ in $\mathcal{S}*$ i.e., the object sort a (possibly empty) sequence of sorts. The intention is that if a has sort S where $ob(S) = \langle S_1, \dots, S_n\rangle$, and $a(x_1 \dots x_n)$ is a prefix, then each x_i has sort S_i. The sorting rule for input then becomes:

$$\frac{\Delta \cup \{\tilde{x} \mapsto ob(\Delta(a))\} \vdash P}{\Delta \vdash a(\tilde{x}).P}$$

It should be read as follows: In order to establish that $a(x_1 \dots x_n).P$ conforms to Δ, find the object sorts $S_1 \dots S_n$ of a according to Δ, and verify that P conforms to Δ where also each x_i is assigned sort S_i. The rule for output is:

$$\frac{ob(\Delta(a)) = \Delta(y_1)\dots\Delta(y_n), \quad \Delta \vdash P}{\Delta \vdash \overline{a}\tilde{y}.P}$$

In order to establish that $\overline{a}\langle y_1 \dots y_n\rangle$ conforms to Δ it is enough to show that it assigns $y_1 \dots y_n$ the object sorts of a, and that P conforms to Δ. In this way agents such $\overline{a}a.P$, where a name is sent along itself, can also be given a sorting: if S is the sort of a then $ob(S) = \langle S\rangle$.

In conclusion, although polyadic interactions do not really increase the expressiveness it adds convenience and clarity when using the calculus, and efficiently implementable sort systems can ascertain that no mismatching arities will ever occur when agents evolve.

Now, we turn our attention to how the sets $\mathcal{R}_A$, $\mathcal{C}_A$ and the behavior $\mathcal{B}_A$ of an agent $A \in \mathcal{A}$ can be modeled formally.

5.2. Modeling Behavior

Now, we turn our attention to modeling the third aspect of an agent, namely, his dynamic behavior. We use polyadic π-calculus for modeling dynamic behavior of an agent.

Polyadic π-calculus is a powerful tool for describing the dynamics of communicating mobile processes. However, π-calculus names, i.e. the objects that are communicated among agent, do not have any structure and any semantics. This is because π-calculus is a pure process algebra and reasoning about the meaning of the involved static objects in a process is not the focus in the process algebra community. As a result static objects are just considered as strings.

Another problem with π-calculus is that it suggests to model even the simplest tasks like adding two number as processes[Mil80, SW01]. Thus, modeling processes of practical interest with pure π-calculus syntax is tedious and one obtains unnecessary long process expressions. This also makes it impossible to model certain tasks as black boxes. Note, that the pure π-calculus has only the silent action τ as local operation which does not have any effects.

Finally, although, π-calculus offers the notion of guarded processes (processes that are executed only if certain condition is fulfilled), the only condition allowed is the equality check on objects. We extend the notion of guarded processes by using boolean predicate symbols as condition expressions, for example to embed access checks.

In this section, we will mainly address these three issues and show how π-calculus can be combined with description logics and how access control policies can be seen as conditions.

An agent performs operations alone or jointly with other agents. The joint operations are communication operations consisting of message exchange. Local operations of an agents are not observable for other agents, whereas the communication operations are observable for the agents that participate in the communication.

Definition 15 (Event, Labelling Function, Agent Function)**.** An event e is an occurrence of an operation. With $\lambda(e)$, we denote the operation occurred at event e. With $\alpha(o)$, we denote the set of agents that can perform the operation o. That is, $\alpha(o) = \{A \in \mathcal{A} : o \in \mathcal{O}_A\}$.

5.2.1. Message Types for Communication Operations

We view π-calculus names as entities, the structure of which is described with TBox and ABox axioms (cf. Figures 3.2 and 3.3).

In π-calculus agents communicate via exchanging messages, which is one more reason for the suitability of π-calculus for describing web processes. In general, more than one resource can be transported or communicated from one agent to another in a communication action. A resource can be transported from one agent to another only if it is serializable to an appropriate type.

As we have seen the polyadic π-calculus supports sorts to ensure that the communication only takes place if the arity of the incoming message matches with the expected arity. However, it does not ensure that the communication only takes place if the resources sent by one party are of the type that the receiving party expects. Furthermore, π-calculus abstracts from protocol types. Protocol types are important in practice, since many business processes run in a mixed environment.

Definition 16 (Message Part)**.** Polyadic π-calculus allows to exchange more than one resource within one communication activity by allowing extended syntax elements $\overline{a}\langle x_1, \ldots, x_n\rangle.P$ for outputs and $a(x_1, \ldots, x_n).Q$ for inputs activities respectively (refer to Section 5.1.4). We call one such x_i a message part.

Definition 17 (Message Type)**.** We denote with $P_1{:}C_1, \ldots, P_n{:}C_n$ a message type, with each C_i belonging to a T-Box $T \in \mathcal{T}$ denoting the type of the message part P_i. Furthermore, there must be at least one serialization type common for every C_i.

Definition 18 (Communication Channel)**.** A communication channel is a tuple (p, a, t), where p is the protocol, a the address and t the type of message that can be transmitted via the channel. Examples of common communication protocols are “HTTP”, “Phone”, “Fax”, “Email (SMTP/POP)”, “Surface Mail”. Each protocol supports a set of MIME types that it can transport. For example, “HTTP” supports “XML” and “HTML” that can not be sent by “Surface Mail”.

Recall, that an input process $c(v_1, \ldots, v_k).Q$ means that the process first receives a message with resources $r_1, \ldots, r_k$ along the channel c, binds $(r_1, \ldots, r_k)$ to variables v_1 to v_k and then behaves like the process Q. An output operation $\overline{c}u_1, \ldots, u_k$ sends a message consisting of k parts $u_1, \ldots, u_k$ along the channel c.

In both case, we give c the structure C_i, where C is a communication channel and i is the unique identifier of the instantiation of C. The semantics of $C_i(v_1, \ldots, v_n)$, with $C = (p, a, t)$, is an input event e that expects a message of type t at address a and with the protocol p. Similarly, the semantics of $\overline{C_i}(v_1, \ldots, v_n)$, with $C = (p, a, t)$, is an output event e that sends a message of type t at address a and with protocol p.

Finally, by modeling communication channels as description logic individuals we make sure that channel descriptions can be sent and received just like any other resources and thus the mobility is kept. While doing so, we model a set of resources $R_i \subseteq \mathcal{R}$ as a description logic concept $\mathsf{R_i}$. For each protocol p and for each serialization type, we model description logic individuals p and a respectively. A communication channel $C = (p, a, t)$ with $t = R_1, \ldots, R_n$ can now be easily modeled by defining corresponding relationships of the individual C with individuals p and a and with the concepts $\mathsf{R_1}, \ldots, \mathsf{R_n}$.

5.2.2. Introducing Local Operations to Model Updates

As we have already mentioned, we model the knowledge bases of the agents with description logics A-Boxes. A DL A-Box consists of the following types of axioms:

- *ClassMember*(x, y): means that the individual x is member or instance of the concept y.
- *DataPropertyMember*(p, x, y): means that the individual x has y as value of the data property p.
- *ObjectPropertyMember*(p, x, y): means that the individual x has individual y as value of the object property p
- *SameIndividual*(x, y): means that the individuals x are y are same.
- *DifferentIndividual*(x, y): means that the individuals x and y are different.

The above types of A-Box axioms represent the only types of changes that can be performed in an A-Box. When an agent updates his local knowledge base, he adds new A-Box axioms to his knowledge base or removes some of the existing axioms from his knowledge base. Obviously, such changes or updates happen locally. That is, except the agent, whom the knowledge base belongs to, there is no other agent involved in an update activity. As already mentioned aboove, we call such operations local operations.

A local operation is a decidable procedure that can add new DL axioms in the knowledge base or remove existing DL axioms from the knowledge base of the agent that executes the local operation. So, we model a local operation as $l(x_1, \ldots, x_n)$ and its effects as a list of changes Δ , where each element $\delta \in \Delta$ is a parameterized DL A-Box axiom. Furthermore, a change $\delta \in \Delta$ is adorned with a sign (+ or -) that indicates whether the axiom corresponing to c is added to or removed from the knowledge base. Every parameter of an axiom corresponding to a change belong to the set $\{x_1, \ldots, x_n\}$. For example, if $A = \{classMember(x_1, x_2)\}$ belongs to the effects of an operation $l(x_1, x_2)$, executing l with arguments $Peter$ and $Person$ will add the axiom $classMember(Peter, Person)$ to the knowledge base.

A local operation may also perform some calculation and connect the result via some property to an individual. For example, $add(x, y, z)$ adds x and y and assigns the result $(x + y)$ to z. The result z can then be treated as an individual and connected via a property to another individual. This can be done by adding a DL-safe rule in the ontology[1]. For example, a DL-safe rule $income(Peter, z) \leftarrow income(Pedro, x), add(x, 100, z)$ sets Peter's income to 100 more than that of Pedro.

5.2.3. Integrating Access Control Policies as Conditions

In [AS05], we have shown that modeling a property as a description logic concept allows a straightforward mapping of the set theoretic operations to description logic constructors (cf. Figure 3.1).

We model access control policies as pre conditions and foresee an input parameter for the set of certificates a user has to show in order to prove his eligibility to access a Web service. We use a built-in predicate $CCD(C, P)$, that is true iff the set C of certificates fulfills the access control policy P according to the certificate chain discovery algorithm [AS05, CEE+01].

Since we specify the properties that are certified to agents as description logic concepts, we can use concept subsumption to specify the mapping between different properties. For example, by defining the axiom $\mathsf{P}_1 \sqsubseteq \mathsf{P}_2$, an agent acting as a certification authority, states, that anyone possessing the property p_1 also possesses the property p_2.

We now turn our attention to Web service providers and consider the problem how they can restrict access to their services. In a capability based setting, access is granted or denied on the basis of certified properties. For a correct specification of the access control policy, a Web service provider faces the following two problems: (1) he must understand the meaning of credentials and certified properties and (2) he must trust the issuers of credentials. A Web service provider can understand the meaning of the credentials issued by a certification authority from the description of the properties that the certification authority certifies. On the basis of the description of the properties

[1]Note, that in order to support this, rule extensions to description logics must be used.

that a certification authority possesses, a Web service provider can build his trust in the certification authority.

We now show how a Web service provider can make use of certification policies as introduced in Section 4.3.2 while (1) *specifying* the access control policy for his service and (2) *verifying* the eligibility for access of a particular user.

Specifying Access Control Policies

An *access control policy* for a Web service w is a set of *authorization terms* (p, w, f). Each authorization term has the intuitive meaning that a user being able to prove property p is granted access to functionality f of Web service w. The tuple $\langle w, f\rangle$ is called an *interface*, the set of all interfaces is denoted by I. We define the *expansion* $exp(p, w, f)$ of an authorization term to be the set $\{(s, w, f)|$ subject s can prove to have property $p\}$ and the expansion $exp(\Pi)$ of a policy Π to be the union of the expansions of elements of Π. An authorization term can be defined with DL axioms as follows:

$$\begin{aligned}\texttt{AuthorizationTerm} \sqsubseteq\ & \top \sqcap \exists \texttt{subject.ca-Property} \sqcap \\ & \exists \texttt{object.WebService} \sqcap \\ & \exists \texttt{authorization.WebServiceFunctionality}\end{aligned}$$

Verifying Eligibility of a User

When a user requests access by showing his set of certificates the Web service provider must be able to verify the user's eligibility in order to decide whether to grant or to deny access to the user. To verify the eligibility of a user, he checks for each access requirement, whether the shown set of credentials (plus possibly published certification policies and delegation credentials published by relevant CAs) contain a valid certificate chain from some trusted CA to the required property. However, in this process, the service provider also needs to consider the possible revocation of one or more certificates shown by the user. This could either be a certificate directly issued to the user or a certificate in the certificate chain issued to one of the involved CAs. If the CAs publish their certification policies including the delegation credentials issued to them in machine readable form on the Web, it might still be possible to automatically find a valid chain that proves the users' eligibility, e.g. via newly stated relationships or other published delegation certificates.

Definition 19 (Access Control Policy)**.** An access control policy is an expression E defined recursively over the properties from the set of properties $\mathcal{P}$ as follows:

$$E ::= p \in \mathcal{P} \mid E + E \mid E \,\&\, E \mid E - E$$

The semantics of such policy algebra expression is a function that maps each policy expression to an expanded policy, inductively extending an environment by using the pertinent interpretation of the operators over policies. The operators addition (+), the conjunction (&) and subtraction (−) are interpreted as set-theoretic union, intersection and difference on the expanded policies, respectively. The scoping restriction "C" restricts a policy to the interfaces $\langle w, f\rangle \in C$.

Access Control Policy - Example

Let us now consider our running example again to illustrate how a Web service provider can specify an access control policy using the knowledge he gains from the certification policies of the certification authorities.

The outdoor shop *OutShop* offers a registration service for approved trekking guides. For the registration they have the following access condition: Each trekking guide must be approved by either the Forest department *FD* or by the wildlife foundation *WLF*, must at least 25 years old and must be knowledgeable in first aid. This leads to the following access control policy:

$$ACP(P) := \{(WLF, \texttt{OS.edu}, \textit{trecking guide}), (FD, \texttt{OS.edu}, \ \textit{trecking guide})\}$$

where WLF and FD are defined as follows:

$$\begin{aligned} WLF &\equiv \texttt{WLF.org;\#appguide} \sqcap \texttt{state.gov;\#above25} \sqcap \texttt{RedCross.org;\#firstaid} \\ FD &\equiv \texttt{FD.org;\#appguide} \sqcap \texttt{state.gov;\#above25} \sqcap \texttt{RedCross.org;\#firstaid} \end{aligned}$$

From the certification policy, the Web service provider can infer that whoever has the property `FD.org;#appguide` also has the properties `state.gov;#above25` and `RedCross.org;#first`
At the time of specification of the policy, it allows him to relax the policy and thus reduce the number of certificates, a potential user has to present:

$$ACP(P) := \{(WLF, \texttt{OS.edu}, \textit{trecking guide}), (FD, \texttt{OS.edu}, \ \textit{trecking guide})\}$$

with FD being defined as

$$FD \equiv \texttt{FD.org;\#appguide}.$$

At the time of verification it allows the Web service provider to verify, that a user, who has presented "only" a valid chain for `FD.org;#appguide` is still eligible for the registration.

Now, we consider how checking the eligibility of a user according to an access control policy can be integrated in the behavior of a Web service. Recall from Section 5.1.2 the process expressions of type IfThenElse. A process expression $\omega?P{:}Q$ checks the condition ω and if it holds, the process behaves like the process P otherwise like the process Q. We model checking of access eligibility as a condition by using a predicate symbol $CCD(C, P, a)$, that is true iff the set of credentials C proves the eligibility of the agent a according to the access control policy P. The eligiblity is checked by the certificate chain discovery algorithm [CEE+01].

Having seen, how resources, credentials and behavior of an invocable agent can be specified semantically, we just view a Web service as a special type of agent that is invocable via a standard Web protocol. That is, a Web service is an agent $c(\ldots).P$ with channel $c = (p, a, t)$ such that $p \in \{HTTP, SMTP, \ldots\}$.

5.3. Related Work

5.3.1. OWL-S

OWL-S formerly known as DAML-S [SPAS03] is perhaps the first initiative to address the need of describing Web services semantically. OWL-S is an OWL ontology for

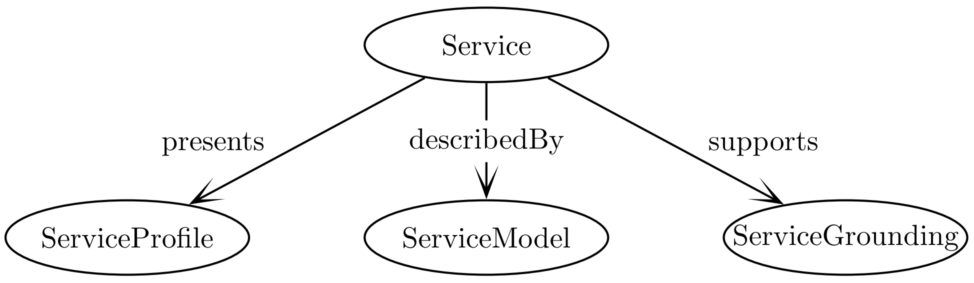


Figure 5.3.: OWL-S Top Level Ontology

describing Web services. Figure 5.3 shows the OWL-S top level or upper level ontology. The structure of the ontology of services is motivated by the need to provide three essential types of knowledge about a service (Figure 5.3), each characterized by the question it answers:

- What does the service require of the user(s), or other agents, and provide for them? Thus, the class Service presents a ServiceProfile.
- How does it work? Thus, the class Service is describedBy a ServiceModel.
- How is it used? Thus, the class Service supports a ServiceGrounding.

The class Service provides an organizational point of reference for declaring Web services; one instance of Service will exist for each distinct published service. The properties presents, describedBy, and supports are properties of Service. The classes ServiceProfile, ServiceModel, and ServiceGrounding are the respective ranges of those properties. Each instance of Service will present a descendant class of ServiceProfile, be describedBy a descendant class of ServiceModel, and support a descendant class of ServiceGrounding. The details of profiles, models, and groundings may vary widely from one type of service to another–that is, from one descendant class of Service to another.

Service Profile

The service profile tells "what the service does"; that is, it gives the types of information needed by a service-seeking agent (or matchmaking agent acting on behalf of a service-seeking agent) to determine whether the service meets its needs. The Service Profile does not mandate any representation of services; rather, using the OWL subclassing it is possible to create specialized representations of services that can be used as service profiles. OWL-S provides one possible representation through the class Profile. An OWL-S Profile describes a service as a function of three basic types of information: what organization provides the service, what function the service computes, and a set of features that specify characteristics of the service.

The provider information consists of contact information that refers to the entity that provides the service. For instance, contact information may refer to the maintenance operator that is responsible for running the service, or to a customer representative that may provide additional information about the service.

The functional description of the service is expressed in terms of the transformation produced by the service. Specifically, it specifices the inputs required by the service and the outputs generated; furthermore, since a service may require external conditions to be satisfied, and it has the effect of changing such conditions, the profile describes the preconditions required by the service and the expected effects that result from the execution of the service. For example, a selling service may require as a precondition a valid credit card and as input the credit card number and expiration date. As output it generates a receipt, and as effect the card is charged.

Finally, the profile allows the description of a host of properties that are used to describe features of the service. The first type of information specifies the category of a given service, for example, the category of the service within the UNSPSC classification system. The second type of information is quality rating of the service: some services may be very good, reliable, and quick to respond; others may be unreliable, sluggish, or even malevolent. Before using a service, a requester may want to check what kind of service it is dealing with; therefore, a service may want to publish its rating within a specified rating system, to showcase the quality of service it provides. It is up to the service requester to use this information, to verify that it is indeed correct, and to decide what to do with it. The last type of information is an unbounded list of service parameters that can contain any type of information. The OWL-S Profile provides a mechanism for representing such parameters; which might include parameters that provide an estimate of the max response time or to the geographic availability of a service.

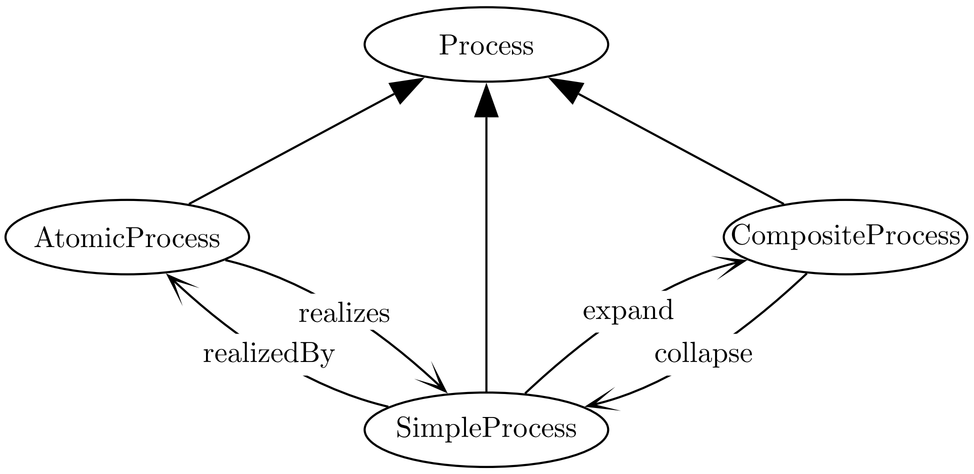


Figure 5.4.: OWL-S Top Level Process Ontology

Service Model

The service model tells “how the service works”; that is, it describes what happens when the service is carried out. For nontrivial services (those composed of several steps over time), this description may be used by a service-seeking agent in at least four different ways: (1) to perform a more in-depth analysis of whether the service meets its needs; (2) to compose service descriptions from multiple services to perform a specific task; (3) during the course of the service enactment, to coordinate the activities of the different participants; and (4) to monitor the execution of the service.

As shown in Figure 5.4, the process model identifies three types of processes: atomic, simple, and composite. Each of these is described below.

The atomic processes are directly invocable (by passing them the appropriate messages). Atomic processes have no subprocesses, and execute in a single step, from the perspective of the service requester. That is, they take an input message, execute, and then return their output message – and the service requester has no visibility into the service’s execution. For each atomic process, there must be provided a grounding that enables a service requester to construct these messages.

Simple processes are not invocable and are not associated with a grounding, but, like atomic processes, they are conceived of as having single-step executions. Simple processes are used as elements of abstraction; a simple process may be used either to provide a view of (a specialized way of using) some atomic process, or a simplified representation of some composite process (for purposes of planning and reasoning). In the former case, the simple process is realizedBy the atomic process; in the latter case, the simple process expandsTo the composite process.

Composite processes are decomposable into other (non-composite or composite) processes; their decomposition can be specified by using control constructs such as Sequence and If-Then-Else. Such a decomposition normally shows, among other things, how the various inputs of the process are accepted by particular subprocesses, and how its various outputs are returned by particular subprocesses.

Service Grounding

A service grounding specifies the details of how an agent can access a service. Typically a grounding will specify a communication protocol, message formats, and other service-specific details such as port numbers used in contacting the service. In addition, the grounding must specify, for each abstract type specified in the ServiceModel, an unambiguous way of exchanging data elements of that type with the service (that is, the serialization techniques employed).

OWL-S suffers from many problems. Firstly, OWL-S Profile though having elements for pre- and post conditions, still does not mandate any concrete formalism for specifying the conditions. In our approach, we have shown how pre conditions and effects can be modeled. This limitation of OWL-S Profile is directly related to the problem of describing relations between input and output parameters. Secondly, the OWL-S Process Model does not have a formal execution semantics. Note, that the execution semantics presented in [AHS02] is of one of the first versions of DAML-S. Since OWL-S is an OWL ontology, OWL-S Process Model has a description logics semantics. However description logics can not capture behavioral semantics, in particular that of changing A-Boxes of the participating actors and variables. In our approach, we model the behaviour with π-calculus that has a thoroughly investigated execution semantics. In particular, the correct semantics of variables is ensured by adding and removing sameIndividual axioms. We believe, that the non-availability of formal semantics for OWL-S Process Model is the major reason for the non-availability of matchmakers based on OWL-S Process Model. The OWL-S matchmaker presented in [SPAS03] actually matches the subsumptions among the input and output parameter of a web service, which is, as discussed in Chatper 1, not sufficient for automation. The third problem with OWL-S is the separation of OWL-S Profile and OWL-S Process Model. Due to this separation, one needs to relate the elements in OWL-S Profile and elements in OWL-S Process Model that actually describe the same artifacts. Note, that there is currently no convincing argument known for this separation. In our approach, we do not have this problem. Our formalism does not require to model the same artifacts multiple times and therefore does not require to relate the descriptions of the same artifacts. OWL-S however has identified and modeled some important non-functional properties of Web services, which we have abstracted from in this work. We believe, that our formal model can be used to provide formal execution semantics to OWL-S.

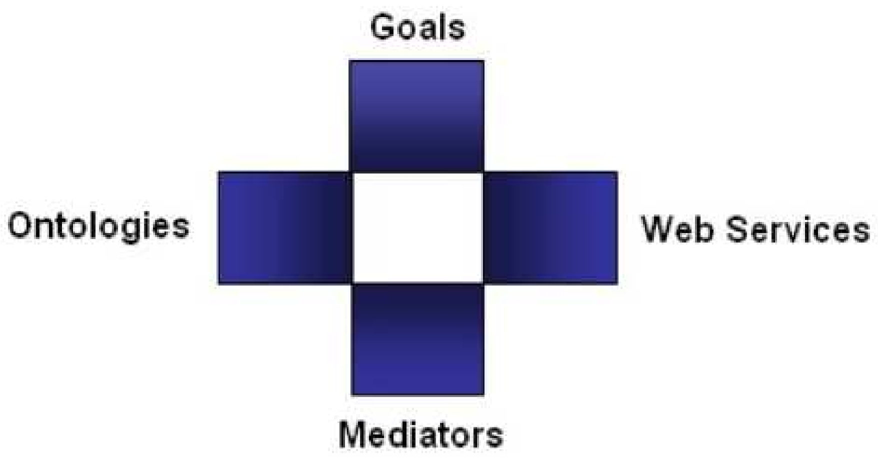


Figure 5.5.: WSMO Top Level Elements

5.3.2. WSMO

WSMO (Web Service Modeling Ontology) provides a conceptual underpinning and a formal language for semantically describing web services in order to facilitate the automatization of discovering, combining and invoking electronic services over the Web [RKL$^+$05]. WSMO builds on the Web Service Modeling Framework (WSMF) [FB02]. WSMO is rather a formalized bird-eye view than a concrete Web service description language. The WSMO working group develops a conceptual model for the semantic description of web services. The WSML working group develops a formal language to describe the formal model developed by the WSMO working group. The WSMX working group works on the development of a execution environment for semantic web services.

WSMO

The conceptual model WSMO consists of four main elements, namely *Ontologies*, *Goals*, *Web Services* and *Mediators* (refer to Figure 5.5).

Ontologies provide the terminology used by other WSMO elements to describe the relevant aspects of the domains of discourse. *Web services* describes the computational entity providing access to services that provide some value in a domain. These descriptions comprise the capabilities, interfaces and internal working of the Web service. *Goals* represent user desires, for which fulfilment could be sought by executing a Web service. Ontologies can be used for the domain terminology to describe the relevant aspects. Goals model the user view in the Web service usage process. Finally, Mediators describe elements that overcome interoperability problems between different WSMO elements. Mediators are the core concept to resolve incompatibilities on the data, process and protocol level, i.e. in order to resolve mismatches between different used terminologies (data level), in how to communicate between Web services (protocol level) and on the level of combining Web services (and goals) (process level).

Ontologies

```
Class ontology
      hasNonFunctionalProperties type nonFunctionalProperties
      importsOntology type ontology
      usesMediator type ooMediator
      hasConcept type concept
      hasRelation type relation
      hasFunction type function
      hasInstance type instance
      hasAxiom type axiom
```

The following non-functional properties recommended for characterizing ontologies are: Contributor, Coverage, Creator, Date, Description, Format, Identifier, Language, Owner, Publisher, Relation, Rights, Source, Subject, Title, Type, Version.

Importing Ontologies allows a modular approach for ontology design. Importing can be used as long as no conflicts need to be resolved, otherwise an Ontology-Ontology-Mediator needs to be used.

When importing ontologies, most likely some steps for aligning, merging, and transforming imported ontologies have to be performed. For this reason and in line with the basic design principles underlying the WSMF ontology mediators (ooMediator) are used when an alignment of the imported ontology is necessary.

```
Class concept
      hasNonFunctionalProperties type nonFunctionalProperties
      hasSuperConcept type concept
      hasAttribute type attribute
      hasDefinition type logicalExpression multiplicity =
      single-valued
```

Concepts constitute the basic elements of the agreed terminology for some problem domain. From a high-level perspective, a concept – described by a concept definition – provides attributes with names and types. Furthermore, a concept can be a subconcept of several (possibly none) direct superconcepts as specified by the isA-relation. Precise semantics of the concepts is defined in WSML as it depends on the concrete ontology formalization language.

A function is a special relation, with an unary range and a n-ary domain (parameters inherited from the relation), where the range value is functionally dependent on the domain values. In contrast to a function symbol, a function is not only a syntactical entity but has a defined semantics that allows to actually evaluate the function if concrete input values for the parameters are given. That means that one can actually substitute the (ground) function term in some expression by its concrete value. Functions for example can be used to represent and exploit built-in predicates of common datatypes. Their semantics can be captured externally by means of an oracle, or can be formalized by assigning a logical expression to the hasDefinition property inherited from relation.

Instances are either defined explicitly or by a link to an instance store, i.e., an external storage of instances and their values. Again, the semantics of the class-instance relationship, is determined by the concrete ontology language.

Finally, an axiom is a logical expression together with its annotations.

Web Services WSMO Web service descriptions consist of functional, non-functional and the behavioral aspects of a Web service. A Web service is a computational entity which is able (by invocation) to achieve a user's goal. A service in contrast is the actual value provided by this invocation. Thereby a Web service might provide different services, such as for example Amazon can be used for acquiring books as well as to find out an ISBN number of a book.

```
Class webService
      hasNonFunctionalProperties type nonFunctionalProperties
      importsOntology type ontology
      usesMediator type {ooMediator, wwMediator}
      hasCapability type capability multiplicity = single-valued
      hasInterface type interface
```

A Web service can import ontologies using ontology mediators (ooMediator) when steps for aligning, merging, and transforming imported ontologies are needed. A Web service can use wwMediators to deal with process and protocol mediation.

Goals Goals are representations of an objective for which fulfillment is sought through the execution of a Web service. Goals can be descriptions of Web services that would potentially satisfy the user desires. The following listing presents the goal definition:

```
Class goal
      hasNonFunctionalProperties type nonFunctionalProperties
      importsOntology type ontology
      usesMediator type {ooMediator, ggMediator}
      requestsCapability type capability multiplicity = single-valued
      requestsInterface type interface
```

A capability defines the Web service by means of its functionality. An interface describes how the functionality of the Web service can be achieved (i.e. how the capability of a Web service can be fulfilled) by providing a twofold view on the operational competence of the Web service:

- choreography decomposes a capability in terms of interaction with the Web service.
- orchestration decomposes a capability in terms of functionality required from other Web services.

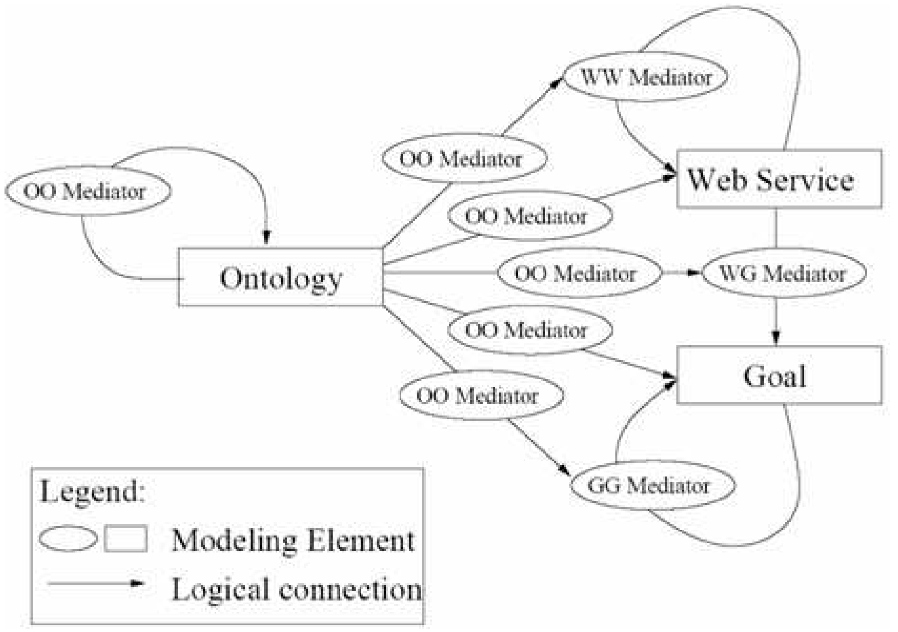


Figure 5.6.: Interplay of WSMO Mediators

This distinction reflects the difference between communication and cooperation. The choreography defines how to communicate with the Web service in order to consume its functionality. The orchestration defines how the overall functionality is achieved by the cooperation of more elementary Web service providers

Mediators WSMO distinguishes among four different types of mediators:

- ggMediators: Mediators that link two goals. This link represents the refinement of the source goal into the target goal or state equivalence if both goals are substitutable.
- ooMediators: Mediators that import ontologies and resolve possible representation mismatches between ontologies.
- wgMediators: Mediators that link Web services to goals, meaning that the Web service (totally or partially) fulfills the goal to which it is linked. wgMediators may explicitly state the difference between the two entities and map different vocabularies (through the use of ooMediators).
- wwMediators: Mediators linking two Web services.

In contrast to OWL-S WSMO differentiates between the descriptions of goals and web services.

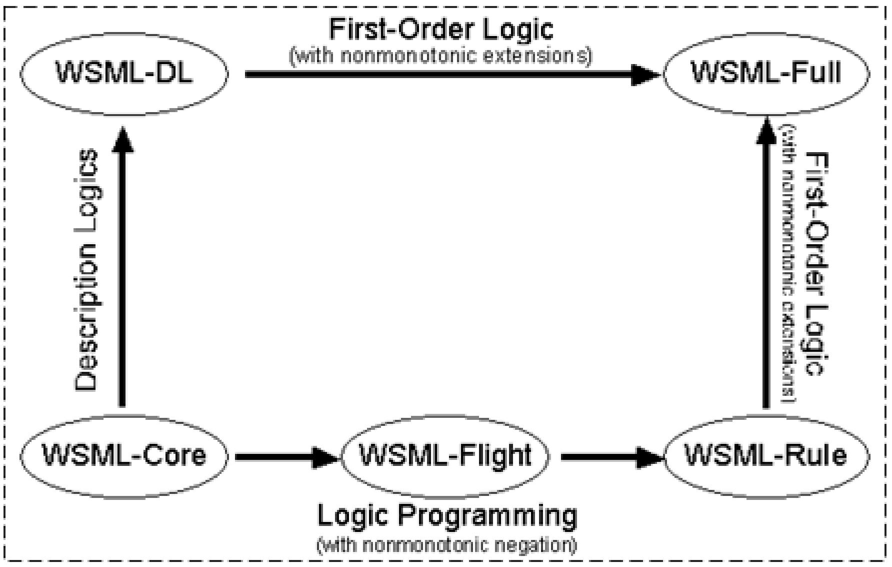


Figure 5.7.: WSML Language Variants

WSML

The Web Service Modeling Language (WSML) aims at providing means to formally describe all the elements defined in WSMO. The different variants of WSML correspond to different levels of logical expressiveness and the use of different languages paradigms. More specifically, Description Logics, First-Order Logic and Logic Programming are taken as starting points for the development of the WSML language variants. The basic language with the least expressive power is WSML-Core. This language is defined by the intersection of Description Logic and Horn Logic, based on Description Logic Programs. The main features of the language are the support for modeling classes, attributes, binary relations and instances. Furthermore, the language supports class hierarchies, as well as relation hierarchies. WSML-Core provides support for datatypes and datatype predicates.

WSML-DL is an extension of WSML-Core which fully captures the Description Logic SHIQ(D), which captures a major part of the (DL species of the) Web Ontology Language OWL, with a datatype extension based on OWL-E, which adds richer datatype support to OWL. WSML-Flight is an extension of WSML-Core with such features as meta-modeling, constraints and nonmonotonic negation. WSML-Flight is based on a logic programming variant of F-Logic [Kifer et al., 1995] and is semantically equivalent to Datalog with inequality and (locally) stratified negation. As such, WSML-Flight provides a powerful rule language. WSML-Rules is an extension of WSML-Flight in the direction of Logic Programming. The language captures several extensions such as the use of function symbols and unsafe rules. WSML-Full unifies WSML-DL and WSML-Rule under a First-Order umbrella with extensions to support the nonmonotonic negation of WSML-Rule. It is yet to be investigated which kind of formalisms are required to achieve this. Possible formalisms are Default Logic, Circumscription and Autoepistemic Logic.

The WSMX-Architecture consists of a set of loosely-coupled components that can be plugged-in and plugged-out from the system. For each component, public interfaces are defined, that can be either accessed by components provided with the reference implementation, or by components provided by independent component providers. Since, execution of Web services is not our focus in this work, we will not discuss WSMX in further detail. For more details on WSMX we refer to [HCM+05].

WSMO is a conceptual model and the modeling approach that we have presented may be seen as a reference implementations of WSMO since we support all top level elements of WSMO, namely ontologies, goals[2],Web services and mediation. So, our formalism for modeling Web services is rather on the level of WSML. However, even if the WSMO conceptual model addresses the need of choreography and orchestration, WSML does not provide any formalism to model them. From this perspective, our Web service modeling language implements WSMO better than WSML. Furthermore, WSMO's definition of effects is still vague. We have shown that invoking a Web service can have two kinds of effects, namely input/output activities and updates. We have also shown how the updates can be modeled with the help of local operations.

5.3.3. WSDL-S

WSDL-S does not provide a formalism for describing Web services semantically. Rather, it extends WSDL by providing extensibility elements to connect semantic descriptions to WSDL documents. It is a means to add semantics inline to WSDL for (1) inputs and outputs (2) Operations and (3) Service categorization. It is agnostic to a particular ontology representation language. WSDL-S is defined by extending the WSDL XML schema.

WSDL-S augments the expressivity of WSDL with semantics by employing concepts analogous to those in OWL-S while being agnostic to the semantic representation language [AFM+05]. The advantage of this evolutionary approach to adding semantics to WSDL is multi-fold. First, users can, in an upwardly compatible way, describe both the semantics and operation level details in WSDL-S a language that the developer community is familiar with. Second, by externalizing the semantic domain models, we take an agnostic approach to ontology representation languages. This allows Web service developers to annotate their Web services with their choice of ontology language (such as UML or OWL). While it is noted that the theoretical underpinnings of OWL-S in de-

[2]we will address goals in the next part of this work.

scription logic makes it a richer language for representing semantics, authors believe that extending the industry standards such as WSDL to include semantics is a more practical approach for adoption. Moreover, authors claim that by externalizing the semantic domain models in their proposal, they allow for richer representations of domain concepts and relationships in languages such as OWL and UML, thereby bringing together the best of both worlds.

So, our approach is complementary to WSDL-S in a sense that the descriptions of Web services with our formalism can be connected to WSDL-S documents.

5.3.4. BPEL4WS

BPEL4WS [ACD+03] is a popular formalism for modeling business processes in the Web. It combines XLANG and WSFL. However, it still lacks formal semantics and reasoning procedures. Therefore, our work is complementary to BPEL4WS as our formal model may be used in its extended form to specify formal semantics for BPEL4WS, which is needed in order to prove certain properties of reasoning algorithms based on BPEL4WS. Note that all the attempts to define formal semantics of BPEL4WS cover only the dynamic behavior of BPEL4WS and resource schemas that are an essential part of multi party business processes are not considered. Furthermore, the main focus of BPEL4WS lies in the execution of business processes, which we have abstracted from. BPML[3] is similar to BPEL4WS in the sense that it focuses more on the execution of a business process than on reasoning about properties of a process.

Perhaps, the work that is closest to our work is [BCG+05]. In [BCG+05] an approach is presented to characterize Web services with their transition behavior and their impacts on the real world (modeled as relational databases). Our Web service modeling approach though follows similar thoughts is different as we have presented a concrete syntax and semantics for modeling Web services, whereas [BCG+05] only uses a mathematical model of transition systems. With our language such transitions systems can be modeled. Another difference between our approach and [BCG+05] is that we model the local knowledge bases of the participating actors with decidable description logics, whereas [BCG+05] proposes to use relational databases.

Finally, to the best of our knowledge, ours is the only work that has addressed the need of credential based access control policies and can model and reason about them together with other aspects of Web services.

5.3.5. Access Control Related Work

In the area of service oriented computing there are already several approaches for declaratively modeling the user's objectives; mainly in terms of policies. On the one hand, there are XML-based approaches, like WS-Security, XACML, EPAL, etc.. These approaches

[3]http://www.bpmi.org

allow to model constraints about domain-specific attributes of a service. XACML has been approved by OASIS and that promises to standardize policy management and access decisions. However, XACML focusses more on technical issues and addresses how the access control can be enforced. EPAL and XACML specifications greatly overlap and do very similar things in slightly different ways. As a consequence, the user has to learn the different approaches and work with different policy tools. Furthermore, it is not possible to specify a policy that combines privacy and communication security concerns such as: send sensitive content over secured lines only. WS-Policy introduces a logic framework that allows domain-specific policy assertions to be plugged in. Nevertheless, the supported assertions are very simplistic in nature and still require the respective native policy interpreters. The major disadvantage of XML is that the semantics is contained implicity in the expressions. Meaning arises only from the shared understanding derived from human consensus. This leads to extra manual work for software engineers and could easily result in fragmentation.

Security-related ontologies to markup DAML-S [ABH+02] elements such as input and output parameters with respect to their security characteristics, such as encryption and digital signatures have been developed in [DKF+03, KFD+04]. [KFJ04] gives an short introduction to Rei, case studies, use cases and open issues. However, the mechanism described in the paper requires clients to send their privacy policies and permissions to a Web service provider, which is not always wishful. Nevertheless, the authors identify the enforcement problem as an open issue. [PKKJ04] introduces an enforcement architecture based on a policy engine and the policy enforcement mechanism for pervasive environments. The policy engine reasons over policies described in Rei and uses Prolog for its reasoning engine. [KFJ03] discusses the policy language Rei in more detail. However, neither [PKKJ04] nor [KFJ03] provide Rei's mapping to Prolog. So, it is not clear what the policy engine acutally does. Since Rei requires a special reasoner, it is not clear, what is the added value of Rei as compared to pure syntax based (mostly XML) approaches except that it is more expressive.

Our work is complementary to the existing approaches as it also addresses the need of machine understandable specification of certification policies that are specified by the certification authorities. It also presents how Web services providers can use such semantically-rich specifications for defining their access control policies. Consequently, our approach covers a broader spectrum and also shows the added value of specifications with formal semantics within the context of access control. We also address the issue of enforcement which can not be ignored while dealing with security related aspects. Finally, our approach is completely based on description logics which is the formalism behind the W3C standard OWL (Web Ontology Language). Consequently, we do not require the users and providers of semantic Web services to install a special reasoner.

5.4. Conclusion

In this chapter, we have developed a novel formalism for modeling the behaviour of Web services. We have shown in Chapter 3 how resources and schemas can be modeled with an expressive description logic like $\mathcal{SHOIN}(\mathcal{D})$ and DL-safe rules and hence in the spirit of Web since OWL-DL, a W3C standard is based on $\mathcal{SHOIN}(\mathcal{D})$.

In Chapter 4, we showed how non-functional properties of Web services can be modeled such that users can build their trust in them. In a capability based access control system a Web service provider specifies the access requirements for his service in terms of required properties. For gaining access, users have to prove that they satisfy these properties. To do so, they need to present certificate chains that prove a delegation chain from a trusted certification authority to the required property. Our approach shows how certification policies can be specified in the semantic Web in a machine understandable way, which makes them suitable for automatic verification of the compatibility between access control policies and governmental or self imposed laws. Further, delegation structures of CAs can be made explicit in our approach, which allows CAs to handle revocation of (delegation) certificates in an online manner.

We then showed, how the behavior of an agent can be described with π-calculus. We augmented π-calculus channels with protocols and types in order to model agents that run in mixed environments. We introduced the notion of local operations for modeling updates in the knowledge bases of the agents and integrated access control policies as conditional processes. The formalism allows to model the resources involved in a Web service, credentials of the involved parties and the dynamic behavior of a Web service in a unified way.

Part III.

Semantic Matchmaking of Web Services

In the previous part, we have seen how Web services can be described semantically. We developed an expressive formalism for modeling functional and non-functional properties of Web services. The main reason of developing such a formalism was to enable expressive matchmaking of Web service, which is our focus in this part of our work. We develop two matchmaking approaches, which we call *goal based matchmaking* and *simulation based matchmaking* respectively. Goal based matchmaking covers the classical case, in which a user specifies the desired constraints on the properties of Web services and every Web service that satisfies the constraints is considered as a match. Simulation based matchmaking deals with the problem which is hardly addressed in the literature though very common in practice. It deals with the case in which two Web service descriptions are matched directly to find out whether one Web service can be replaced by another without changing the overall behaviour of the system they are embedded in.

In order to achieve goal based matchmaking, one needs a formalism for specifying the goals, the desired constraints. Even though we use the goal specification formalism only for matchmaking in this work, such a language can serve for specifying constraints for automatic composition of Web services as well. Therefore, we develop a goal specification formalism in Chapter 6 separately from the goal based matchmaking algorithm. Having a goal specification formalism, we develop the goal based matchmaking algorithm in Chapter 7. In Chapter 8, we turn our attention to simulation based matchmaking. We develop an algorithm that checks whether two Web service descriptions are equivalent.

6. Goal Specification

In the previous part, we have developed an expressive formalism for describing Web services semantically. In order to perform matchmaking that uses all the expressiveness of the formalism, one first needs a technique that allows a user to specify properties of desired web services. The properties specified by a user are constraints that a Web service must fulfill in order to be considered as a match. In this chapter, our focus is on developing a formalism for specification of various types of constraints on the properties of Web services.

In Chapter 2, we have identified types of constraints a user may wish to specify in order to search for desired Web services. In most cases, a user is a potential client of a Web service. That is, he is looking for Web services that he can incorporate in his system. However, in some cases a user only wants to ensure whether a Web is compliant with certain policies. For example, a Web service provider himself may wish to check whether the process underlying his Web service behaves as it is supposed to, before he publishes its description. In other example, a user might be just looking for Web services that uses secure communication because he is doing a survey for his student research project.

So, we do not want to restrict the search possibilities for a user to matchmaking based on input/output types, as it is the case in existing approaches. In our approach, a user must be able to select Web services by any properties that we model, since those who describe a Web service do not know in which context a user may be interested in the Web service.

In Chapter 2, we have identified that a user may wish to specify constraints on the resources and the actors that may be involved in the run of a Web service and on the choreography and orchestration of Web services. The requirements regarding resources, that is Requirment 3, 4, 5 and 6 are directly supported due to availability of the query language SPARQL for expressive description logics like $\mathcal{SHOIN}(\mathcal{D})$. We have shown in Chapter 5 that checking whether an actor's credentials are enough to prove his trustworthiness according to some access control policy, can be performed by a description logic query. So, $\mathcal{SHOIN}(\mathcal{D})$ with DL-safe rules and SPARQL fulfill the Requirements 7 and 8. Therefore, our first aim in this chapter is to develop a formalism for specifying constraints on the choreography (Requirments 9) and orchestration (Requirments 10) of Web services. Then, we will focus on combining different types of constraints to build complex constraints in order to fulfill the Requirement 11.

6.1. Modal Logics for Processes

The application of modal and temporal logics to process calculi is part of a line of program verification going back to the 1960s and program schemes and Floyed-Hoare logic. Originally the emphasis was on *proof*: Floyd-Hoare logic allows one to make assertions about programs, and there is a proof system to verify these assertions. This line of work has continued and today there are highly sophisticated theories for proving properties of programs, and equally sophisticated machine support for these theories. However, the use of proof systems has some disadvantages, and one hankers after a more purely algorithmic approach for simple problems. One technique was pioneered by Manna and Pneuli [MP69], who turned program properties into questions of satisfiability or validity in first order logic, which can then be attacked by means that are not just proof-theoretic; this idea was later applied by them to linear temporal logics.

During tre 1970s, the theory of program correctness was extended by investigating more powerful logics, and studying them in a manner more similar to the traditions of mathematical logic. A family of logics which received much attention was that of dynamic logics, which can be seen as extending the ideas of Hoare logic [Pra76]. Dynamic logics are modal logics, where the different modalities correspond to the execution of difference programs – the formula $\langle\alpha\rangle\phi$ is read as “it is possible for α to execute and result in a state satisfying ϕ”. The programs may be of any type of interest; the variety of dynamic logic most often referred to is a propositional language in which the programs are built from atomic programs by regular expression constructors; henceforth, Propositional Dynamic Logic, PDL refers to this logic [FL79]. PDL is interpreted with respect to a model on a Kripke structure, formalizing the notion of the global state in which programs execute and which they change – each point in the structure corresponds to a possible state, and programs determine a relation between states giving the changes effected by the programs.

Once one has the idea of a modal logic defined on a Kripke structure, it become quite natural to think the finite case and write programs which just check whether a formula is satisfied. This idea was developed in early 80s by Clarke, Emerson, Sistla and others. They worked with a logic that has much simpler modalities than PDL–in fact, it has just a single “next state” modality – but which has built-in temporal connectives such as “until”. This logic is CTL, and it and its extensions remain some of the most popular logics for expressing properties of systems [CES86].

Meanwhile, the theory of process calculi was being developed in the late 1970s, most notably my Milner. An essential component was the use of labeled Kripke structures (“labeled transitions systems”) as a raw model of concurrent behavior. An important difference between the use of Kripke structures here and their use in program correctness was that the states are the behavior expressions, which model concurrent systems, themselves and the labels on the accessibility relation (the transitions) are simple actions (and not programs). Hennessy and Milner [HM80] introduced a primitive modal logic in which the modalities refer to actions: $\langle\alpha\rangle\phi$ means "it is possible to do an α action and then have ϕ be true" and its dual $[\alpha]\phi$ meaning "ϕ holds after every α action". Together with the usual boolean connectives, this given Hennessy-Milner logic, HML [HM80]. However, HML is inadequate to express many properties, as it has no means of saying "always in the future" or other temporal connectives–except by allowing infinitary conjunction. Using an infinitary logic is undesirable both for the obvious reason that infinite formulae are not amenable to automatic processing, and because infinitary logic gives much more expressive power than is needed to express temporal properties.

In 1983, Dexter Kozen published a study of a logic that combined simple modalities, as in HML, with fixpoint operators to provide a form of recursion [Koz83]. This logic, the modal mu-calculus has become probably the most studied of all temporal logics of programs. It has a simple syntax, an easily given semantics, and yet the fixpoint operators provide immense power. Most other temporal logics, such as the CTL family, can be seen as fragments of the modal mu-calculus. Moreover, this logic lends itself to transparent model-checking algorithms.

6.2. Early Logics

HML, Hennessy-Milner Logic [HM80], is a primitive modal logic of action. The syntax of HML has, in addition to the boolean operators, a modality $\langle\alpha\rangle$, where α is a process *action*. A structure for the logic is just a labeled transition system. Atomic formulas of the logic are the constants `tt` and `ff`. The meaning of $\langle\alpha\rangle$ is "it is possible to do an α-action to a state where ϕ holds". The formal semantics is given in the obvious way by inductively defining when a state (a process) of a transition system has a property; for example $E \models \langle\alpha\rangle$ iff $\exists F.E \xrightarrow{\alpha} F$ and $F \models \phi$. We may also add some notion of variable or atomic proposition to the logic, in which case we provide a valuation which maps a variable to the set of states at which it holds. The expressive power of HML in this form is quite weak; obviously a given HML formula can only make statements about a given finite number of steps into the future. HML was introduced not so much as a language to express properties, but rather as an aid to understanding process equivalence: two processes are equivalent iff they satisfy exactly the same HML formulae. To obtain the expressivity desired in practice, we need stronger logics.

The logic PDL, Propositional Dynamic Logic [FL79, Pra76], as mentioned above, is both a development of Floyd-Hoare style logics, and a development of modal logics. PDL is an extension of HML in the circumstance that the action set has some structure. There is room for variation in the meaning of action, but in the standard logic, a program is considered to have a number of *atomic actions*, which in process algebraic terms are just process actions, and α is allowed to be a regular expression over the atomic actions; $a, \alpha, \beta, \alpha \cup \beta$, or $\alpha*$. We may consider atomic actions to be uninterpreted atoms; but in the development form Floyd-Hoare logics, one would see the atomic actions as, for example, assignments statement in a **while** program.

PDL enriches the labels in the modalities of HML. An alternative extension of HML is to include further modalities. The branching time logic CTL, Computational Tree Logic [CE82], can be described in this way as an extension of HML, with some extra "temporal" operators which permit expression of liveness and safety properties. For the semantics we need to consider "runs" of a process. A run from an intial state or process E_0 is a sequence $E_0 \xrightarrow{a_1} E_1 \xrightarrow{a_2} \ldots$ which may have finite or infinite length; if it has finite length then its final process is a "sink" process which has no transitions. A run $E_0 \xrightarrow{a_1} E_1 \xrightarrow{a_2} \ldots$ has the property $\phi \mathbf{U} \psi$, ϕ until ψ, if there is an $i \geq 0$ such that $E \models \psi$ and for all $j : 0 \leq j \leq i, E_j \models \phi$.

$$\begin{tikzcd}
E_0 \arrow[r, "a_1"] & E_1 \arrow[r, "a_2"] & \cdots & E_i \arrow[r, "a_{i+1}"] & \cdots \\
\models & \models & \cdots & \models & \\
\phi & \phi & & \psi &
\end{tikzcd}$$

The formula $\mathbf{F}\phi = (\mathtt{tt}\mathbf{U}\phi)$ means "ϕ eventually holds" and $\mathbf{G}\phi = \neg(\mathtt{tt}\mathbf{U}\neg\phi)$: "$\phi$ always hold". For each "temporal" operator such as $\mathbf{U}$ there are two modal variants, a strong variant ranging over all runs of a process and a weak variant ranging over some run of a process. We preface a strong version with $\forall$ and a weak variant with $\exists$. If HML is extended with the two kinds of until operator the resulting logic is a slight but inessential variant of CTL. The formal semantics is given by inductively defining when a state (process) has a property. For instance $E \models \forall[\phi\mathbf{U}\psi]$ iff every run of E has the property $\phi\mathbf{U}\psi$.

CTL has variants and enrichments such as CTL* [EL86] and ECTL [VW83]. These allow free mixing of path operators and quantifiers: for example, the CTL* formula $\forall[P\mathbf{U}\exists\mathbf{F}Q]$ is also a CTL formula, but $\forall[P\mathbf{U}\mathbf{F}Q]$ is not, because the $\mathbf{F}$ is not immediately governed by a quantifier. Hence expansions also cover traditional temporal logics–that is, literally logics of time–as advocated in Manna and Pnueli and others. In this view, time is a linear sequence of instants, corresponding to the the states of just one execution path through the program. One can define a logic on paths which has operators $O\phi$ meaning "in the next instant (on this path) ϕ is true", and $\phi\mathbf{U}\psi$ meaning "ϕ holds until ψ holds (on this path)"; and then a system satisfies a formula if all execution paths satisfy the formula – in CTL* terms, the specification is a path formula with a single outermost universal quantifier. One can also extend PDL with temporal operators, as in process logic.

6.3. Introduction to mu-calculi

The defining feature of mu-calculi is the use of fixpoint operators. The use of fixpoint operators in program logic goes back at least to De Bakker, Part and Scott [Par69].

However, their use in modal logics of programs dates from work of Pratt, Emerson and Clarke and Kozen. Pratt's version [Pra82] used a fixpoint operator like the minimization operator of recursion theory, and this has not been further studied. Emerson and Clarke added fixed points to a temporal logic to capture fairness and other correctness properties [EC80]. Kozen's introduced the modal mu-calculus as we use it today, and established a number of basic results [Koz83].

Suppose that $\mathcal{S}$ is the state space of some system. For example $\mathcal{S}$ could be the set of all processes reachable from by arbitrary length sequences of transitions from some initial process E. One way to provide semantics of a state-based modal logic is to map formulae ϕ to sets of states, that is to elements of $2^{\mathcal{S}}$. For any formula ϕ this mapping is given by $[\![\phi]\!]$. The idea is that this mapping tells us in which states each formula holds. If we allow our logic to contain variables with interpretations ranging over $2^{\mathcal{S}}$, then we can view the semantics of a formula with a free variable, $\phi(Z)$, as a function $f : 2^{\mathcal{S}} \rightarrow 2^{\mathcal{S}}$. If we take the usual lattice structure on $2^{\mathcal{S}}$, given by set inclusion, and if f is a monotonic function, then by the Knaster-Tarski theorem we know that f has fixed points, and indeed has a unique maximal and a unique minimal fixed point. So we could extend our basic logic with a minimal fixpoint operator μ, so that $\mu Z.\phi(Z)$ is a formula whose semantics is the least fixed point of f; and similarly a maximal fixpoint operator ν, so that $\nu Z.\phi(Z)$ is a formula whose semantics is the greatest fixed point of f.

The reason we might want to do this is that, as in domain theory for example, it provides a semantics for *recursion*. Writing recursive modal logic formulae may not be an immediately obvious thing to do, but it provides a neat way of expressing all the usual operators of temporal logics. For example, consider the CTL formula $\forall\mathbf{G}\phi$, "always ϕ". Another way of expressing this is to say that it is a property X such that if X is true, then ϕ is true, and wherever we go next, X remains true; so X satisfies the modal operation

$$X = \phi \wedge [-]X$$

where $=$ is logical (truth-table) equivalence, and $[-]X$ means that X is true at every immediate successor. So we could write this formula as $?X.\phi \wedge [-]X$. But what is '?'? Which fixed point is the right solution of the equation – least, greatest, or something in between? We may argue thus: if a state satisfies *any* solution X' of the solution, then surely it satisfies $\forall\mathbf{G}\phi$. Hence the meaning of the formula is the largest solution, $\nu X.\phi \wedge [-]Z$.

On the other hand, consider the CTL property $\exists\mathbf{F}\phi$, "there exists a path on which ϕ eventually holds". We could write this recursively as "Y holds if either ϕ holds now, or there is some successor on which Y is true":

$$Y = \phi \vee \langle - \rangle Y.$$

This time, it is perhaps less obvious which solution is correct. However we can argue that if a state satisfies $\exists\mathbf{F}\phi$, then it surely satisfies any solution Y' of the equation; and so we want the least such solution.

It is easy to notice that the equations do no capture the meaning of the English recursive definition–we should have written

$$X \Rightarrow \phi \wedge [-]X$$

and

$$Y \Leftarrow \phi \vee \langle - \rangle Y,$$

or in terms of the semantic function,

$$[\![X]\!] \subseteq [\![\phi \wedge [-]X]\!]$$

and

$$[\![Y]\!] \supseteq [\![\phi \vee \langle - \rangle]\!].$$

6.3.1. Syntax and Semantics of μ-calculus

Let Var be an (infinite) set of variables names, typically indicated by $X, Y, Z \ldots$; let *Prop* be a set of atomic propositions, typically indicated by $P, Q, \ldots$; and let A be a set of action typically indicated by $a, b, \ldots$. The set of formulae (with respect to $(Var, Prop, A)$ is defined as follows:

- P is a formula
- Z is a formula
- If ϕ_1 and ϕ_2 are formulae, so is $\phi_1 \wedge \phi_2$
- If ϕ is a formula, so is $[a]\phi$
- If ϕ is a formula, so is $\neg\phi$
- If ϕ is a formula, then $\nu Z.\phi$ is a formula, provided that every free occurrence of Z in ϕ occurs positively, i.e. within the scope of an even number of negations.

It is usually convenient to introduce derived operators defined by de Morgan duality and work in positive form:

- $\phi_1 \vee \phi_2$ means $\neg(\neg\phi_1 \wedge \neg\phi_2)$
- $\langle a \rangle \phi$ means $\neg[a]\neg\phi$
- $\mu Z.\phi(Z)$ means $\neg\nu Z.\neg\phi(\neg Z)$

Note the triple use of negation in μ, which is required to maintain the positivity. A formula is said to be in *positive form* if it written with the derived operators so that $\neg$ only occurs applied to atomic propositions. It is in *positive normal form* if in addition all bound variables are distinct. Any formula can be put into positive normal form by use of de Morgan laws and α-conversion.

Definition 20 (Structure)**.** A *structure* $\mathcal{T}$(over $Prop$, A) is a labeled transition systems, namely a set $\mathcal{S}$ of states and a transition relation $\rightarrow \subseteq \mathcal{S} \times A \times \mathcal{S}$, together with an interpretation $\mathcal{V}_{Prop} : Prop \rightarrow \mathcal{P}(\mathcal{S})$ for the atomic propositions. We often write $s \xrightarrow{a} t$ for $(s, a, t) \in \rightarrow$.

Given a structure $\mathcal{T}$and an interpretation $\mathcal{V} : Var \rightarrow \mathcal{P}(S)$ of the variables, the set $[\![\phi]\!]_{\mathcal{V}}$ of states satisfying a formula ϕ is defined as follows:

$$\begin{aligned}
[\![P]\!]_{\mathcal{V}} &= \mathcal{V}_{Prop}(P) \\
[\![Z]\!]_{\mathcal{V}} &= \mathcal{V}(Z) \\
[\![\neg\phi]\!]_{\mathcal{V}} &= \mathcal{S} - [\![\phi]\!]_{\mathcal{V}} \\
[\![\phi_1 \wedge \phi_2]\!]_{\mathcal{V}} &= [\![\phi_1]\!]_{\mathcal{V}} \cap [\![\phi_2]\!]_{\mathcal{V}} \\
[\![[a]\phi]\!]_{\mathcal{V}} &= \{s | \forall t. s \xrightarrow{a} t \Rightarrow t \in [\![\phi]\!]_{\mathcal{V}}\} \\
[\![\nu Z.\phi]\!]_{\mathcal{V}} &= \bigcup \{S \subseteq \mathcal{S} | S \subseteq [\![\phi]\!]_{\mathcal{V}[Z:=S]}\}
\end{aligned}$$

where $\mathcal{V}[Z := S]$ is the valuation which maps Z to S and otherwise agrees with $\mathcal{V}$. The semantics of the derived operators is defined by duality as follows:

$$\begin{aligned}
[\![\phi_1 \vee \phi_2]\!]_{\mathcal{V}} &= [\![\phi_1]\!]_{\mathcal{V}} \cup [\![\phi_2]\!]_{\mathcal{V}} \\
[\![\langle a \rangle\phi]\!]_{\mathcal{V}} &= \{s | \exists t. s \xrightarrow{a} t \wedge t \in [\![\phi]\!]_{\mathcal{V}}\} \\
[\![\mu Z.\phi]\!]_{\mathcal{V}} &= \bigcap \{S \subseteq \mathcal{S} | S \supseteq [\![\phi]\!]_{\mathcal{V}[Z:=S]}\}
\end{aligned}$$

6.3.2. Examples of Some Common Patterns

- **Until** The formula $\mu X.(G \vee (F \wedge \langle - \rangle \texttt{true} \wedge [-]X))$ describes F until G, as it can be read as: either G holds in the current state or sooner or later the process reaches a state in which G holds and until then F holds.

- **Always** The formula for "P is true along every a-path" can be written as $\nu Z.P \wedge [a]Z$.

- **Eventually** Having the next and until modalities, the modality eventually can be easily defined as `true Until` G.

6.4. Specification of Hybrid Formulas with Logic $\mathcal{B}$

μ-calculus in its pure form is a logic with arbitrary valuations for atomic propositions, but this is neither possible nor desirable in practice. In practice one needs to fix a language for specifying atomic propositions to be able to reason about "sensible" properties of a system. Note, that restricting the language for atomic propositions does not lead to better decidability, complexity etc. as it might appear at the first sight. The reason is that the basic μ-calculus operators are sufficient to expose any undecidability that be lurking in a system - for example, the halting property is just $\mu Z.[-]Z$.

In this section our aim is to develop a logic for reasoning about agents. We derive the logic $\mathcal{B}$ from μ-calculus for specifying constraints on Web service processes. μ-calculus abstracts from the meaning and structure of the propositions in the same way the π-calculus abstracts from the meanings of π-calculus names.

The main idea is to fix the language for atomic propositions to description logics queries. That is, we fix the valuation function $\mathcal{V}_{Prop}$ to description logic query answering. In the rest of this section, we describe how description logic queries are connected to μ-calculus formulas.

Credentials

As described in Chapter 5, an agent α has a set of certificate $\mathcal{C}_\alpha$ and agents trust other agents on basis of their credentials. As we have seen in Chapter 6, users can restrict the desired set of agents that may be involved in a Web service process.

In Chapter 5 and 6, we have shown how certifiable properties can be modeled as concept expressions. The fact that an agent α possesses a property P is modeled by a description logic class membership axiom $P(\alpha)$. Furthermore, trust as well as access control policies are modeled as description logic queries such that checking whether an agent can be trusted or granted access is reduced to checking whether the agent is in the answer of the corresponding query. Sometimes, we will refer to such a trust or access control query as agent selection query as it is used to select agents on the basis of their credentials. An example of such an agent selection query is the description logic concept $\mathsf{Adult} \sqcap \mathsf{Male}$ which denotes the set of all agents that are adults as well as male.

Propositions

The set of propositions of an agent corresponds to the facts in the knowledge base of the agent. The facts in the knowledge base of an agent at some point of time represent the set of propositions that are true at that point of time. Furthermore, the facts that can be derived from the explicit facts in the knowledge base also represent the propositions that are true.

Recall from Section 6.3.1 formulas of type atomic proposition. We give an atomic proposition P the structure $Q_F@Q_{Ag}$. The proposition $Q_F@Q_{Ag}$ is true if the answer of the query Q_F performed by an agent that is in the answer of the agent selection query Q_{Ag} on his local knowledge base is not empty.

Actions

We make similar structural extensions for an atomic action a that appears in formulas $\langle a\rangle\phi$ and $[a]\phi$ (refer to Section Recall from Section 6.3.1). We differentiate between two types of atomic actions, namely input actions and output actions. Since input actions receive values and output actions emit values, we will use $+$ and $-$ for input and output actions respectively to differentiate them for each other.

Input and output actions take place via communication channels. As defined is Chapter 5, a communication channel is characterized by a tuple (p, a, t), where p is a protocol type, e.g. HTTP, Email, Fax, Phone, a is the address and t is a message type. A communication channel is instantiated to receive or send messages of type t.

With expression

$$+(P)(A)(v_1{:}T_1, \ldots, v_m{:}T_m)@Q_{Ag}$$

in place of a in formulas $\langle a\rangle\phi$ and $[a]\phi$ we denote the set of input actions that are performed by an agent in the answer of the agent selection query Q_{Ag} and that can receive m values of types $T_1, \ldots, T_m$ respectively over a channel of protocol type P at address A and bind them to process variables $v_1, \ldots, v_m$. The process variables are used to remember the received values so that they can be referred to in other formulas as we will see below. $T_1, \ldots, T_m$ are description logic concepts.

With expression

$$-(P)(A)(q_1, \ldots, q_m)@Q_{Ag}$$

in place of a in formulas $\langle a\rangle\phi$ and $[a]\phi$ we denote the set of output actions that are performed by an agent in the answer of the agent selection query Q_{Ag} and that send m values which are answers of the resource selection queries $q_1 \ldots, q_m$ respectively along a channel of protocol type P and address A. The queries $q_1 \ldots, q_m$ can use the process variables, e.g. those bound by an input action. This way, it becomes possible to establish a relationship between inputs and outputs.

Consider a Web service depicted in Figure 6.1. The white circles denote the communication activities and the black circles denote the local operations. Suppose the book selling Web services has credentials above21. The Web service obtains a book description and a delivery address as input (refer to receive order activity in Figure 6.1). According to the execution semantics of an input activity the axioms

$$?\mathit{bookDesc} = \text{inBookDesc}$$

and

$$?\mathit{delAdd} = \text{ inDelAdd}$$

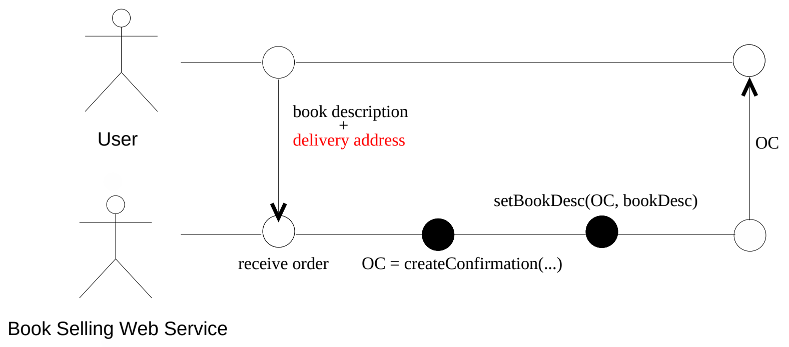


Figure 6.1.: Example of a query on the local knowledge base of an agent

are added in the local knowledge base of the Web service. After the input activity, the Web service performs two local operations, one for creating a new order confirmation and the other for setting the book description of the newly created order confirmation equal to the book description that has been entered by user. This happens by adding axioms

$$OC \text{ rdf:type } \mathsf{Confirmation}$$

and

$$OC \text{ hasBookDesc } ?bookDesc$$

respectively.

Now, suppose one wishes to know whether in the Web service process there exists an agent who is adult and who sends an order confirmation about the ordered book. Such a constraint can be formulated with our goal specification as:

$$\langle \overline{c}(\text{select ?x where ?x rdf:type Confirmation. ?x hasBookDesc inBookDesc})\rangle \mathtt{TRUE}@\mathsf{Adult}$$

Considering that the A-Box of the Web service contains the above four axioms, it is easy to see that the answer of the query

$$\text{select ?x where ?x rdf:type Confirmation. ?x hasBookDesc inBookDesc}$$

is not empty. Further considering that $\mathsf{above21} \sqsubseteq \mathsf{Adult}$ it is easy to see that the Web service provider fulfils the agent selection query too.

6.5. Related Work

OWL-S Matchmaker [SPAS03] uses OWL-S Profile for describing Web services as well as describing the goal. Recall, that even if OWL-S Profile is designed for modeling pre and post conditions in addition to the types of input and output parameters of the Web services, there is still no concrete formalism fixed for describing the conditions. As a result, the goal specification reduces to the types of input and output parameters. So, it neither allows the specification of relationships between inputs and outputs nor constraints on the temporal structure of a Web service.

Secondly, in scenarios, e.g. continuous double auctions[1], the allocation mechanism must be able to differentiate among offers and requests. Otherwise, the allocation mechanism may allocate a web service offer to another Web service offer or a request to another request. In the electronic market theory often Web service requests and Web service offers are not distinguished and both forms are just considered as bids. However, one still needs a mechanism to determine which bids will fit with which other bids. This is best illustrated with the example of a partner matching portal. Both women and men register their profiles, so there is differentiation between offers and requests. However according to gender information in the profiles, it is still possible to determine which profiles are matching candidates for which other profiles.

In case of OWL-S profiles however, there is no possibility to determine which other OWL-S profiles can be potential candidates for a match. We believe, this problem can be easily resolved if OWL-S introduces a simple temporal structure in OWL-S profile. The simple temporal structure would only fix whether the input happens before or after output. Currently, input and output are just properties and one does not know whether input takes place before or after output. Having such a simple constraint on the sequence of inputs and outputs, one can view OWL-S profiles with "input first" as provider bids wheres those with "output first" as client bids.

[APS05] presents a procedure for the verification of correctness claims about OWL-S process models. It presents a mapping of OWL-S Process to PROMELA that can then be verified using SPIN model checker. SPIN model checker supports LTL as a constraint specification language. However, the authors did not show that the mapping of OWL-S Process to PROMELA is correct. In order to prove the correctness of the mapping, one needs formal semantics of OWL-S Process Model, which does not exist. Secondly, the authors have abstracted from the ontological descriptions of resources involved in the Web service process. In OWL-S however the input and output activities have types. The intended semantics of input and output types is that a communication should only take place if the types are compatible. Abstracting from the types loses this useful feature. The fundamental problem with this work is that if authors' claim that OWL-S Process Model can be mapped to PROMELA, a process description language, is true, then there is no added value of OWL-S Process Model except being an XML syntax for PROMELA.

WSMO defines a conceptual service discovery model consisting of the elements shown

[1]A "continuous double auction" is one in which many individual transactions are carried on at a single moment and trading does not stop as each auction is concluded. The pit of the Chicago Commodities market is an example of a continuous double auction and the New York Stock Exchange is another. In those institutions a specialist matches bids and asking prices to find matches.

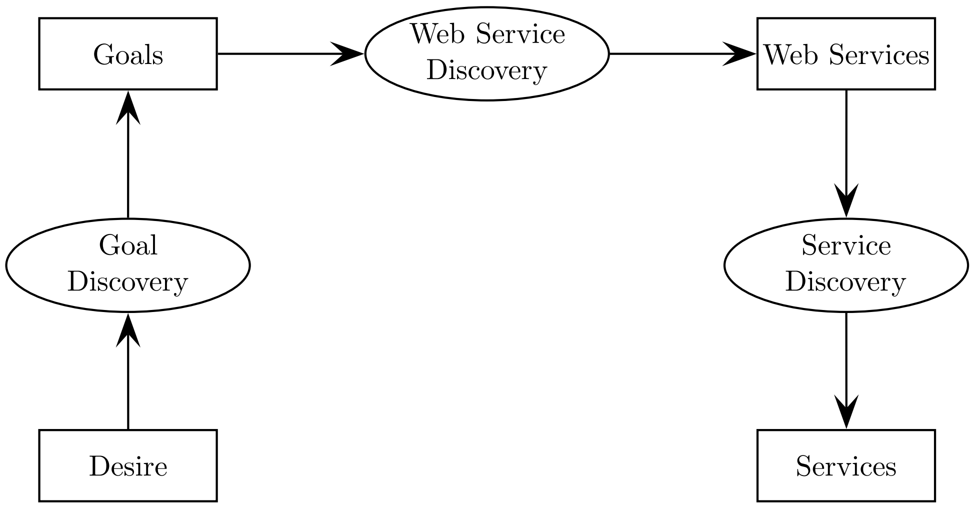


Figure 6.2.: Three major processes in WSMO service discovery

in Figure 6.2. WSMO discovery approach recommends the need for a “Goal Discovery” component that extracts user’s goal from his desire by making user’s desire more generic.

WSMO goal specification approach actually represents a general framework for describing goals and Web services. WSMO proposes the same formalism for specifying Web services as well as goals. In WSMO Web service discovery Web services as well as goals are represented as *sets of objects*.

A WSML goal specification is identified by the `goal` keyword and consists of capability and interfaces. A capability description consists of shared variables, pre- and post conditions as well as assumptions and effects. Due to the availability of shared variables WSML goals are more expressive than OWL-S Profiles. The interface description is used to specify the desired choreography and orchestration of a Web service. In other words, WSMO interface used in a goal specifies constraints on the dynamic behaviour of a Web service. However, currently there is no concrete proposal for describing the choreography and orchestration. Furthermore, due the separation of capabilities and dynamic behaviour already at the conceptual level, one would not be able to specify temporal constraints on effects. That is, constraints like “credit card should be charged (effect) should take place after delivery (choreography)“. Furthermore, WSMO goal specification does not address trust and access control policies. However, we believe that access control policies can be integrated in WSML as pre-conditions.

In [NS06] authors propose SWRL rules for specifying constraints on business processes and checking the satisfiability of the constraints using a description logic reasoner, such as KAON2. Firstly, SWRL is undecidable. Secondly, even if DL-safe rules, a decidable fragment of SWRL, are used, the semantics of a DL-safe rule is an implication and not a constraint. In other words, a DL-safe rule $A(x) \rightarrow B(x)$ means “if $A(x)$ holds then $B(x)$

holds" and not "something may/should/must hold". The latter semantics is the semantics of a query. In any case, authors use syntactically wrong DL-safe rules, because they use a predicate inside another predicate for example, $M(Property(X), Property(Y))$. Furthermore, DL-safe rules can not capture the semantics of the dynamic changes that occur during the execution of a business process. Such a DL-safe rules and query based approach is covered by our formalism by the formulae of type atomic propositions.

Major difference between [BCG+05] and our work is the choice of the logic for specifying constraints on Web services. [BCG+05] uses propositional dynamics logic (PDL) whereas we have used a more expressive logic known as μ-calculus, which also allows to specify fixpoint formulae.

6.6. Conclusion

In this chapter, we have developed an expressive formalism for modeling goals. We showed how temporal and security constraints as well as constraints on the objects involved in a Web service process can be specified with one logic. The logic developed in this chapter is a novel combination of description logics query language and μ-calculus. We have shown in Chapter 4, that credentials of an agent can be modeled with description logics and the verification of eligibility can be checked with description logic queries. Hence, our goal or request specification formalism presented in this chapter covers all the three aspects of the agents that we are considering in this work.

7. Goal based Matchmaking

Goal based matchmaking deals with finding Web services that fulfill some user defined requirements. These requirements may or may not be derived from the business goals automatically. Large number of web services and hence their descriptions lead to the need for automated Web services matchmaking procedures.

Current approaches for matchmaking focus on the interface description and not on the functionality of the Web service. Interface matching is important in order to ensure that a Web service can be actually embedded in a business process, i.e. to ensure that there are no type errors during the execution of a business process that incorporates Web services. However, when a user is looking for a Web service, he is not interested only in Web services, that he can actually execute but his primary interest is to find Web services that offers the functionality that he needs.

In chapters 5 and 6, we have seen how Web services can be modeled and how constraints can be specified formally. In this chapter our aim is to develop an algorithm that can check for a given Web service description and a given specification, whether the description fulfills the specification. Such an approach is called *model checking* in literature since one aims at checking whether a description is a model of a specificitation and the algorithm that checks this is called *model checking algorithm*.

7.1. Introduction to Model Checking

Model checking has turned out to be one of the most useful techniques for automated reasoning about reactive systems began with the advent of the efficient temporal logic model checking [CE82, CES86, Eme81, QS82]. The basic idea is that the global state transition graph of a finite state reactive system defines a (Kripke) structure in the sense of temporal logic (cf. [Pnu77]), and we can give an efficient algorithm for checking if the state graph defines a model of a given specification expressed in an appropriate temporal logic. While earlier work in the protocol community had addressed the problem of analysis of simple reachability properties, model checking provided an expressive, uniform specfication language in the form of temporal logic along with a single, efficient verification algorithm which automatically handled a wide variety of correctness properties.

Model checking methods can be rougly categorized as follows:

Exiplicit State Representation vs. Symbolic State Representation In the explicit state approach the Kripke structure is represented extensionally using conventional data structures such as adjancency matrices and linked lists so that each state and transition is enumerated explicitly. In contast, in the symbolic approach boolean expressions denote large Kripke structures implicitly. Typically, the data structure involved is that of Binary Decision Diagrams (BDDs), which can, in many applications, although not always, manipulate boolean expressions denoting large sets of states efficiently.

The distinction between explicit state and symbolic representation is to a large extent an implementation issue rather than a conceptual one. The original model checking method was based on an algorithm for fixpoint computation and it was implemented using explicit state representation. The subsequent symbolic model checking method uses the same fixpoint computation algorithm, but now represents sets of states implicitly. However, the succinctness of BDD data structures underlying the implementation can make a significant practical difference.

Global Calculation vs. Local Search In the global approach, we are given a structure M and a formula ϕ. The algorithm calculates $\phi^M = \{s : M, s \models \phi\}$, that is the set of all states in M where ϕ is true. This necessarily entails examining the entire structure. Global algorithms typically proceed by induction on the formula structure, calculating ψ^M for the various subformula ψ of ϕ. The algorithm can be presented in recursive form; as the recursion "unwinds" the values of the shorter subformula are calculated first, then the next shortest and so on.

In contrast, in the local approach, we are given a specific state s_0 in M along with M and ϕ. We wish to determine whether $M, s_0 \models \phi$. The computation proceeds by performing a search of M starting at s_0. The potential advantage is that, many times in practice, only a portion of M may need to be examined to settle the question. In the worst case, however, it may still be necessary to examine all of M (cf. [SW91]).

While the local model-checking problem must determine modelhood of a single state, the global problem must decide modelhood of all the states in the structure. Obviously, solution of the global model-checking problem comprises solution of the local problem, and solving the local model-checking problem for each state in the structure solves the global model-checking problem. Thus, the two problems are closely related, but global and local model-checker have different applications.

For example, a classic application of model-checking is the verification of properties of models of hardware systems, where the hardware system contains many parallel components whose interactions is modeled by interleaving. The system's model structure grows exponentially with the number of parallel components, a problem known as *state-explosion problem.* A similar prolem arises when software systems, with variables ranging over a finite domain, are analyzed – the state space grows exponentially with the number of variables.

In such an application, local model-checking is usually preferred, because the property of interest is often expressed with respect to a specific initial state–a local model checker might inspect only a small part of the structure to decide the problem, and the part of the structure that is not inspected need not even be constructed. Thus, local model-checking is one means for fighting the state-explosion problem.

For other applications, like the use of model-checking for data-flow analysis, one is really interested in solving the global question, as the very purpose of the model-checking activity is to gain knowledge about all the states of a structure. Such applications use structures that are rather small in comparison to those arising in verification activities, and the state explosion problem hold less importance. Global methods are preferred in such situations.

Monolithic Structures vs. Incremental Algorithms To some extent this is also more of an implementation issue than a conceptual one. Again, however, it can have significant practical consequences. In the monolithic approach, the entire structure M is built and represented at one time in computer memory. While conceptually simple and consistend with standard conventions for judging the complexity of graph algorithms, in practice this may be highly undesirable because the entire graph of M may not fit in computer memory at once. In contrast, the incremental approach (also referred to as the "on-the-fly" or "online" approach) entails building and storing only small portions of the graph of M at any one time (cf. [JJ89]).

7.2. Matchmaking Algorithm

Now, we present a matchmaking algorithm for Web services that works on the semantically rich descriptions of Web services rather than just the types of input and output parameters. Furthermore, our matchmaking algorithm not only returns match/no-match answers but in case of a match a set of conditions under which a Web service offers the desired functionality.

Definition 21 (Matchmaking Problem)**.** A process P satisfies a formula ϕ under the conditions Ω if and only if the initial state s of the process P such that $s \models \phi$ under conditions Ω .

We adapt the approach of local model checking [SW91] in which for dertermining the truth of $s \models \phi$, one looks at the immediate neighbourhood of s as required by the formula. If there are fixpoints, one will of course move further and further away from s, but one can stop as soon as the truth or falsity of ϕ at s is established, rather than working with the whole state space. The reason for choosing this approach is also that in our case the states are process expressions and the operational semantics of the underlying process algebra gives the transition graph and we can save the effort of building the transition graph explicitly. The algorithm first decomposes a formula ϕ.

In outline, suppose we want to check $E \models \phi$. We start with a *sequent* $E \vdash \phi$, and then apply rules according to the structure of ϕ. For example, if $\phi = \phi_1 \wedge \phi_2$, the there is an $\wedge$ rule which is:

$$\frac{E \vdash \phi_1 \wedge \phi_2}{E \vdash \phi_1 \quad E \vdash \phi_2}$$

and if we get to a sequent $s \vdash \langle a \rangle \psi$, we apply the $\langle\rangle$-rule

$$\frac{E \vdash \langle a \rangle \psi}{F \vdash \psi} \quad E \xrightarrow{a} F$$

The transition $E \xrightarrow{a} F$ is derived from the transition rules for the process calculus. If we reach a fixpoint formula, or a variable Z which is bound by $\mu Z.\psi$ or $\nu Z.\psi$, we apply the fixpoint rules:

$$\frac{E \vdash \mu Z.\psi}{E \vdash Z} \quad \frac{E \vdash Z}{E \vdash \psi}$$

and

$$\frac{E \vdash \nu Z.\psi}{E \vdash Z} \quad \frac{E \vdash Z}{E \vdash \psi}$$

Then we build up (or rather down, since the rules grow downwards) a goal-directed tree of sequents to be estrablished. The question is, when do we stop. and how do we know we've succeeded. Obviously we stop on an a-modality if the process has no a-transitions, in which case $E \vdash [a]\psi$ succeeds and $E \vdash \langle a \rangle \psi$ fails; and we stop at $E \vdash P$, when we look E up in the proposition valuation to determine success. The key is the treatment of fixpoints. If we get to $E \vdash \phi$, and we have already seen the same sequent higher up in the tree, then we stop and examine the highest fixed point variable Z between these repeated sequents; the leaf sequent is successful if Z is bound by ν and unsuccessful if it is bound by μ. One we have a finite tree, we propogate success and failure back up to the root via the rules.

The model checker is a tableau system for testing whether or not a state E has the property expressed by a closed formula ϕ. As is common in tableau systems, the rules are inverse deduction typed rules. Here they are built from sequents of the form $E \vdash \phi$, proof-theoretic analogues of $E \in [\![\phi]\!]_{\mathcal{V}}$. The premise sequent $E \vdash \phi$ is the goal to be achieved while the consequents are the subgoals, which are determined by the structure of the model near E and the structure of ϕ. [SW91] presented the following tableau rules for the modal μ-calculus.

$$\frac{E \vdash \neg\neg\phi}{E \vdash \phi}$$

$$\frac{E \vdash \phi_1 \wedge \phi_2}{E \vdash \phi_1 \quad E \vdash \phi_2}$$

$$\frac{E \vdash \neg(\phi_1 \wedge \phi_2)}{E \vdash \neg\phi_1}$$

$$\frac{E \vdash \neg(\phi_1 \wedge \phi_2)}{E \vdash \neg\phi_2}$$

$$\frac{E \vdash [a]\phi}{E_1 \vdash \phi \ldots E_n \vdash \phi} \quad \{E_1, \ldots, E_n\} = \{E' | E \xrightarrow{a} E'\}$$

$$\frac{E \vdash \neg[a]\phi}{E' \vdash \neg\phi} \quad E \xrightarrow{a} E'$$

$$\frac{E \vdash \nu Z.\phi}{E \vdash U} \quad U = \nu Z.\phi$$

$$\frac{E \vdash \neg\nu Z.\phi}{E \vdash U} \quad U = \neg\nu Z.\phi$$

$$\frac{E \vdash U}{E \vdash \phi[Z := U]}$$

$$\frac{E \vdash \neg U}{E \vdash \neg\phi[Z := U]}$$

A tableau for $E \vdash \phi$ is a maximal proof tree whose root is labelled with the sequent $E \vdash \phi$. The sequents labelling the immediate successors of a node labelled $E \vdash \phi$ are determined by an application of one of the rules, dependent on the structure of ϕ. Maximaality means that no rule applies to a sequent labelling a leaf of a tableau.

A successful tableau for $E \vdash \phi_1$ is a finite tableau in which every leaf is labelled by a sequent $F \vdash \phi_2$ fulfilling one of the following requirements:

1. $\phi_2 = Q$ and $F \in \mathcal{V}(Q)$
2. $\phi_2 = \neg Q$ and $F \notin \mathcal{V}(Q)$
3. $\phi_2 = [a]\phi_3$
4. $\phi_2 = U$ and $\Delta(U) = \nu Z.\phi_3$

Tableau rules for the derived operators are just reformulations of some of the negation rules.

$$\frac{E \vdash \phi_1 \vee \phi_2}{E \vdash \phi_1}$$

$$\frac{E \vdash \phi_1 \vee \phi_2}{E \vdash \phi_2}$$

$$\frac{E \vdash \langle a \rangle \phi}{E' \vdash \phi} \quad E \xrightarrow{a} E'$$

$$\frac{E \vdash \mu Z.\phi}{E \vdash U} \quad U = \mu Z.\phi$$

Having the tableau rules for the propositional μ-calculus, we build our algorithm with the following idea. In the tableau rules above E is a state, which is equivalent to a process expression considering the semantics of the behavior (π-calculus) presented in Chapter 5. Roughly, the tableau rules break down a complex formula into atomic formulas. So the only thing that we need to do is to check when a process expression satisfies an atomic formula.

Recalling the formal model of an agent, we have for each agent $A \in \mathcal{A}$ a process $\mathcal{B}_A$ describing its behavior. In the following, let P denote the process representing the behavior of an agent and Ω_P the set of conditions intialized to $\emptyset$.

- If $P = \mathbf{0}$, do nothing, since this process cannot satisfy any formula except `false`.
- If $P = c(v_1, \ldots, v_k).Q$ with channel $c = (p, a, t)$ having the message type $t = p_1{:}R_1, \ldots, p_k{:}R_k$, we add in the knowledge base of agent A axioms $\mathsf{sameIndividual}(\mathsf{v_i}, \mathsf{c_{p_i}})$ for each $i \in \{1, \ldots, k\}$ to bind the variables $v_1, \ldots, v_k$ to message parts $p_1, \ldots, p_k$. Then we continue with the process Q.

  If $\phi = c(u_1{:}S_1, \ldots, u_n{:}S_n)$

  - if for each u_i there exists a v_j such that $S_i \sqsubseteq T_j$, then add P to the resulting set of states, where T_j is the current type of the variable v_j.
  - if there exists an input activity $c_l(v_1{:}T_1, \ldots, v_n{:}T_n)$ if for each u_i there exists a v_j such that $S_i \sqcap T_j \neq \emptyset$, then add P to the resulting set of states and add the condition $\Omega(e) \cup \{$"typeOf$(p_j(c)) \sqsubseteq S_i \sqcap T_j$"$\}$ (refer to Figure 7.1).

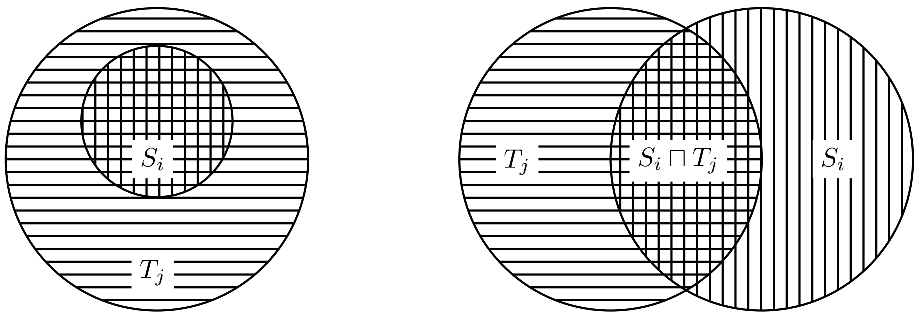


Figure 7.1.: Testing Input Formula

An input event $e = c_l(v_1{:}T_1, \ldots, v_n{:}T_n)$ models $c(u_1{:}S_1, \ldots, u_n{:}S_n)$

  - under conditions $\Omega(e)$ if for each u_i there exists a v_j such that $S_i \sqsubseteq T_j$.
  - under conditions $\Omega(e) \cup \{\text{"typeOf}(p_j(c)) \sqsubseteq S_i \sqcap T_j\text{"}\}$ if there exists an input activity $c_l(v_1{:}T_1, \ldots, v_n{:}T_n)$ if for each u_i theres exists a v_j such that $S_i \sqcap T_j \neq \emptyset$ (refer to Figure 7.1).

  Consider a Web service that requires an input and has the output type C. Further, depending on the value of an input parameter the Web service either returns an output of type C_1 or of type C_2, where $C_1 \sqsubseteq C$ and $C_2 \sqsubseteq C$. If a user requests for Web services that have output type C_1, this Web service should be detected as a match. Since, the output type of the Web service is C. This can be done easily by subsumption check. However, it is important to tell the user for which values of the input the service will offer the desired output, that is of type C_1.

- If $P = \overline{c}u_1, \ldots, u_k.Q$ with channel $c = (p, a, t)$ having the message type $t = p_1{:}R_1, \ldots, p_k{:}R_k$, we add in the knowledge base of agent A axioms $\mathsf{sameIndividual}(\mathsf{u_i}, \mathsf{c_{p_i}})$ for each $i \in \{1, \ldots, k\}$ to bind the message parts $p_1, \ldots, p_k$ to values $u_1, \ldots, u_k$. Then we continue with the process Q.

  If $\phi = \overline{c}(q_1, \ldots, q_n)$ then perform queries $q_1, \ldots, q_n$ on the local knowledge base of the current agent and add P to the resulting set of states if the set of answers for each query q_i is not empty.

  An output event $e = \overline{c}_l(v_1, \ldots, v_n)$ models $\overline{c}(q_1, \ldots, q_n)$ under conditions $\Omega(e)$ if for each q_i there exists a unique v_j such v_j belongs to answer of query q_i.

- If $P = l(x_1, \ldots, x_n).Q$ with effects (X, Y), we add each axiom in the set X in the knowledge base of A and remove each axiom in the set Y from the knowledge base of A appropriately instantiated with the parameters $x_1, \ldots, x_n$.

  If $\phi = q@A$ where q is a query and A is an agent, then pose the query on the local knowledge base of the agent A. Proposition $q@A$ holds in a state, if the answer of the query q performed by agent A on its local knowledge base is not empty. If this is the case, then add P to the resulting set of states.

- If $P = \tau.Q$, we continue with Q.

- If $P = P_1 \parallel P_2$, we continue with P_1 as well as P_2

- If $P = P_1 + P_2$, we continue with P_1 and P_2 alternatively.

- If $P = (\texttt{new}\ x)Q$, we create a new individual x in the knowledge base of A and continue with process Q.

- If $P = \omega?Q{:}R$, we add ω to $\Omega(Q)$ and continue with Q and add $\neg\omega$ to $\Omega(R)$ and continue with R.

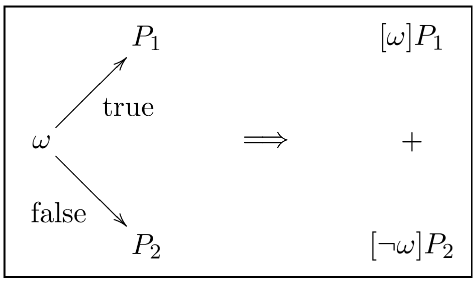


Figure 7.2.: Eliminating If-Then-Else

A Web service can take different execution paths at run time, depending on the values of the input parameters provided by the user, access control policies or any other semantic constraints of the web service process. So, we have to consider all possible runs of a web service process to check whether there is a run of the Web service process that satisfies the request.

- If $P = B(y_1, \ldots, y_n)$, we continue with the defining process of the agent identifier B.

Consider the example from Section 6.4. Suppose a user wishes to know whether the Web service eventually outputs an order confirmation for a book with the same description as entered by the client. From Section 6.3.2, we know that the modality "eventually ϕ" can be modeled as `true Until` ϕ. Now considering the query from the example in Section 6.4, our user can define his query as

$$\texttt{true Until } \langle\overline{c}(Q)\rangle\texttt{true@Adult}$$

where Q is defined as

$$Q = \text{select ?x where ?x rdf:type Confirmation. ?x hasBookDesc inBookDesc}$$

The formula `true Until` $\langle\overline{c}(Q)\rangle$`true@Adult` is equal to

$$\mu X.(G \vee (\texttt{true} \wedge \langle - \rangle\texttt{true} \wedge [-]X)),$$

where

$$G = \langle\overline{c}(Q)\rangle\texttt{true@Adult}$$

As we have shown in Section 6.4, the formula G holds in the last activity of the Web service, in which it outputs an order confirmation. Applying the tableau rule

$$\frac{E \vdash \mu Z.\phi}{E \vdash U} \quad U = \mu Z.\phi$$

to the our formula recursively, we obtain that the formula hold in start state of the Web service. Thus, the Web service form Section 6.4 satisfies the constraint that it eventually emits an order confirmation with the same book description as entered by the user.

7.2.1. Complexity

To discuss the complexity of our model-checking algorith, we need to introduce the notion of *alternation depth*. Intuitively, it means the maximum number of μ/ν alternations in a chain of nested fixpoints.

Theorem 1. The model-checking problem for a formula of size m and alternation depth d on a system of size n is $O(m \cdot n^{d+1})$, where d is the depth of fixpoint operators.

Proof. Every fixpoint closes at, at worst, the nth approximant. Calculating the boolean takes time $O(n)$; calculating modalities takes time $O(n)$. For an innermost fixpoint, given a valuation of the free variables, we may need to evaluate the body n times to close. So, the innermoset fixpoint takes time $O(n \cdot (mn))$. Now the next fixpoint out may also need to be evaluated n times, each of which involves evaluating the inner fixpoint (and some non-fixpoint operators); and so that takes $O(n \cdot n \cdot (mn))$. And so on: so we end up with $O(m \cdot n^{d+1})$. □

So model-checking in general is exponential in alternation depth. However, it is a very behaved exponential problem, both theoretically, as by stratifying the formulae according to alternation depth one gets a chain of polynomial problems of increasing degree; and practically, because systems are typically large, formulae are typically small, and alternation depths are typically 1 or 2.

7.3. Related Work

Recently, automatic matchmaking of Web services has gained tremendous importance and new approaches are introduced frequently. In the following, we will discuss some of the most widely known approaches.

7.3.1. OWL-S Matchmaker

OWL-S Matchmaker [SPAS03] matches a request described as OWL-S profile with the OWL-S profiles of the Web services. Let *Req* denote the OWL-S profile describing the request and *Ad* the OWL-S profile of a Web service. If in_{Ad} and in_{Req} represent the inputs of the advertisement and the request respectively, and out_{Ad} and out_{Req} represent their outputs, the OWL-S matchmaker recognizes an *exact output match* when $out_{Ad} = out_{Req}$ and an *exact input match* when $in_{Ad} = in_{Req}$. Also, the matchmaker recognizes a *plugIn match* when $out_{Ad} \sqsupset out_{Req}$, or $in_{Req} \sqsupset in_{Ad}$. When the outputs of the advertisement are more specific than the outputs of the request, then the advertised service provides less information than the requester needs. Still, it may be that the information provided by the advertiser is all that the requester needs, or that the requester may find another provider for the remaining data. In these cases, the matchmaker recognizes a *subsumed* match. Formally, the matchmaker recognizes a *subsumed match* when $out_{Req} \sqsupset out_{Ad}$ or $in_{Ad} \sqsupset in_{Req}$. When neither of the conditions above succeeds, there is not relation between the advertisement and the request and the match fails. OWL-S matchmaker defines a scoring function based on the degree of match detected. The scoring function is ordered as exact match > plugIn match > subsumed match > no match. Furthermore, in general the matchmaker prefers output matches over input matches. This is because the requester knows what he expects form the provider, but he cannot know what the provider needs until the provider is actually selected. Input matching is therefore relegated to a secondary role of tie breaker among the providers with equivalent outputs.

OWL-S matchmaking approach does not consider the relationship between inputs and outputs [PKPS02, SPAS03]. Because of this, the meaning of the service has to be described with a natural language text (refer to the text "*The service advertised is a car selling service which given a price reports which cars can be bought for that price*" in section 3.2 of [PKPS02]). OWL-S foresees to specify pre- and postconditions as SWRL rules to model relationships between inputs and outputs. However, to the best of our knowledge there is no matchmaking approach that uses the SWRL rules. Considering only the types of input and output parameter is the major shortcoming of OWL-S matchmaker. Consider two Web services, one adds two numbers and the other subtracts two numbers. That is, both the services have same input types and same output type. OWL-S matchmaker will detect both the services as a match to a request asking for a Web service that can add two numbers or a Web service that can multiply two numbers. Further, note that matchmaking algorithms that consider only the types of input and output parameters do not actually require OWL-S profile, since the required information is available is WSDL documents.

The basic difference between OWL-S Matchmaker [SPAS03, PKPS02] and our approach is that the former matches interfaces whereas the latter the functionality of Web services. The idea behind the work done in [APS05] is closer to ours. However, there are still some major differences. The approach developed in [APS05] considers only the dynamic behaviour of processes whereas our approach considers ontology based resource descriptions as well as access control and trust policies. Note, that verifying only the dynamic behaviour of processes has been already thoroughly investigated in past decades. Further the approach presented in [APS05], since based on SPIN model checker, uses LTL, whereas our formalism is based on μ-calculus which is more expressive than LTL.

7.3.2. WSMO Discovery

WSMO Web service discovery approach differentiates between service and Web service discovery [KLP+04]. In WSMO Web service discovery Web services as well as goals are represented as *sets of objects*. The objects described in some Web service description and the objects used in some goal description can or might be interrelated in some way by ontologies. In the general case, this interrelation has to explicated by a suitable mediator that resolves heterogenities amongst the terms used in goal and web service descriptions. In order to consider a goal G and a Web service W to match on a semantic level, the sets R_G and R_W describing their elements are interrelated as follows:

- $R_G = R_W$. Here the objects that are advertised by the service provider (and which thus can potentially be delivered by the described web service W) and the set of relevant objects for the requester as specified in the goal G perfectly match, i.e. they coincide. In other words, the service might be able to deliver all relevant objects. In any case, it is guaranteed that no irrelevant objects will be delivered by the service.

- $R_G \subseteq R_W$. Here the relevant objects that are advertised by the service provider form a superset of the set of relevant objects for the requester as specified in the goal G. In other words, the service might be able to deliver all relevant objects. Moreover, there is no guarantee that no irrelevant objects will be delivered by the service, i.e. it is possible that the service delivers objects that are irrelevant for the client as well.

- $R_W \subseteq R_G$. Here the relevant objects that are advertised by the service provider only form a subset of the set of relevant objects for the requester as specified in the goal G. In other words, the service in general is not able to deliver all objects that are relevant objects for the requester. But, there is a guarantee that no irrelevant objects will be delivered by the service.

- $R_G \cap R_W \neq \emptyset$. Here there the set of relevant objects that are advertised by the service provider and the set of relevant objects for the requester have a non- empty intersection, i.e. there is an object (in the common) universe which is declared as relevant by both parties. In other words, the service in general is not able to deliver all objects that are relevant objects for the requester, but at least one such element can be delivered. Moreover, there is no guarantee that no irrelevant objects will be delivered by the service.

- $R_G \cap R_W = \emptyset$. Here, the objects the web service description refers to and the objects the requester goal refers to are disjoint. That means there is no semantic link between both descriptions; they are talking about different things.

As we have mentioned in the previous chapter, WSMO goals are more expressive than OWL-S Profile and less expressive than the formalism that we developed in the previous chapter. WSML captures the relationships between various resources with the help of rules, whereas OWL-S has not yet showed how the pre- and postconditions can be modeled concretely. However, WSML lacks a syntax and semantics for specifying constraints on the dynamic behaviour of Web services. Furthermore, to the best of our knowledge there exist no algorithms for matching goals and Web services described in WSML.

7.3.3. Other Matchmaking Approaches

In [LH03], service advertisements and goals are specified as DL concepts and the approach has similar drawbacks as the goal specification technique in OWL-S Matchmaker.

In [GMP04], it is argued that subsumption based matchmaking can be too strong. That is, it may not find some matches. The authors suggest *entailment of concept non-disjointness* instead of subsumption to overcome this shortcoming. Section 4.3 of [GMP04] says "The requester accepts shipping from Plymouth or Dublin and the provider accepts shipping from UK cities. Since Dublin is not in the UK, neither of the two associated sets of service instances fully contains the other in every possible world.". That is, matching based on subsumption will not detect the service as a match, but matching based on entailment of concept non-disjointness will. However, only a positive boolean answer may make a requester believe that the service also accepts shipping from Dublin.

Major difference between our approach and all other currently existing approaches is that the existing approaches produce only yes/no answers. That is, they either do not consider the conditions while performing matchmaking or do not provide the client with conditions under which a Web service offers the desired functionality. Another difference between our approach and existing approaches is that we consider security constraints in the matchmaking. Furthermore, our request specification technique is much more expressive than those used in the existing approaches that allow only the type of output as request and hence a user can filter Web services in a much richer way than only the by the types of their inputs and outputs.

7.4. Conclusion

In this chapter, we presented a matchmaking approach based on model checking. Unlike most of the existing approaches that may provide expressive formalism for describing Web services but do not provide matchmaking algorithms that exploits the expressiveness of the formalism, our matchmaking approach can use all the available information in the semantic description of Web services. To the best of our knowledge ours is the only work that has considered the matchmaking problem as a satisfiability problem and could thus gain from the insights gained from many years of theortical work done in the process algebra community.

Our matchmaking approach matches functionalities of Web services rather only their interface with user request. That is, our technique allows users to define constraints on the Web service functionalities and then search for Web services that fulfill the constraints. Note that matching Web service interfaces only ensures that a user can execute a Web service found as a match. However, when a user needs a Web service, his main concern is to find Web services that offer the required functionality and not Web services that he can invoke in principle. Furthermore, our goal specfication formalism developed in the previous chapter covers the interface matching due to the availability of constraints on the type and values of input and output parameters.

8. Simulation based Matchmaking

In the previous two chapters we have seen how goals can be specified and how Web services that satisfy a goal can be found. Such a technique can be used to specify certain steps of a business process as goals rather than hard-wiring a concrete Web service. When the business process is executed, Web services satisfying the goals can be found and integrated automatically. In some situations Web services are already integrated in a business process. In such scenarios, it is often desired to simply replace one Web service by another Web service that offers the same functionality. That is given a Web service W_{old} that needs to be replaced, one seeks another Web service W_{new} such that for all formulas ϕ if the old Web service W_{old} satisfies the formula ϕ, W_{new} must also satisfy the formula ϕ. However, often the specifications of the constraints that W_{old} satisfies may not be available. In order to use the model checking based matchmaking developed in the previous chapter, one needs to derive from the description of W_{old} all the formulas, say $\phi_1, \ldots, \phi_n$, that W_{old} satisfies and then check for each Web service W whether W satisfies each ϕ_i. Since n, the number of formulas that W_{old} satisfies, can be very large, finding an equivalent new Web service by model checking is not efficient. Rather, there is a need for an alternative approach, that can directly check whether a Web service can offer the same functionality as another Web service. In this chapter, we develop a technique for determining simulation of agents by comparing them directly.

For comparing two agents directly, we need to consider all the three aspects of the agent, namely the credentials of the agents, the resources of the agents and the behavior of the agents. Otherwise, we can not ensure that the two agents satisfy same set of formulas.

8.1. Simulation of Resources

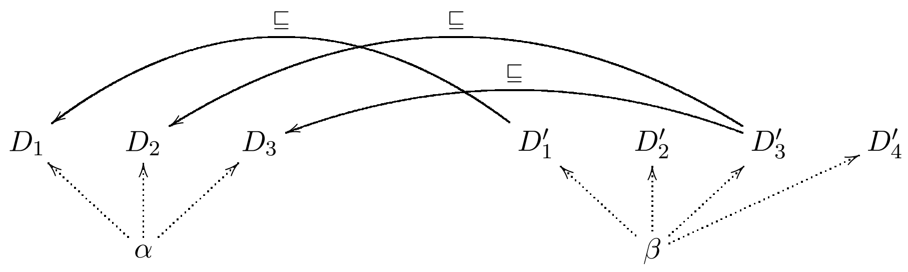
In case of resources, we differentiate between actual resources and resource types, where the latter denotes a set of resources. In Chapter 5, we have modeled resource types as description logic concepts. Thus, simulation of resource types can be directly obtained by subsumption of the corresponding concepts.

Description logics however do not provide any reasoning procedure to directly find out whether a resource can be replaced by another. Note, that the $sameIndividual(x, y)$ axiom provided by OWL has the semantics that x and y describe the same resource. To determine whether a resource x can be replaced by another resource y, we need to ensure that for any given DL query q, if x is an answer to query q, y is also the answer of query q. Obviously if the individuals x and y are equal either both or none of them appears in the answer of any DL query. However, equality is too strong for the notion of simulation that we seek.

Consider for example two book selling Web services A and B. Both of them require the ISBN of the desired book and delivery address as input and send the ordered book to the delivery address. Now, if A and B are provided by different organizations, say amazon.de and bol.de, they have different warehouses, that is disjoint sets of resources. So, they will actually deliver different copies, say α and β, of the ordered book and not the same copy. However, both the copies α and β, even if being two different real world resources, have same properties, i.e. same ISBN, same title etc.. The *sameIndividual*(α, β) axiom means that α and β describe the same resource.

Definition 22 (Simulation of Resources)**.** We define simulation of resources α and β as follows:

- If $\alpha = \beta$ then a is simulated by b and b is simulated by a
- If $\alpha \neq \beta$, α is simulated by β, if the following two conditions hold



$$\begin{tikzcd}
D_1 & D_2 & D_3 & D'_1 \arrow[lll, bend right, "\sqsubseteq"'] & D'_2 & D'_3 \arrow[llll, bend right, "\sqsubseteq"'] \arrow[lll, bend right, "\sqsubseteq"'] & D'_4 \\
 & \alpha \arrow[ul, dotted] \arrow[u, dotted] \arrow[ur, dotted] & & & \beta \arrow[ul, dotted] \arrow[u, dotted] \arrow[ur, dotted] \arrow[urr, dotted] & &
\end{tikzcd}$$

Figure 8.1.: Resource Simulation Example: Individual α is member of the classes D_1,D_2 and D_3. Individual β is member of the classes D'_1,D'_2, D'_3 and D'_4. Furthermore, $D'_1 \sqsubseteq D_1$, $D'_3 \sqsubseteq D_2$ and $D'_3 \sqsubseteq D_3$ hold.

1. for every decription D that α is member of there must exist a description D' that β is member of such that $D' \sqsubseteq D$ (refer to Figure 8.1).
2. for every property instance p of a property P that α has with an individual γ, there must exist a unique property instance p' of the property $P' \sqsubseteq P$ that β has with an individual γ' such that γ is simulated by γ' (refer to Figure 8.2).

Now, we present the algorithm to determine whether a resource α can be simulated by another resource β. The idea is to build a binary relation S as defined in the definition above and then check whether the pair (α, β) belongs to S or not. We initialize the relation S as presented in the Algorithm 1.

Algorithm 1 Initialize

```
calculate the set of all individuals reachable directly or indirectly by $\alpha$ and denote it by $A$.
Add $\alpha$ to $A$.
calculate the set of all individuals reachable directly or indirectly by $\beta$ and denote it by $B$.
Add $\beta$ to $B$.
for all pairs $(a, b)$ such that $a \in A$ and $b \in B$ do
  if $a = b$ then
    add $(a, b)$ to $S$.
  end if
end for
```

Algorithm 2 Constructing the Simulation Relation between Resources

```
while $(\alpha, \beta) \notin S$ do
  set $changed = false$
  for all $(a, b) \in A \times B$ do
    if $(a, b) \notin S$ then
      if for every description $D$ that $a$ is a member of there exists a description $D' \sqsubseteq D$ that $b$ is member of then
        if for every property instance $p$ of property $P$ that $a$ has with an individual $c$, there exists a unique property instance $p'$ of property $P' \sqsubseteq P$ with an individual $c'$ such that $(c, c') \in S$ then
          add $(a, b)$ to $S$
          set $changed = true$
        end if
      end if
    end if
  end for
  if $changed = false$ then
    return false
  end if
end while
return true
```

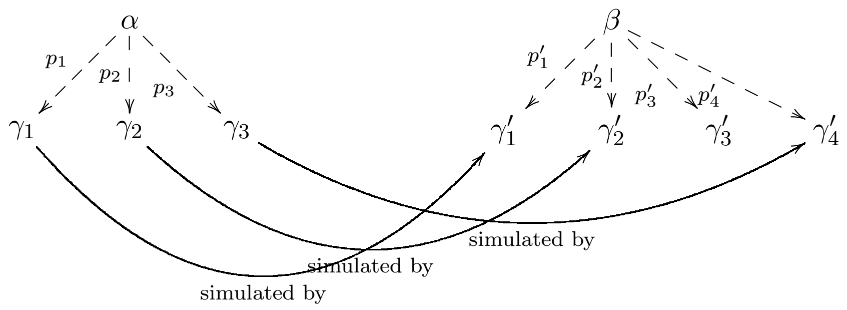


Figure 8.2.: Resource Simulation Example: Property instances p_1, p_2, p_3 are instances of the a property P, whereas property instances p'_1, p'_2, p'_3, p'_4 are instances of property P', with $P' \sqsubseteq P$

Having initialized the relation, we build the set as presented in the Algorithm 2. The outermost **while** loop of the Algorithm 2 terminates if $(\alpha, \beta) \in S$, hence it returns true. Otherwise, it keeps on checking whether an individual $a \in A$ can be simulated by an individual $b \in B$. If such a pair is found it is added in the set S and the fact that the set S has been changed is recorded by setting a boolean variable *changed* to *true*. After all the pairs $(a, b) \in A \times B$ have been processed, a new round is started if the set S was changed. Otherwise, that is if $changed = false$, we have arrived the saturation since the set S is not changing and there is no need of starting another round. In this case, the algorithm terminates by returning false, which means that the resource α can not be simulated by resource β.

The complexity of the algorithm is easy to determine. Let $|A \times B|$ denote the number of elements in the set $A \times B$. Every round, except perhaps the last round, adds at least one $(a, b) \in A \times B$ to S. So, there can be atmost $|A \times B|$ rounds before a round takes place that does not change the set S, which leads to termination. In every round $|A \times B|$ elements are processed. So, the complexity of the algorithm is $O(|A \times B|^2)$.

8.2. Comparing Credentials of Agents

As described in Chapter 5, an agent α has a set of certificate $\mathcal{C}_\alpha$ and agents trust other agents on basis of their credentials. As we have seen in Chapter 6, users can restrict the desired set of agents that may be involved in a Web service process.

If an agent α is supposed to be replaced by another agent β, a user must be able to trust agent β at least as much as he trusts agent α. Since, we do not know the user and his trust policy, agent β should fulfill every trust policy that agent α also fulfills in order to be a replacement candidate for agent α.

In Chapter 5 and 6, we have shown how certifiable properties can be modeled as concept expressions. The fact that an agent α possesses a property P is modeled by a description logic class membership axiom $P(\alpha)$. Furthermore, trust as well as access control policies are modeled as description logic queries such that checking whether an agent can be trusted or granted access is reduced to checking whether the agent is in the answer of the corresponding query.

In order to ensure that agent β is trusted or granted access, whenever α is trusted or granted access, agent β must also have at least those properties that agent α has. In other words, for every property P, if the axiom $P(\alpha)$ exists then there must an axiom $P'(\beta)$ with $P' \sqsubseteq P$.

Consider for example, properties $\mathsf{above18}$ and $\mathsf{above25}$ and an obvious axiom

$$\mathsf{above25} \sqsubseteq \mathsf{above18},$$

which means that an agent who is above 25 years is also above 18 years. Suppose, an agent α possesses the property $\mathsf{above18}$, that is there exists an axiom $\mathsf{above18}(\alpha)$ and agent β possesses the property $\mathsf{above25}$, that is there exists an axiom $\mathsf{above25}(\beta)$. Now, if the only criteria for trusting or granting access to an agent is that the agent must be above 18 years of age, then agent β can always be trusted or granted access whenever agent α is trusted or granted acesss.

Coming back to general case, suppose agent α has properties $P_1, \ldots, P_n$. That is there exists axioms $P_1(\alpha), \ldots P_n(\alpha)$. Similarly, supppose agent β has properties $Q_1, \ldots, Q_m$. That is there exists axioms $Q_1(\beta), \ldots Q_m(\beta)$. We require that agent β must also have all the properties that agent α has. That is, for every $i \in \{1, \ldots, n\}$ there must be a $j \in \{1, \ldots, m\}$ such that $Q_j \sqsubseteq P_i$.

8.3. Comparing Behavior of Agents

Now we turn our attention to the last aspect that is simulation of agent behaviors.

8.3.1. Trace Equivalence: A First Attempt

Let us say that a *trace* of a process P is a sequence $\alpha_1, \ldots, \alpha_k$ such that there exists a sequence of transitions

$$P = P_0 \xrightarrow{\alpha_1} P_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_k} P_k,$$

for some $P_1, \ldots P_k$. Let us denote with $\Gamma(P)$ the collection of traces of P. Since ΓP describes all the possible finite sequences of interactions that we may have with process P, it is reasonable to require that our notion of behavioral equivalence only relates processes that afford the same traces, or else we should have a very good reason for telling them apart – namely a sequence of communications that can be performed with one, but not with the other. This means that, for all processes P and Q, we require that if P and Q are behaviorally equivalent, then $\Gamma(P) = \Gamma(Q)$.

This point of view is totally justified and natural if we view our labelled transition systems as non-deterministic devices that may generate or accept sequences of actions. However, it turned out that trace equivalence is not particularly reasonable if we view our automata as reactive machines that interact with their environment.

To understand this, consider two coffee machines CTM and CTM' defined as follows:

$$\begin{aligned} CTM &\stackrel{\text{def}}{=} coin.(\overline{coffee}.CTM + \overline{tea}.CTM) \\ CTM' &\stackrel{\text{def}}{=} coin.\overline{coffee}.CTM' + coin.\overline{tea}.CTM' \end{aligned}$$

It is easy to see that CTM and CTM' afford the same traces. However if a user, a coffee addict (CA) wants coffee and not tea, he will prefer to interact with CTM and not CTM'. The reason is that the machine CTM will give him coffee after receiving a coin, whereas CTM' may refuse to deliver coffee after having accepted his coin. The reason why the machines behave differently from the point of view of the user become clearer if we formalize the user, a coffe addict as follows:

$$CA \stackrel{\text{def}}{=} \overline{coin}.coffee.CA$$

and the processes of his interaction with the coffee machines as

$$CA \parallel CTM$$

and

$$CA \parallel CTM'$$

In the first case, after the coin has been inserted, the user is willing to accept coffee and the machine CTM can provide coffee or tea. However, since the user is willing to accept only coffee, the interaction can take place only if coffee is delivered. In the second case, however the machines decides already before the user inserts the coin, whether it would deliver coffee or tea after the user has inserted the coin. In case, it decides to deliver tea, it will lead to a deadlock since the user is willing to accept only coffee whereas the machine is only willing to deliver tea, as a result the interaction can not take place.

Therefore we need to refine our notion of equivalence in order to differentiate processes that, like the two vending machines above, exhibit different reactive behaviour while still having the same traces.

8.3.2. Strong Bisimilarity

The problem with trace equivalence is that when looking purely at the (completed) traces of a process, we focus only on the sequence of actions that the process may perform, but do not take into account the communication capabilities of the intermediate states that the process traverses as it computes. As the above example shows, the communication potential of the intermediate states *does matter* when we may interact with the process all the times. In particular, there is a crucial difference in the capabilites of the states reached by CTM and CTM' after inputting a coin. Indeed, after accepting a coin the machine CTM always enters a state in which it is willing to output either coffee or tea, depending on what its user wants, whereas the machine CTM' can only enter a state in which it is willing to deliver either coffee or tea, but not both.

The insight that we gain from the above example is that a suitable notion of behavioral relation between reactive systems should allow us to distinguish processes that may have different deadlock potential when made to interact with other processes. Such a notion of behavioral relation must take into account the communication capabilities of the intermediate states that processes my reach as they compute. One way to ensure that this holds is to require that in order for two processes to be equivalent, not only they should afford the same traces, but, in some formal sense, the states that they reach should still be equivalent.

$$\begin{tikzcd}
P \arrow[r, "\alpha"] \arrow[d, dash, "\leq"'] & P' \arrow[d, dash, "\leq"'] \\
Q \arrow[r, "\alpha"] & Q'
\end{tikzcd}$$

Figure 8.3.: Agent P is simulated by agent Q

A binary relation $\leq$ on agents is a simulation if $P \leq Q$ implies that any transition from P to P' must be simulated by a transition in Q to Q' such that the derivatives P' and Q' remain in the simulation, that is $P' \leq Q'$ holds (refer to Figure 8.3).

Definition 23. A binary relation $\mathcal{S}$ on agents is a *(strong) simulation* if $P\mathcal{S}Q$ implies

1. If $P \xrightarrow{\alpha} P'$ and α is free action, then for some Q', $Q \xrightarrow{\alpha} Q'$ and $P'\mathcal{S}Q'$
2. If $P \xrightarrow{x(y)} P'$ and $y \notin n(P,Q)$, then for some Q', $Q \xrightarrow{x(y)} Q'$ and for all w, $P'\{w/y\}\mathcal{S}Q'\{w/y\}$
3. If $P \xrightarrow{\overline{x}(y)} P'$ and $y \notin n(P,Q)$, then for some Q', $Q \xrightarrow{\overline{x}(y)} Q'$ and $P'\mathcal{S}Q'$

Definition 24. The relation $\mathcal{S}$ is a *(strong) bisimulation* if both $\mathcal{S}$ and its inverse are simulations. The relation $\dot{\sim}$, *(strong) bisimilarity*, on agents is defined by $P\dot{\sim}Q$ if and only if there exists a bisimulation $\mathcal{S}$ such that $P\mathcal{S}Q$.

8.3.3. Weak Bisimilarity

Consider, for instance, the processes $a.\tau.\mathbf{0}$ and $a.\mathbf{0}$. Since τ actions should be unobservable, we intuitively expect these to be observationally equivalent. Unfortunately, however, the process $a.\tau.\mathbf{0}$ and $a.\mathbf{0}$ are not strongly bisimlar. In fact, the definition of strong bisimulation requires that each transition in the behaviour of one process should be matched one transition of the other, regardless of whether that transition is labelled by an observable action or τ, and $a.\tau.\mathbf{0}$ affords the trace $a\tau$, whereas $a.\mathbf{0}$ does not.

In hindsight, this failure of strong bisimlarity to account for the unobservable nature of τ actions is expected because the definition of strong bisimulation treats internal actions as if they were ordinary observable actions. What we should like to have is a notion of bisimulation equivalence that affords all of the good properties of strong bisimilarity, and abstracts from τ actions in the behaviour of processes. However, in order to fulfill the aim, first we need to understand what "abstracting from τ actions" actually means. Does this simply mean that we can "erase" all of the τ actions in the behaviour of a process? This would be enough to show that $a.\tau.\mathbf{0}$ and $a.\mathbf{0}$ are equivalent, as the former process is identical to the latter if we "erase the τ prefix".

$$\boxed{\begin{tikzcd}
& \text{Start} \arrow[d, "\overline{pub}"'] & \\
& CM \parallel CS \arrow[dl, "\tau"'] \arrow[dr, "\tau"] & \\
\text{Good} \arrow[d, "\tau"'] & & \text{Bad} \\
\text{Start} & &
\end{tikzcd}}$$

$$\begin{aligned}
\text{Start} &\stackrel{\text{def}}{=} CM \parallel CS \\
\text{Good} &\stackrel{\text{def}}{=} \overline{coffee}.CM \parallel CS'' \\
\text{Bad} &\stackrel{\text{def}}{=} CM \parallel CS'' \\
CS &\stackrel{\text{def}}{=} \overline{pub}.CS' \\
CS' &\stackrel{\text{def}}{=} \overline{coin}.CS'' \\
CS'' &\stackrel{\text{def}}{=} coffee.CS
\end{aligned}$$

Figure 8.4.: The behaviour of $CM \parallel CS$

To understand this issue better let us consider a variation of the coffee machine

$$CM \stackrel{\text{def}}{=} coin.\overline{coffee}.CM + coin.CM$$

Note that, upon receipt of a coin, the coffee machine CM can decide to go back to its initial state without delivering the coffee. Figure 8.4 describes the possible behaviours of the sytem $CM \parallel CS$. Note, that the two possible τ-transitions that stem form the process $CM \parallel CS'$, and that one of them, namely

$$CM \parallel CS' \xrightarrow{\tau} CM \parallel CS'',$$

leads to a deadlocked state. Although directly unobservable, this transition cannot be ignored in our analysis of the behaviour of this system because it pre-empts the other possible behaviour of the machine. So, unobservable actions cannot be just erased from the behavour of processes because, in light of their pre-emptive power in the presense of non-deterministic choices, they may affect what we may observe.

In order to define a notion of bisimulation that allows us to abstract from internal transitions in process behaviours, a new notion of transition relation between processes is introduced.

Definition 25. Let P and Q be processes. We write $P \overset{\epsilon}{\Rightarrow} Q$ iff there is a (possibly empty) sequence of τ-labelled transitions that lead from P to Q. (if the sequence is empty, then $P = Q$.)

For each action α, we write $P \overset{\alpha}{\Rightarrow} Q$ iff there are processes P' and Q' such that

$$P \overset{\epsilon}{\Rightarrow} P' \xrightarrow{\alpha} Q' \overset{\epsilon}{\Rightarrow} Q.$$

For each action α, we use $hat\alpha$ to stand for ϵ if $\alpha = \tau$, and for α otherwise. Thus $P \overset{\alpha}{\Rightarrow} Q$ holds if P can reach Q by performing an α-labelled transition, possibly preceded and followed by sequences of τ-transitions. For example, $a.\tau.\mathbf{0} \overset{a}{\Rightarrow} \mathbf{0}$ and $a.\tau.\mathbf{0} \overset{a}{\Rightarrow} \tau.\mathbf{0}$ both hold.

In the LTS depicted in Figure 8.4, apart from the obvious one step $\overline{pub}$-labelled transition, we have that

$$\begin{array}{lcl} \text{Start} & \overset{\overline{\text{pub}}}{\Rightarrow} & Good \\ \text{Start} & \overset{\overline{\text{pub}}}{\Rightarrow} & Bad \\ \text{Start} & \overset{\overline{\text{pub}}}{\Rightarrow} & Start \end{array}$$

Now we use the new transition relations presented above to define a notion of bisimulation that can be used to equate processes that offer the same observable behaviour despite possibly having every different amount of internal computations. The idea underlying the definition of the new notion of bisimulation is that a transition of a process can now be matched by a sequence of transitions from the other that has the same "oberservational content" and leads to a state that is similar to that reached by the first process.

Definition 26 (Weak Bisimulation and Observational Equivalence)**.** A binary relation $\mathcal{R}$ over the set of states of an LTS is a weak bisimulation iff whenever $s_1 \mathcal{R} s_2$ and α is an action:

- if $s_1 \xrightarrow{\alpha} s_1'$ then there is a transition $s_2 \overset{\hat{\alpha}}{\Rightarrow} s_2'$ such that $s_1' \mathcal{R} s_2'$
- if $s_2 \xrightarrow{\alpha} s_2'$ then there is a transition $s_1 \overset{\hat{\alpha}}{\Rightarrow} s_1'$ such that $s_1' \mathcal{R} s_2'$

Two states s and s' are observationally equivalent (or weakly bisimlar), written $s \approx s'$, iff there is a weak bisimulation that related them. Henceforth the relation $\approx$ will be referred to as observational equivalence or weak bisimilarity.

8.4. Algorithm

For two Web service descriptions W_1 and W_2, we wish to determine, whether they are congruent. That is, whether one can be replaced by another in a larger system without changing the functionality of the larger system.

α-convertibility is a well known technique used in process algebras to determine two process expressions P and Q can become syntactically equal by renaming their variables without changing their semantics [MPW92].

Consider $u = v$. That is, u is bound and v is free. Renaming v to u will lead to $u = u$, which is a problem, since it changes the semantics of the process. The solution is to first rename u to a new name, say u_1 and then rename v to u. That is, changing of a bound name is a recursive process.

Definition 27 (Sub Process)**.** The set of direct sub processes $DSP(P)$ of a process P is defined as follows (refer to Section 5.1.2 for the syntax of valid process expressions):

- $P = \mathbf{0}$ implies $DSP(P) = \emptyset$.
- $P = \tau.Q$ implies $DSP(P) = \{Q\}$.
- $P = y(x).Q$ implies $DSP(P) = \{Q\}$.
- $P = \overline{y}x.Q$ implies $DSP(P) = \{Q\}$.
- $P = Q \parallel R$ implies $DSP(P) = \{Q, R\}$.
- $P = Q + R$ implies $DSP(P) = \{Q, R\}$.
- $P = \omega?Q{:}R$ implies $DSP(P) = \{Q, R\}$.

- $P = (\texttt{new}\ x)Q$ implies $DSP(P) = \{Q\}$.
- $P = A(y_1, \ldots, y_n)$ implies $DSP(P) = \{Q\}$ if $A(x_1, \ldots, x_n) \overset{\text{def}}{=} Q$.

The set of all sub processes of a process P is defined as the union of the set of all direct sub processes of P and sets of all sub processes of the all direct sub processes of P

Now, we present the algorithm for checking the equivalence of the Web services W_1 and W_2.

1. Let B_1 and B_2 denote the processes describing the dynamic behaviours of the Web services W_1 and W_2 respectively.
2. We first construct the sets of all the sub processes of P and Q and denote the set by $\mathcal{S}$ (refer to Definition 27).
3. Intitialize a set $\mathcal{R}$ to $\emptyset$. The set $\mathcal{R}$ will contain the pairs of processes that are equivalent. That is, $(P, Q) \in \mathcal{R}$ it means that $P \equiv Q$ (refer to Definitions 13 and 14).

After the above initialization steps, we construct the set $\mathcal{R}$ by the following rules for each pair (P, Q), with $P, Q \in \mathcal{S}$. In the following, whenever we deduce $P \approx Q$ it means that we insert the (P, Q) in the set $\mathcal{R}$.

- **Syntactic Equivalence** $P = Q \Rightarrow P \approx Q$
- **Congruence** $P \approx Q \Rightarrow$
  - $\tau.P \approx \tau.Q$
  - $\overline{x}y.P \approx \overline{x}y.Q$
  - $P + R \approx Q + R$
  - $P \parallel R \approx Q \parallel R$
  - $(x)P \approx (x)Q$
  - $\omega?P{:}R \approx \omega?Q{:}R$
  - $\omega?R{:}P \approx \omega?R{:}Q$
  - $P\{z/y\} = Q\{z/y\}$, for all names $z \in fn(P, Q, y), \Rightarrow x(y).P \approx x(y).Q$
- **Summation**
  - $P + \mathbf{0} \approx P$
  - $P + P \approx P$
  - $P + Q \approx Q + P$

  - $P + (Q + R) \approx (P + Q) + R$

- **Restriction**
  - $(\texttt{new}\, x)P \approx P$, if $x \notin fn(P)$
  - $(\texttt{new}\, x)(\texttt{new}\, y)P \approx (\texttt{new}\, y)(\texttt{new}\, x)P$
  - $(\texttt{new}\, x)P + Q \approx (\texttt{new}\, x)P + (\texttt{new}\, x)Q$
  - $(\texttt{new}\ x)\alpha.P \approx \alpha.(\texttt{new}\ x)P$ for α an input, output or local action and if $x \notin n(\alpha)$
  - $(\texttt{new}\, x)\alpha.P \approx \mathbf{0}$ for α an input, output or local action and if x is the subject of α

- **IfThenElse**
  - $\omega?P{:}Q \approx P$ if $\omega = true$
  - $\omega?P{:}Q \approx Q$ if $\omega = false$
  - $\omega_1?P{:}Q \approx \omega_2?P{:}Q$ if $\omega_1 \sqsubseteq \omega_2$ and $\omega_2 \sqsubseteq \omega_1$

- **Communication**
  - $\overline{x}y.P \approx \overline{x}z.P$ if resource y simulates resource z.
  - $x(z).P \approx y(z).P$, if the message type of channel x simulates the message type of channel y.

- **Identifier** $A(x_1, \ldots, x_n) \overset{\text{def}}{=} P \Rightarrow A(y_1, \ldots, y_n) \approx P\{y_1/x_1, \ldots, y_n/x_n\}$

After the saturation is complete, we check whether the pair (B_1, B_2) is in the set $\mathcal{R}$. If $(B_1, B_2) \in \mathcal{R}$, then they are equivalent and we stop with a positive answer. If $(B_1, B_2) \notin \mathcal{R}$, then they are not equivalent and we stop with a negative answer.

Consider the Web services w_1 and w_2 defined as follows:

$$\begin{array}{rcl} w_1 & = & a(x_1, x_2, x_3).\overline{b}x_1.\mathbf{0} \\ w_2 & = & a(y_1, y_2, y_3).\overline{b}y_1.\mathbf{0} \end{array}$$

From the rule for syntactic equivalence (α conversion), it can be found out, that w_1 and w_2 are identical except the choice of the names of the variables. That is, if the variable y_1 is renamed by x_1 in the definition of w_2, w_2 becomes syntactially identical to w_1.

Now consider two more Web services w_3 and w_4 defined as follows:

$$\begin{array}{rcl} w_3 & = & a(x).Person(x)?w_1{:}\mathbf{0} \\ w_4 & = & a(x).Person(x)?w_2{:}\mathbf{0} \end{array}$$

Consider the rule $\omega?P{:}R \approx \omega?Q{:}R$, given $P \approx R$. Since we know that w_1 and w_2 are α convertible, we can replace w_2 by w_1 in the definition of w_4 as w_3 and w_4 check exactly the same condition $Person(x)$. That is, we would identify w_3 and w_4 as equivalent Web services.

8.5. Conclusion

In this chapter, we have developed an algorithm for directly comparing two Web service descriptions. We developed methods for simulation of resources and showed that the simulation of credentials can be done with subsumption of concepts that is directly provided by description logics. Finally, we presented a complete algorithm by combining the weak bisimulation of behavior known from process algebras with bisimulation of resources. The algorithm checks whether a Web service can simulate the other Web service. Such a technique for directly comparing two Web services is useful in business scenarios, where a business process already uses Web services for certain tasks and a Web service it is desired to replace a web service by another that offers the same functionality. To the best of our knowledge existing Web service matchmaking approaches are goal driven and ours is the only work that motivates the need for a simulation based matchmaking and provides a procedure for checking whether a Web service can be replaced by another by comparing them directly.

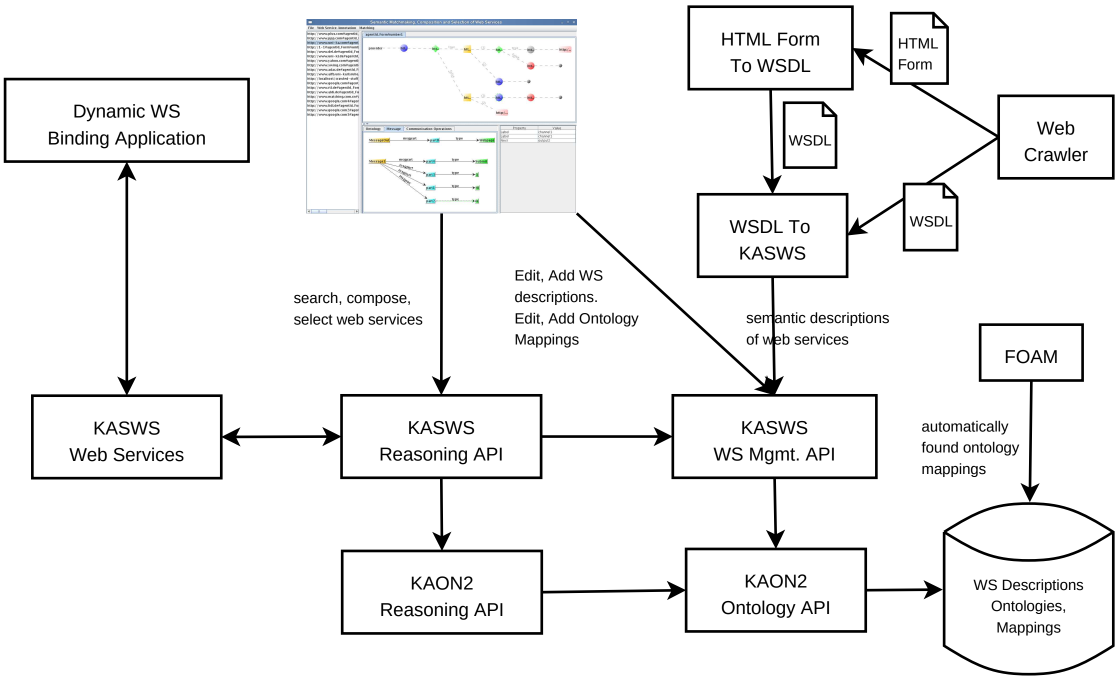
Part IV.

Implementation and Applications

In this final part of the work, we present in Chapter 9 the detailed architecture of our protypical implementation of the Web service description formalism developed in Part II and the algorithms developed in Part III. We will then present how our methods have been employed in the context of research projects in Chapter 10. In Chapter 11, we conclude by summarizing our main contributions and referring to our own publications, in which the results presented in this work have published. Furthermore, we discuss some open problems that may be of further research interest.

9. Implementation

In this chapter, we present the protypical implementation of our semantic Web service modeling and matchmaking tool. We call the tool KASWS which stands for Karlsruhe Semantic Web Services Framework. KASWS is an open source tool and hosted at http://sourceforge.net/projects/kasws/. We present the architecture of the system and discuss its various components in detail. Figure 9.1 shows the overall architecture of KASWS.



HTML Form To WSDL
HTML Form
Dynamic WS Binding Application
WSDL
Web Crawler
WSDL To KASWS
WSDL
search, compose, select web services
Edit, Add WS descriptions. Edit, Add Ontology Mappings
semantic descriptions of web services
FOAM
KASWS Web Services
KASWS Reasoning API
KASWS WS Mgmt. API
automatically found ontology mappings
KAON2 Reasoning API
KAON2 Ontology API
WS Descriptions Ontologies, Mappings

Figure 9.1.: Architecture of Karlsruhe Semantic Web Service Framework

9.1. KASWS Web Service Management API

The KASWS system needs to maintain a large number of Web service descriptions in the formalism we have developed in Part II. In general every Web service description may be associated with a separate ontology and there may be mappings among such ontologies. So, the system also needs to manage a large set of ontologies. We use KAON2[1] to manage the repository. KAON2 is an infrastructure for managing OWL-DL ontologies with DL-Safe rules. KAON2 is a successor to the KAON[2] project. The main difference to KAON is the supported ontology language. KAON used a proprietary extension of RDFS, whereas KAON2 is based on OWL-DL. The KASWS Web Service Management API allows to work with the Web service descriptions in the repository.

While the ontologies associated with Web services can be directly managed by KAON2, we developed an extra component to manage the behavioural descriptions of Web services. Although KAON2 repository together with a repository of behavioural descriptions fulfilled the purpose, it was hard to manage them since they have to be synchronized all the time. Another drawback of having two repositories was that we needed to implement methods for the retrieval of information about the behaviour needed by the reasoning algorithms. The fact that KAON2 can efficiently manage any type of information expressable with OWL-DL and provides efficient query answering, that is retrieval of the information led us to the idea that modeling the behaviour of Web services as an ontology would save us to manage a separate repository.

9.1.1. Generating Domain APIs from Ontology

Mapping Ontologies To Packages

We map ontologies to java packages. For programmatic access, we generate for each ontology O an interface O^*. Our generator expects a simple mapping from logical URIs denoting ontologies to java package names. This enables flexible configuration of the mapping and enables collaborative ontology development. The mapping is needed in order to know the java package name a generated interface should belong to.

In KAON2 an ontology can include other ontologies. If the ontology O includes ontologies $O_1, \ldots, O_k$, the interface O^* is sub interface of $O_1^*, \ldots, O_k^*$, where O_i^* denotes the generated interface for the ontology O_i. If O does not include any models it is subinterface of `RootModel`, which is a manually programmed interface having methods as described below. Further, if an ontology O_i^* includes other ontologies, we generate import statements to import packages corresponding to the included ontologies according to the mappings between logical URIs and java package names. Interface `RootModel` has the following methods:

- `public RootClass[] getAll(Class c)` – returns all instances of the given type c from this model
- `public RootClass createNew(Class c)` – creates a new instance

[1] http://kaon2.semanticweb.org
[2] http://kaon.semanticweb.org

- `public void remove(RootClass c)` – removes a given instance c from the model
- `public void save()` – manages the model persistence
- `public RootClass rdfToJava(RDFTriple[] triples)` – allows objects serialised as RDF to be read back in
- `public RootClass[] executeQuery(String query)` – allows to use the expressivity of KAON2 queres within the model

Now, we consider the set of concepts C in an ontology O. For every concept $c \in C$, we generate methods `public` C^* `create`C^*`()` and `public` C^*`[] getAll`C^*`s()` in the interface O^*. The method `create`C^* returns a new object of type C^* whereas the method `getAll`C^*`s` returns all objects of type C^* that exist in the ontology O.

Mapping Concepts to Interfaces

In order to represent the concepts in java, we generate a Java interface for each concept. The root concept of the ontology, $\top$, maps to a manually programmed interface `RootClass` with the following methods:

- `public String getID()` – return a unique identifier string
- `public RDFTriple[] toRDF()` – serialises this concept as RDF triples
- `public String toHTML()` – returns a human-readable HTML representation for displaying at the user interface

More formally, for a given set of concepts C, we generate for each concept $c \in C$, an interface c^*. The package of the interface c^* is same as that of the interface O^*, if the set of concepts C belongs to the ontology O. Further, if the concept c is sub concept of $c_1, \ldots, c_k$, the interface c^* is sub interface of $c_1^*, \ldots, c_k^*$, where $c_1^*, \ldots, c_k^*$ denote the interfaces generated for the concept c_i. If c has no super concepts, c^* is subinterface of `RootClass` only.

Mapping Relations to Methods

Now, we consider the set of relations R of the ontology O. While processing a concept c, we are only interested in those relations that have the concept c as their domain concept. That is, for a given concept c, we are interested in all relations $p \in R$ such that $c \in d(p)$. Let $R_c = \{p : p \in R \text{ and } c \in d(p)\}$. Now, we generate `get` and `set` methods in the interface `A` for each relation $p \in R_c$ as described in algorithm 3[3].

[3] we currently do not support mulitple ranges of a relation.

We support primitive Java data types via a small ontology. These are handled specially with a built-in mapping. We perform type conversions to and from primitive types internally as needed.

Algorithm 3 generateGetSetMethods(`Relation` p)

```
consider the set of range concepts of the relation $p$, $r(p)$, let concept $b$ denote the
range concept of the relation $p$, thus $r(p) = \{b\}$.
if $n_{max}(p) = 1$ then
  generate method "public "+$l_C(b)$+" get"+$l_R(p)$+"()"
  generate method "public void set"+$l_R(p)$+"("+$l_C(b)$+" value)"
else
  generate method "public "+$l_C(b)$+"[] get"+$l_R(p)$+"()"
  generate method "public void set"+$l_R(p)$+"("+$l_C(b)$+"[] values)"
end if
```

Mapping Instances to Proxy Objects

At runtime, ontology instances are mapped to Java dynamic proxies. Instances of this class `java.lang.reflect.Proxy` can implement a set of interfaces at runtime. Method calls are redirected to the method `invoke` of an implementation of the class `InvocationHandler` of the Java package `java.lang.reflect`. We implement our interfaces using this technique and map method calls on objets to KAON2 API method calls inside the `invoke` method. Our proxy object instances are obtained by the developer through the methods of the `ModelImpl` in the same package. All method invocations are handled by `ClassImpl`, which uses the KAON2 API.

Now, we turn our focus to `ClassImpl implements java.lang.reflect.InvocationHandler`. Recall, that we have generated an interface for each concept c with methods `"set"`$l_R(p)$ and `"get"`$l_R(p)$ for each property. Now, we describe how these getters and setters are implemented inside the implementation of the `invoke` method of the class `ClassImpl`.

Set-methods

Since, the method names are generated from the name $l_R(p)$ of the relation p, we can extract it from the method name. At this stage, we do not have the URI of the relation p yet. Luckily, we can get the package name, in which the class on which the method `"set"`$l_R(p)$ has been called, resides. Then, we use the mapping between packages and logical URIs of the ontologies to get the logical URI of the ontology and construct the property URI from it. Now, we have the property URI and a value that should be set for the given instance. Depending on the maximum cardinality $n_{max}(p)$ of the relation p, the value to be set can either be a single value or an array of values. Recall, that we have also differentiated between these two case while generating the `"get"`$l_R(p)$ methods. The internal methods `setPropertyValue` and `setPropertyValues` set the values for the relation p by directly using the KAON2 API. Though KAON2 allows to model the cardinalities, it does not check whether the cardinality constraints are fulfilled when relation instances are inserted in the ontology. The `setPropertyValues` method solves this problem by first checking whether the number of elements in array fulfils the cardinality constraints. If not, it throws an exception, otherwise it sets the values by calling the appropriate KAON2 API methods.

Get-methods

Here, we have to differentiate, whether the return type is an array or not. Here, we will describe only the former case.

To construct the required array, it is not sufficient to retrieve the instances that are in relation p with the given instance and then wrap them to the type of the array. The reason is, if two concepts a and b are related by p, every subconcept a_i of a is also related by p with every sub concept b_j of b. Which means, if an instance α of any a_i is related with an instance of any b_j and the method `getp` is called from α^*, then it should not deliver the objects of type b^*, but of type b_j^*. To achieve this, we first retrieve the set of instance that are in relation p with the given instance and obtain the parent class of the instances by first obtaining the direct parent concept and then mapping the concept to the java class. These steps are performed in the method `getParentClass`. We create an array of the parent class with appropriate number of elements. This array will be the return value of the method.

The only thing that remains is to fill this array. Again, we have to differentiate between two cases. The return type may be a primitive data type or an interface generated from the ontology. Here, we will describe only the latter case. In this case, we iterate through the set of instances that are in relation p with the given instance, wrap each instance to the parent class with the help of the method `wrapInstance` and insert the java object in the array.

Models At Runtime

Now we take a step back an look at the big picture. The class `ModelFactory` is a utility class to manage different ontologies. It can be parameterised with a KAON2 model, which is then the target of all data manipulations. We need this parametrisation to avoid passing the instance of `Model` as parameter to every constructor of a class that needs it.

Additionally, it has a static method

```
public static Model createModel(String logicalURI)
```

for creating new models. Every KAON2 ontology has a unique URI, which is also called logical URI of the ontology. This method expects such a URI. First it generates an instance of `OIModel` (KAON2-API class to manage a model) by passing the logical URI and then, it creates and returns a java object that wraps the KAON2 `OIModel` within a dynamic proxy , where `iface` is the class generated for the ontology with the given logical URI.

A Dynamic Model Implementation

The class `ModelImpl implements java.lang.reflect.InvocationHandler` is the dynamic implementation for all interfaces of type `Model` or its subinterfaces. We mentioned above the generated methods getAll$c$s and create$c$ in a subinterface of `Model`. Now, we describe their respective implementations within the method `invoke`. getAll$C$s – This method returns the set of all instances of interface c^* that exist in the ontology. create$C$ – This methods creates a new instance of type C^*. The method `createInstance` creates a new instance in the ontology via KAON2 API and is not described here any further. At this moment, only the `else` case is important, in which the class `ClassImpl` is used as the implementation of the interface.

9.1.2. Modeling the Knowledge Base with Description Logics

KAON2 can efficiently manage and reason about OWL-DL knowledge bases that may also include DL-safe rules. We have seen in the previous section how a domain specific Java API can be automatically generated from an OWL-DL ontology. Such an automatically generated API allows to rapidly build Java applications and lifts the availability of the description logic reasoning algorithms to the application layer.

Figure 9.2 shows different types of processes that are allowed to model the behavior of an agent (cf. Section 5.1.2). The concept Process is a central concept that represents an abstract process, whereas the sub concepts of Process correspond to the types of processes that build the behavior of an agent.

Figure 9.3 shows various properties of the different types of processes. Most notable are the concepts AgentIdentifier and Condition and the property definition. The concept AgentIdentifier denotes a named process description, whereas the concept Agent is an invocation of a named process. The property definition describes which agent identifier an agent is an invocation of.

Figure 9.4 shows the two types of sequential processes in more detail. Aligned with the terminology of π-calculus, we denote sequential composition with the concept Prefix. A sequential process performs either a communication activity or a local operation. Further, a communication activity is either an input activity or an output activity.

The concept AgentIdentifer denotes the set of agents $\mathcal{A}$. Instances of AgentIdentifer are the agent descriptions, in particular Web service descriptions. The concept Certifi-

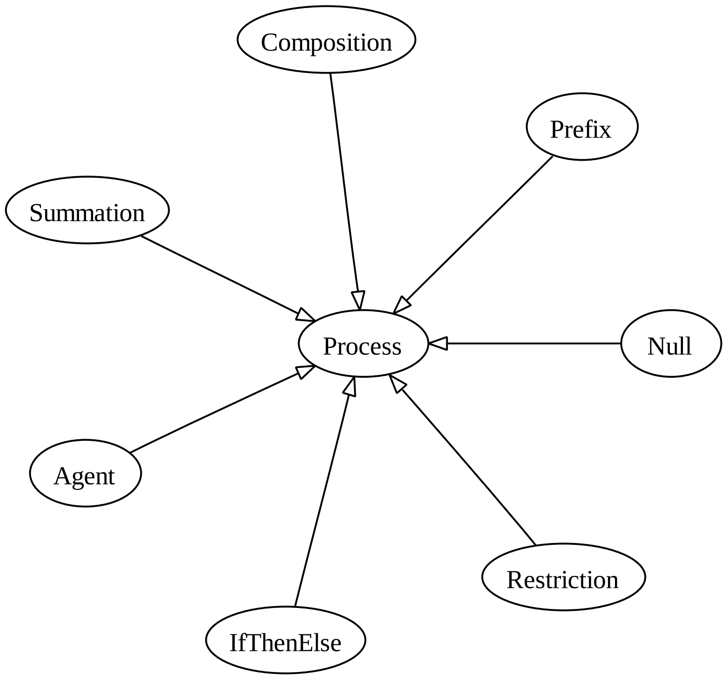


Figure 9.2.: Ontology for Describing Behaviour of an Agent

cationAuthority denotes the subset of agents that can issue credentials to other agents (refer to Figure 9.5). The concept Property represents the set of properties that can be issued by a certification authority to an agent. We foresee a description logic concept $\mathsf{P} \sqsubseteq \mathsf{Property}$, thus allowing an ontology of property names as described in Chapter 5. The concept Credential represents the set of credentials $\mathcal{C}$. The properties issuer and recipient with domain Credential and range AgentIdentifier represent the issuer and recipient agent of a credential, whereas the property about with range Property denotes the certified property.

The property definition of AgentIdentifer with range Process connects an agent to a process that describes its dynamic behaviour. The concepts Input, Output, Local, Agent, Composition, Summation and IfThenElse represent different types of processes that our formalism supports. The property hasID of the concept Agent connects it to an AgentIdentifier and thus allows to call a Web service from another Web service. The properties next, first and second have obvious meanings. The concept Channel represents the set of communication channels connected with properties hasType with the concept MessageType.

Note, that the sole reason of modeling the behavior of Web services as an ontology is to reuse the data management facilities of KAON2 and thus save time in the software

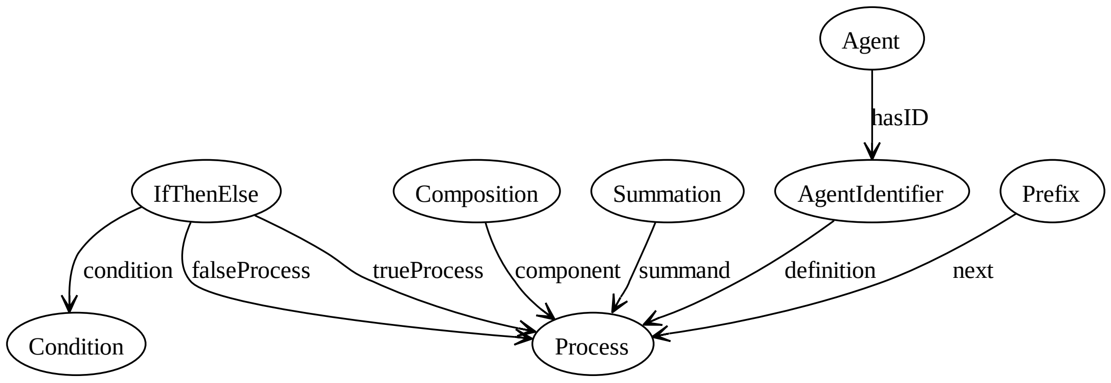


Figure 9.3.: Ontology for Describing Behaviour of an Agent

development process. Reasoning about the temporal aspects of Web services has to take place outside the KAON2 DL reasoner, since DL reasoning does not support changes in the knowledge bases during a run of a DL reasoner. At this point we wish to remark that the process ontolgoy presented above, though being rather small as compared to OWL-S process model, allows to model processes that are Turing complete, whereas the expressivity of the OWL-S process model is unknown. Furthermore, OWL-S has till date failed to point out that the added value of modeling behaviour as a DL ontology is that it enables the reuse of ontology managment infrastructures for managing behavior descriptions and not that DL reasoning can be used to reason about the dynamics of processes.

9.2. KASWS Reasoning API

KASWS reasoning API uses KASWS Web Service Management API to retrieve required information from the repository. KASWS Reasoning API provides reasoning algorithms for checking satifiability of a formula, checking whether a Web service description is a model of a formula and checking whether a Web service can simulate another Web service. As shown in the Chapters 6, 7 and 8, the KASWS reasoning algorithms need to check for concept subsumtion or perform a query on a description logic knowledge base at various stages. We use KAON2 reasoning API for performing reasoning tasks on the ontologies of Web services.

KASWS Reasoning API also provides API methods to create formulas of the logic $\mathcal{B}$ developed in Chapter 6. It also provides a concrete syntax for the formulas and contains a parser written in JavaCC [4]. JavaCC (Java Compiler Compiler) is an open source parser generator for the Java programming language. JavaCC is similar to Yacc in that

[4]https://javacc.dev.java.net/

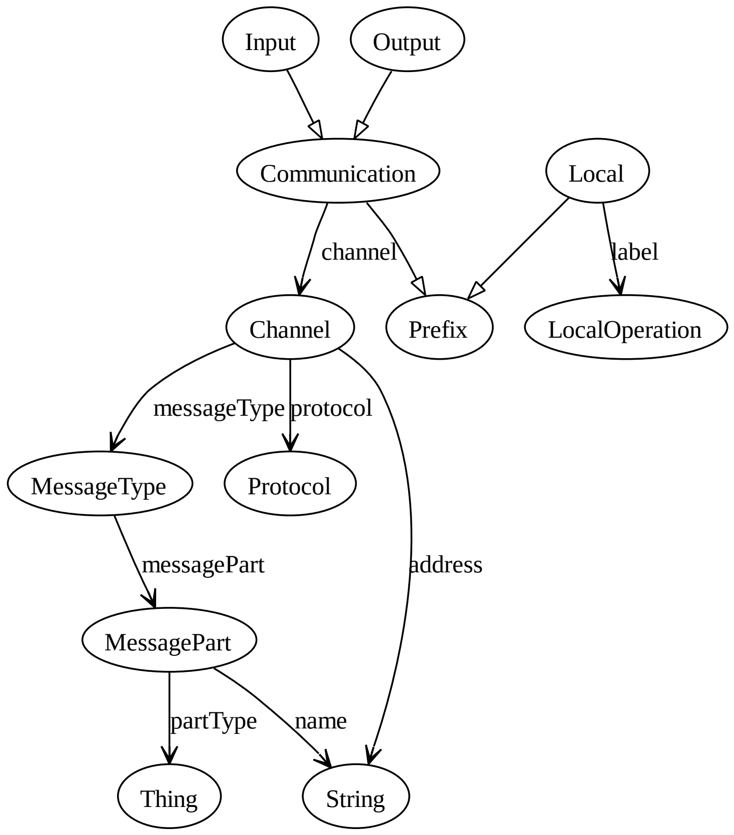


Figure 9.4.: Ontology for Describing Behaviour of an Agent

it generates a parser for a grammar provided in EBNF notation, except the output is Java source code. Unlike Yacc, however, JavaCC generates top-down parsers, which limits it to the LL(k) class of grammars, in particular, left recursion cannot be used.

Furthermore, the search, compose and selection algorithms[5] are exposed as Web services so that external client applications can embed KASWS funtionalities in their applications. From this perspective, KASWS system is used like a UDDI repository with the difference that it allows much more expressive search and thus the matches are more likely suitable for direct (without human intervention) dynamic binding than those found in a UDDI search.

[5]Note, that the architecture of the KASWS system is designed to provide composition and selection facilities, although the topics were not the focus of this work

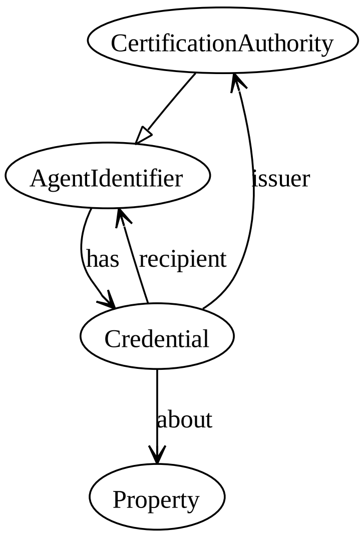


Figure 9.5.: Ontology for Describing Non-Functional Properties of an Agent

9.3. Bootstrapping

In order to show the suitability of our Web service description formalism for real Web services and evaluate the performance of our algorithms, we needed a significantly large set of Web service descriptions. Obviously, one can always generate arbitrary large set of Web service descriptions randomly. However, since showing suitability of our formalism for randomely generated descriptions would be a weak argument for practicability, we aimed at using descriptions of Web services that already exist and used in practice.

So the first idea was to develop a tool for converting WSDL documents into descriptions in our formalism automatically. Obviously, an automatically generated description is often not semantically correct. Apart from that a lot of information that is present inside the `documentation` tag in natural language is not considered during the conversion. Therefore, automatically generated descriptions have to be manually completed or rectified. Even though, automatic conversion tools like WSDL to KASWS converter are very helpful, since one does not have to describe Web services from scratch, which saves time. Another advantage of such tools is of psychological nature. Most people have less hesitation to rectify information rather than creating new information. Probably, the best known system that works on this assumption is CiteSeer [6]. In the following, we describe how WSDL to KASWS converstion tool works. The tool was developed in the scope of a student research project [Bai06].

A WSDL document describes more than one Web service. For each Web service, there

[6] http://citeseer.ist.psu.edu/

is an `Operation` block inside the `portType` block. The XML schema used for describing the Web services is defined in the `types` block. Each `Operation` block contains operation name, input message and output message.

- For each operation in the `portType` block create a new instance of the concept AgentIdentifier. For example, consider the following fragment of a WSDL document.

  ```
  <wsdl:portType name="CityInfo">
  <wsdl:operation name="getCityInfo" parameterOrder="plz">
  <wsdl:input message="impl:getCityInfoRequest" name="getCityInfoRequest" />
  <wsdl:output message="impl:getCityInfoResponse" name="getCityInfoResponse" />
  </wsdl:operation>
  </wsdl:portType>
  ```

  The `portType` specifies an operation `getCityInfo`. Therefore, a new instance getCityInfo of AgentIdentifier is inserted in the knowledge base by calling appropriate methods of the KASWS WS Management API.

- Similarly, instances of the concept MessageType are inserted in the knowledge base for the input and output parameters. In our example, `getCityInfoRequest` and `getCityInfoResponse` are input and output message types respectively. Both message types are put together inside the operation `getCityInfo`. The order of occurrence of the messages inside the `Operation` block (in our example, first input message then output message) defines the data flow (in our example it corresponds to request-response message exchange pattern).

  ```
  <wsdl:message name="getCityInfoResponse">
  <wsdl:part name="getCityInfoReturn" type="tns1:City" />
  </wsdl:message>
  ```

  The parts of a message type are listed inside the `message` block. In our example, the message type `getCityInfoRequest` contains only one part `plz`.

- For every complex type the XML schema described in the `types` block, a corresponding concept is inserted.

  ```
  <complexType name="City">
  <sequence>
  <element name="cid" nillable="true" type="xsd:string" />
  <element name="latitude" type="xsd:int" />
  <element name="longitude" type="xsd:int" />
  <element name="name" nillable="true" type="xsd:string" />
  </sequence>
  </complexType>
  ```

  For the above example, concept City for the complex type `City` is inserted in the knowledge base. Furthermore, for the elements of the complex type `cid`, `latitude`, longitude and name, object properties with corresponding names and domain City are inserted. For the range of the properties, we do not use primitive data types like xsd:string, xsd:int etc., since they rather represent the serialization types needed at the time of execution of the Web service and it is not possible to relate them to other concepts of the ontology via subsumption. We rather create concepts Cid, Latitude, Longitude, and Name from the property names by replacing the first character of a property name by its upper case representation. These concepts are then defined as ranges of the corresponding properties by inserting appropriate axioms in the knowledge base with the help of the KASWS WS Management API.

- Finally, the instance of the AgentIdentifier is connected to an instance of the concept Process via the property definition. The instance of Process is derived from the data flow information inside the Operation block. In our example, the process is an instance of the concept Input connected via the property next with an instance of the concept Output.

While looking for WSDL documents in the Web, we faced the problem that there are not many WSDL documents available or not easily accessible. To overcome this problem, we used a slightly modified version of the open source tool FORM2WSDL [7] to convert HTML forms to WSDL. Having the two converters, we could easily generate a large number of semantic description of Web services.

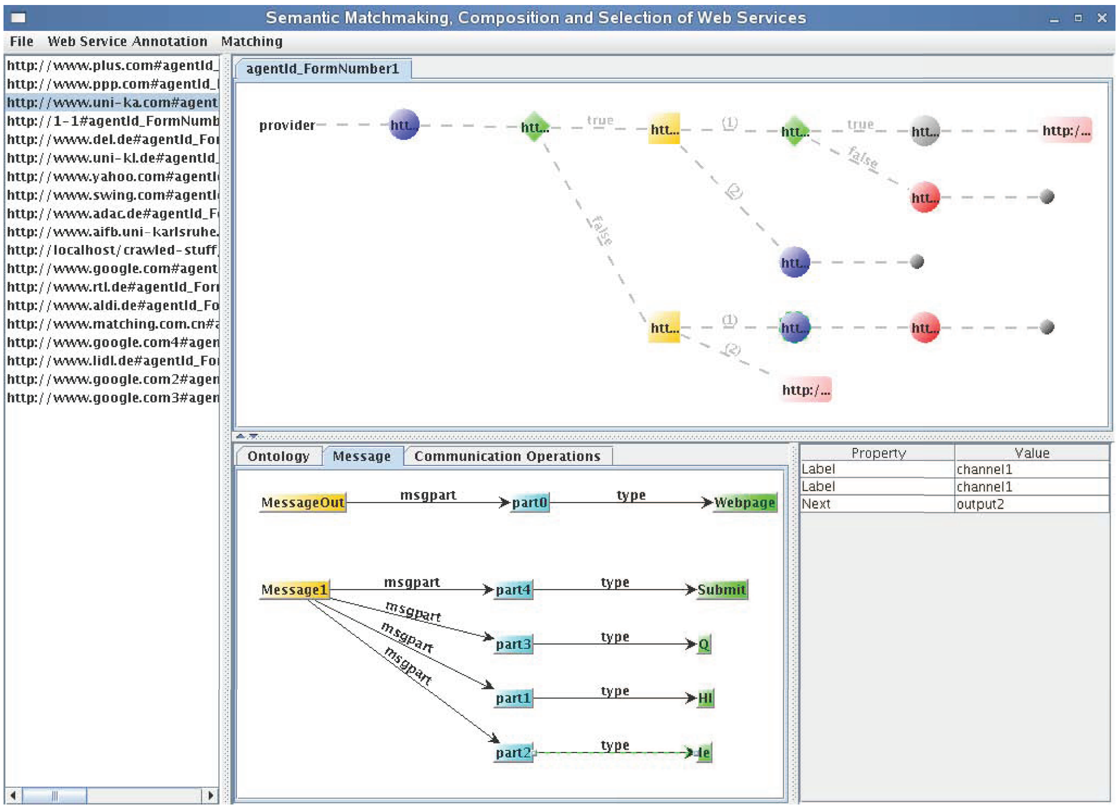
Since the types used in various WSDL documents are not interconnected and it is a time consuming task to map the large number of ontologies manually, we use the tool FOAM [8] for detecting simple mappings between the ontologies automatically [ESS05].

9.4. User Interface

The Graphical User Interface component supports a user to do various tasks.

- It allows to describe a new Web service including the corresponding ontology graphically and save it in the knowledge base.
- The automatically generated Web service descriptions from WSDL documents and HTML forms have very simple temporal structure due to the missing information in the corresponding documents. Apart from that, the automatic generated description may be semantically incorrect. Therefore, the GUI allows the user to edit existing descriptions and ontologies. Similarly, since the automatically found ontology mappings can be faulty, the GUI allows to manually edit the ontology mappings and add new more expressive ones [HM05].

[7] http://www.yourhtmlsource.com/projects/Form2WSDL/
[8] http://www.aifb.uni-karlsruhe.de/WBS/meh/foam/



http://www.plus.com#agentId_
http://www.ppp.com#agentId_
http://www.uni-ka.com#agent
http://1-1#agentId_FormNumb
http://www.del.de#agentId_For
http://www.uni-kl.de#agentId_
http://www.yahoo.com#agentI
http://www.swing.com#agentI
http://www.adac.de#agentId_F
http://www.aifb.uni-karlsruhe.
http://localhost/crawled-stuff
http://www.google.com#agent
http://www.rtl.de#agentId_For
http://www.aldi.de#agentId_Fo
http://www.matching.com.cn#a
http://www.google.com4#agen
http://www.lidl.de#agentId_Fo
http://www.google.com2#agen
http://www.google.com3#agen

Property	Value
Label	channel1
Label	channel1
Next	output2

Figure 9.6.: Graphical user interface showing the details of the matches

- The GUI allow a user to specify his request graphically and send it to the KASWS reasoner. Furthermore, a user can select a Web service from the list of Web service and search for similar Web services. The matches are presented graphically to the user.

10. Applications

In this chapter, we present how some of the formalisms and techniques developed in this work and presented in the previous chapters have been used in concrete setting given by the research projects we have been involved in. Further we discuss some potential applications in which our formalisms and techniques can be directly used.

10.1. Knowledge Integration in SemIPort

SemIPort stands for “Semantic Methods and Tools for Information Portals”. It is a research project funded by german ministry for education and research (Bundesministerium für Bildung und Forschung (bmb+f)). The overall goal of the project is to develop innovative techniques and tools for semantic based information portals.

The number of information sources in the Web is growing day by day with a tremendous speed. At the same time, users tend to manage more and more tasks electronically, thereby using publicly available information sources. Since the Web is an open, distributed and dynamic environment, in which information sources are offered and controlled autonomously and independently, the interoperability of available information is still a big challenge. The ever growing number of information sources in the Web and the ever increasing tendency of users to accomplish more and more tasks electonically give rise to the need of dynamic and flexible methods for knowledge integration. Aim of the work package “Knowledge Integration” in the SemIPort project is to develop flexible techniques to integrate different heterogeneous information sources. The information sources can be already equipped with schemas or ontologies. The technique should allow dynamic integration of the information sources provided by the user.

In this section, we present how our formalisms and techniques were used for integrating various information sources and illustrate our approach by an example. Our approach also allows dynamic and flexible integration of information sources provided by the users.

The heterogeneity of information sources can be classified in the following three categories.

- **Heterogeneity in software and hardware architectures**. For example, different operating systems, different programming languages and different hardwares like PCs, mainframe, Sun etc.. When various heterogeneous information sources are supposed to be integrated they must be able to communicate with each other.

Since the information sources may have different software and hardware architectures their native access methods differ. For example, software written in Java is easily accessible with Java RMI, whereas software written in C is easily accesible via sockets. Therefore, we need a more abstract communication medium that is suitable for accessing any information source. This is where Web services come into play. Web services provide a standard communication mechanism for controlled access to different heterogeneous information sources.

- **Heterogeneity in the structure and syntax of the information sources**. For example, relational databases, object-oriented databases, XML databases etc.. In order to overcome this problem, we abstract from the concrete structures and syntaxes and use OWL-DL ontologies for modeling the information.
- **Heterogeneity in the modeling of the information stored in the information sources**. For example, one information source models firstname and lastname as one attribute say name, the other models them as two attributes. Other causes are different languages, for example german and english and different naming conventions. We assume that the ontologies of information sources that can be subscribed by a user are known. The access to the content or functionalities provided by an information source is done via Web services and the ontology of an information source is published alongwith the description of the web services. When a user subscribes a new information source the new ontologies must be integrated with the already existing ontologies. For mapping among the various ontologies, we rely on methods known from [HM05]

An information source consists of information and operations that can be performed to access or modify the information. Current approaches for schema integration consider the integration on the data level and can be used for generating an integrated view of the schemas [Len02]. Hence, they are suitable for integrating information but not for information sources and offer solutions for relatively static situations.

To enable dynamic and flexible integration of information sources, the operations of an information source must also be taken into account. So, we propose a Web services based approach for flexible and dynamic integration of heterogeneous information sources. Our approach also allows integration of information sources provided by the users. The basic idea lies in abstract specification of information as well the operations of an information source. We specify each use case as a process with the formalism that we introduced in Chapter 5. Once the information sources are described in such a way, the integration of the sources can be specified by composing various relevant processes.

10.1.1. Sample Scenario

In our sample scenario, we consider the integration of multiple bibliographic information sources in a semantic portal. These information sources are heterogeneous both with

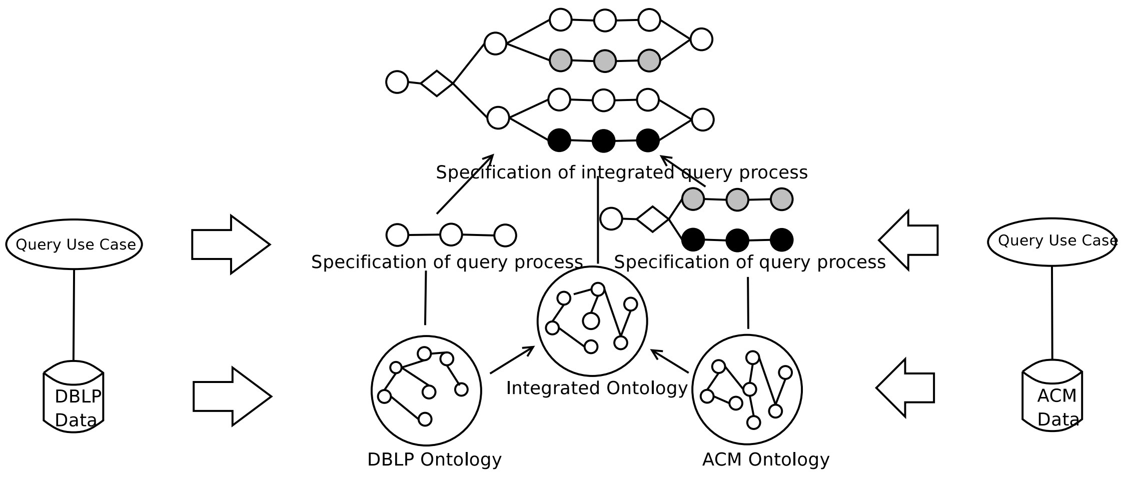

Specification of integrated query process
Query Use Case
Specification of query process
Specification of query process
Query Use Case
DBLP Data
Integrated Ontology
ACM Data
DBLP Ontology
ACM Ontology

Figure 10.1.: Specification of Information Sources

respect to their schema as well as the operations they provide. Suppose, one of the information sources is the DBLP bibliographic database, organized according to the DBLP metadata scheme. The DBLP database can be queried using a simple operation with the query as the input and the query result as the output. Suppose, another data source in the scenario is the ACM Digital Library, which uses its own metadata scheme for organizing its data. Also, the process for querying the ACM Digital Library is more complex: The user has the choice to either logon to the library and use more advanced search capabilities (e.g. on a more extensive dataset), or he may perform query operations without providing account information. When integrating these information sources to provide transparent querying in a semantic portal, we need to provide an integration on both the schema and the process level. We will use this sample scenario throughout the following sections to explain the model of process-based integration of information sources.

10.1.2. Integration of Heterogeneous Information Sources

In this section, we show how the schemas of information sources can be specified with ontologies, giving them formal and hence machine understandble semantics. Further, we show how the operations offered by an information source can be specified homogeneously with our process decription language introduced in Chapter 5. Having the formal descriptions of the various schemas and processes, we the show how various information sources can be integrated dynamically and flexibly, once their schemas and operations have been specified formally and abstractly, simply by composing the desired processes (refer to Figure 10.1).

Specifying Schemas with Ontologies

The integration of the information residing in heterogenous information sources requires to consider the intended meaning of the information to achieve an interoperability on a semantic level.

A database schema typically only describes the structural representation of the information. Ontologies feature richer modelling primitives, such as concept and relation hierarchies as well as axioms. Ontologies thus provide the means to formally specify the conceptualization of a domain, allowing for a semantic interoperability. Using ontologies to specify the meaning of the schemas therefore enables to semantically integrate the underlying information sources [WVV+01].

In many cases, a description of the structural schema of the information sources is already given. Several approaches exist to extract the conceptualization inherent in a schema, such as relational schemas and XML schema as an ontology [VOSS03]. The relationship between the ontology and the information sources they describe is established via mappings that resemble and enrich the structure of the information source and are used to define and annotate terms and resources from the information source and its schema.

To integrate the schemas of multiple heterogeneous sources, either a global ontology is derived which integrates local schemas, or each information source has its own ontology and the different ontologies are linked directly. Hybrid solutions combine the two approaches by building ontologies of single information sources using elements from a shared vocabulary. Again, mappings are used to express the relationship between the ontologies. The mappings can either be defined manually or automatically by using lexical relations, top-level groundings and semantic correspondences.

For the sample scenario, the content of the ACM and DBLP information sources would be described using an ontology reflecting their respective metadata schemes. However, the user would be presented with an integrated ontology (obtained by one of the means described above), allowing transparent querying against the various information sources.

Specifying Operations as Processes

In general, an information source has many use cases. For each use case, it offers operations to users. In simple cases, such operations are like remote procedures, but in general, especially in case of Web portals, they can be more complex and need multiple user interactions. Further, an information source performs local operations as partial tasks of a user case, e.g. calculations or database query.

In this step, we model each use case of an information source as a process with our process description language introduced in Chapter 5. The process description uses the ontology described in the previous section in order to refer to any objects that take part in the process. The local operations are specified as local operation activities and can be mapped to a concrete implementation, e.g. a method in a Java bean. Any interaction with the user is specified as communication activities. In general, an information source can itself be an integrated information source. That is, it may use operations of other information sources. In such a case the owner of the information may or may not want to disclose this. In the first case, he specifies the operation as process activity and in the second case as a local activity.

For our sample scenario, the querying of the DBLP database can be specified as a sequence of a message, a local operation and another message. The querying of the ACM Digital Library consists of a sequence of a message and a choice. The message is for asking the user about his choice. The choice consists of two sequences, which are similar to the sequence for DBLP.

Integration of Sources by Composing Processes

We assume the situation that there are many information sources that have been specified as described in Section 10.1.2 and their specifications (ontologies and process descriptions) are publicly available. Process descriptions serve as semantical descriptions (at least execution semantics) of the corresponding operations. In order to have an integrated system spanning over some information sources, an actor identifies for each use case how its functionality can be implemented by using the operations provided by other information sources or locally available [AHS03, AHS05, SPAS03]. The operations of other information sources can be embedded as process activities or local activities in the description of the use case. Since the basic activities of a process have concrete groundings, such a process specification is directly executable. Mappings between different ontologies can be integrated as local operations to support interoperability at the data level.

For our sample scenario, the process for querying the integrated system consists of a sequence of a message and a choice. As in ACM, the message is for asking the user about his choice. The choice contains for each choice value two sequences (one for querying DBLP and the other for querying ACM) that are executed in parallel.

In this section, we have shown how the declarative formal specification of heteregenous information sources help to integrate the information sources in a flexible way. A user can easily add or remove information sources as per demand dynamically and the integrated "virtual" information source can be queried efficiently.

10.2. Modeling Auctions in SESAM

SESAM[1] stands for "Self-Organisation and Spontaneity in Liberalized and Harmonized Markets". It is a research project funded by the German Ministry for Education and

[1]http://www.sesam.uni-karlsruhe.de

Reseach (bmb+f) under the reseach program “Internetökonomie”.

10.2.1. Introduction

In order to achieve a flexible implementation of market mechanisms in the SESAM prototype, it is necessary to describe the course and regulations as well as the necessary calculations of an auction in a declarative way. Such a description should be composable in the sense that it should consist of of reusable components, that can be replaced and recombined without easily to form new auctions. On the other side, such declarative descriptions enable the use of generic execution engines for running an auction. *Minimal Market Model* developed in one of the earlier work packages play an important role by providing basic data structures.

On one side, in dynamic environments such as virtual power plants, it should be possible to customize market mechanisms during run time of the system. In the current implementation of the system though the allocation service can be replaced completely; it is neither possible to customize an existing allocation service nor it is possible to undertake any dynamic customizations during the current market events. On the other side, decomposing market mechanisms in functional units build a basis for the decentralization of mechanism in the following work packages.

To cover both the requirements - the flexibility of market mechanisms as well as their decentralization - we chose to model the market mechanisms declaratively as processes that use reusable components. The approach uses the minimal market ontology developed in the work package 2.1 “Minimal Market Model” of the project SESAM [RA03, RNW04] and the modeling formalism presented in Chapter 5 for modeling dynamics of a market mechanism.

In the following, we first present a reference model for market mechanisms that is based on the earlier works in the field of auctions [RE05]. Then we present how the processes of market mechanisms can be modeled.

10.2.2. Reference Model for Market Mechanisms

Figure 10.2 depicts the architecture of the auction reference model as introduced by [RE05]. On the bottom of the Figure 10.2 is the data basis that contains the market instance data. The parallelogram in the middle of the picture represents an excerpt of an auction mechanism that always consists of stages or phases. There is only one stage active at a time and the sequence of stages is controlled by the auction flow. For example, in a classical english auction, stages are executed sequentially The first phase accepts the intention of the seller about his offer. The second phase accepts the offers of the bidders. The third phase calculates the result of an auction. Each stage contains componenents of four types, *view*, *validation*, *agreement generator* and *transition*. At its top the Figure 10.2 shows participants. Naturally, they do not belong to the auction mechanism

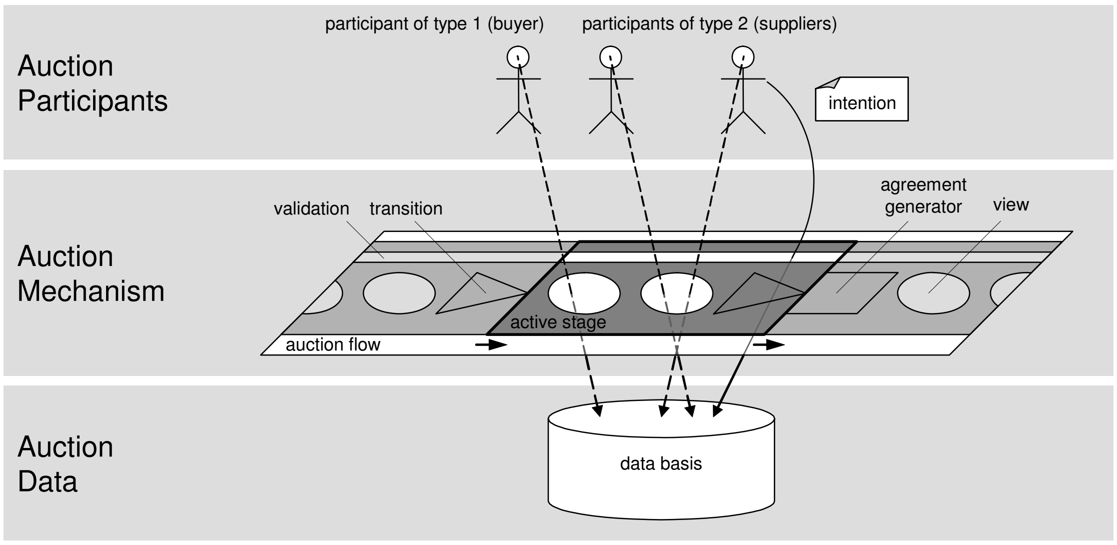


Figure 10.2.: Architecture of the Auction Reference Model

itself. They are primarily displayed for illustrating how views take effect. The middle and the right participants are of the same type and therefore share the same view and consequently can see the same market instance information in the active stage. The participant of the left is of another type and therefore can only access another view which implies different information visibility of him. The right participant submits an intention that is checked positive and forwarded to the data basis by the validation component.

Data Structure

Figure 10.3 shows Minimal Market Model as UML class diagramm. As shown in Figure 10.3 an *Intention* is defined by relationships to a participant und two types of products, namely incoming product and outgoing product. Incoming products are those, that the corresponding participant wishes to obtain, whereas he is willing to give the outgoing product in return. An Intention can be defined as binding, which means that the corresponding participant is commited to fulfill the terms in the offer. If an Intention is not binding, it just represents an information in the market, which is by no means a commitment of the participant. An agreement describes the connection between two intentions in a deal. If more than one intentions are connected by XOR, it means that only one of the intentions can be accepted in an agreement. All other with XOR connected intentions the become invalid [RNW04].

The data structure for Minimal Market Model was introduced in the work package 2.1 “Minimal Market Model” of the project SESAM [RA03, RNW04]. This data structure serves as static model for modeling market mechanisms as processes. Since the Minimal

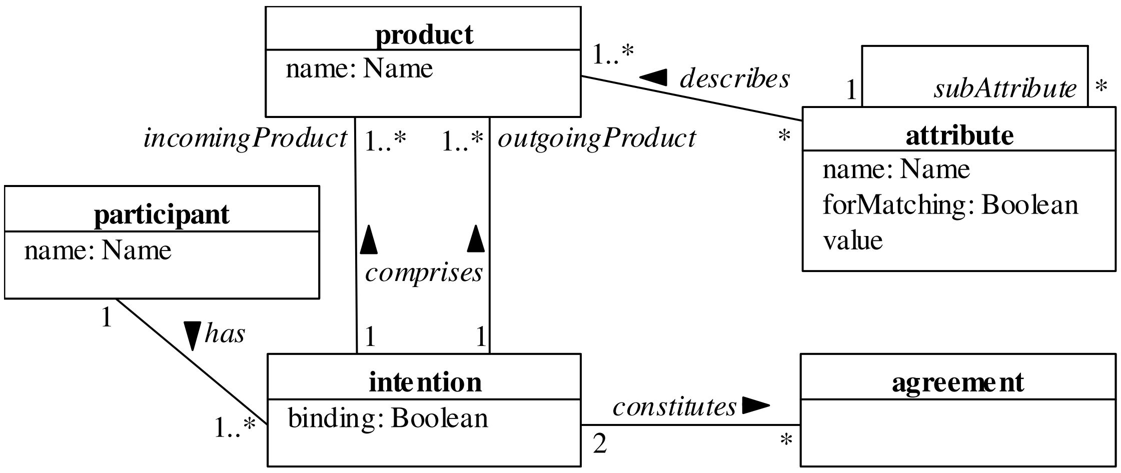


Figure 10.3.: UML Class Diagramm for Data Structure of Minimal Market Model

Market Model (MMM) is generic and can be applied to every kind of market, its data structures can be overtaken in auctions and negotation scenarios. For the scenarios of combinatorial nature, in which various products and tarifs can be combined in an offer or bid, the MMM was extended by a logical XOR-component.

View

The theory of the pure MMM is not primarily designed for creating views on the market data. However, while dealing with market mechanisms, it is essential to have a possibility to provide different participants different views on market data at different points of time. For this purpose, we use views. There can be zero to many views in a phase. The access on the views and thus on the information they deliver is controlled by access control policies. Every view has principly access to all all the data in the data base that is assigned to it. The view can limit this data or expand this data(e.g. by calculating an average of some values), before it delivers the data to the participant.

An illustrating example of the views is the second phase of the english auction. The seller has the right to see all bids together with the corresponding participants and prices entered until some point of time. Whereas, the participants in the role of a bidder are only allowed to see the current highest price.

Validation

In every phase, there is exactly one validation component. During the whole execution of the corresponding process, the component checks whether incoming intentions are admissible in the respective aucton or negotiation. In case they are, they stored unchanged in the data base; otherwise they are rejected.

In the english auction for example, the validation component in the second phase checks whether the corresponding intention of the incoming bid has a higher price than the current highest price. Only if this is the case, a bid is accepted.

Transition

The purpose of transitions is to end the phase they are assigned to. A simple example of an english auction with fixed end is the transition in phase two. Such a transition can wait exactly one hour, for example with the help of a timer, and then terminate the bidding phase.

Agreement Generator

The generator for agreements is responsible for determining the results, that is, who exchanges which products to which conditions. A generator can create multiple agreements. A generator is executed at the beginning of a phase, which means it is triggered from the transition of the previous phase. In order to create agreements, a generator can access all intentions the data base as well as information about the dynamic of the market mechanism and the previous agreements.

In the third pahse of an english auction, an agreement generator takes the intention of the seller from phase one and the intention with highest offered price from phase two. It connects both the intentions in form of an agreement and reconciles the values, e.g. the final price in both the intentions.

10.2.3. Formal Description of Reference Model for Market Mechanisms

In this section, we show how we describe the reference model for market mechanisms with our formalism.

Data Structures

We model the necessary data structures with description logics concepts. The classes and the relationships among them correspond to those depicted in the UML class diagramm in Figure 10.3.

$$\begin{aligned}
\mathsf{Intention} &\sqsubseteq \exists \mathsf{incomingProduct.Product} \sqcap \exists \mathsf{outgoingProduct.Product} \\
\mathsf{Product} &\sqsubseteq \exists \mathsf{name} \\
\mathsf{Participant} &\sqsubseteq \exists \mathsf{name} \sqcap \mathsf{offers.Intention} \\
\mathsf{Attribute} &\sqsubseteq \exists \mathsf{name} \sqcap \exists \mathsf{forMatching} \sqcap \exists \mathsf{value} \sqcap \exists \mathsf{describes.Product} \\
\mathsf{Agreement} &\sqsubseteq \exists \mathsf{offer.Intention} \sqcap \exists \mathsf{acceptance.Intention}
\end{aligned}$$

Notable is the concept Agreement that constitutes of two intentions. One intention (connected via the property offer) represents the offering intention, that is one specified by the participant who is willing to sell some product, the other intention (connected via the property acceptance represents the intention specified by the buyer.

Modeling the Phases of an Auction

We model an auction as a process. An auction consists of multiple phases that are as processes as well. In the following, we show how the components of a phase can be modeled.

Views Since in general, a participant is allowed to access different informations at different points of time, it is not sufficient to bind the access rights to a participant statically. Rather the access control policies must be context dependent that is dependent on the state of the process. Therefore, we model views as processes with access control policies. A view without any access control policy means that every participant has access to the information provided by the view. A view is a process that answers the pre defined paramerterized query. The process expects as input the values for the parameters from the user. In case the user has the required credentials, the process sets the values in the corresponding parameters to obtain a syntactially correct query. It then poses the query to the knowledge base. The answer delivered by the knowledge base is then forwarded to the user.

Validation Each phase of an auction can receive intentions. However, the incoming intentions must fulfill certain conditions in order to be accepted in a phase. The conditions are dependent on the phase, that is, for different phases, the incoming intentions must fulfill different conditions.

We model such conditions as description logic predicates. For example, the condition that an incoming intention I_1 should be equal to an existing intention I_2, can be modeled with $\mathsf{equals}(I_1, I_2)$. In another phase, the condition that the price of an incoming intention I_1 should be higher than the current highest price can be modeled as

$$P_{max}(\mathsf{price}, \mathsf{maxPrice}) \sqcap P_{>}(\mathsf{newPrice}, \mathsf{maxPrice}).$$

Transitions Each phase has exactly one transition that represents the termination condition of the phase. The transition condition is checked at the end of an interation of a phase. In case, it is evaluated to true, the process representing the phase is terminated thereby yielding control back to the process that triggered it. In case, the condition is not met, a new iteraction of the phase is stated.

Agreement Generator Agreement generator is also modeled as a process, that inserts a new agreement in the knowledge base. Agreement genarator obtains as input two intentions and inserts a new instance of the concept Agreement in the knowledge base.

$$\mathit{user}(\mathsf{i}_1, \mathsf{i}_2).\mathbf{new}\ \mathsf{agr}.\mathit{setIntentions}(\mathsf{agr}, \mathsf{i}_1, \mathsf{i}_2).\mathbf{0}$$

where $\mathit{setIntentions}(\mathsf{a,i,j})$ is a local operation that adds axioms $\mathsf{offer}(\mathsf{a}, \mathsf{i})$ and $\mathsf{acceptance}(\mathsf{a}, \mathsf{j})$ in the local knowledge base. That is, the invocation of the local operation $\mathit{setIntentions}(\mathsf{agr}, \mathsf{i}_1, \mathsf{i}_2)$ will add axioms $\mathsf{offer}(\mathsf{agr}, \mathsf{i}_1)$ and $\mathsf{acceptance}(\mathsf{agr}, \mathsf{i}_2)$ in the knowledge base.

In this section, we have shown how auctions can be modeled with our formalisms presented in Part II. Modeling auctions in such a declarative and formal way brings many advantages. Firstly, it enables easy replacement of an auction by another without writing program code. Secondly, a generic process execution engine will suffice to run any sort of auction that can be modeled with the formalism and there is no need to implement each kind of auction separately. Thirdly, the formal nature of the modeling language allows to develop automatic reasoning procedures for reasoning about various properties of auctions.

10.3. Further Potential Applications

10.3.1. Verification of Business Processes

The Web services description formalism developed in Part II is based on the idea that every Web service has an underlying process that can be very simple like answering a simple query or very complex involving multiple interactions with multiple actors having different credentials and access rights. The formalism allows to model the functionality of Web services as distributed processes that may run in multiple environments. Hence, the formalism can be directly used to model business processes that run inside an organization or even across organizations. Similar arguments hold for the formalism and algorithms that we have presented in Part III. In particular, our goal specification language can be used for specifying constraints on business processes and the goal based matchmaking algorithm for verifying business processes. Furthermore, simulation based matchmaking algorithm can be used to detect duplicate business processes or sub processes which can be very useful for managing business processes effectively.

10.3.2. Reasoning about Market Mechanisms

In Section 10.2, we presented an approach for formalizing auctions by modeling them with our process description formalism presented in Part II. With the availability of an execution engine such models can be directly executed and easily replaced by other auctions in a larger system. Another advantage of formalizing the auction or in general market mechanisms is that one can reason about them automatically by employing our algorithms from Part III. For example, before participating in an auction, one could automatically detect whether an auction accepts only higher bids.

10.3.3. Model Driven Architecture in Software Development

We have presented in Section 9.1 an approach for generating Java API from a domain ontology in such a way that the API can be used in a typical programming style while the data is managed by the underlying ontology management system, in our case KAON2. Such a technique not only enables faster development since the conceptual model is the executable model are same but also lifts the reasoning capabilities of the underlying description logic reasoner to the application level. Similarly, our formalism for modeling processes (even without the credentials part) can be used to programm business logic by modeling instead of programming. In Section 9.1, we also presented a description logic ontology for modeling the dynamic behavior of a process, which allows to manage even the instances of a process with an ontology management system.

10.4. Conclusion

In this chapter, we have presented how some of the formalisms and techniques developed in this work and presented in the previous chapters have been used in concrete setting given by the research projects SemIPort and SESAM we have been involved in. Further we have discussed some potential applications in which our formalisms and techniques can be directly used.

11. Conclusion and Outlook

In this chapter, we will summarize our main contributions and align them to our own publications, in which the results have been published. Furthermore, we will discuss some open problems that are directly related to our work as well as ideas that can be of further research interest.

11.1. Contribution

In this work, we addressed the problem of automatic matchmaking of Web services. Part I of this work was meant for motivating the problem of automatic matchmaking of Web services and identifying requirements for such matchmaking. We began in Chapter 1 with the introduction of Web services, standards for describing and executing Web services, namely WSDL and SOAP as well as industrial standard of Web service discovery, namely UDDI. We motivated the problem of automatic reasoning about Web services, especially matchmaking of Web services by introducing the vision of Web service markets. We shared the idea of many other researchers that in order to achieve spontaneous and transparent Web services markets, current UDDI based discovery is not sufficient, since it requires a lot of manual effort.

Existing semantic Web service matchmaking approaches propose to use ontologies for achieving interoperability among the terminologies used in schemas of Web services as well as between request and offer descriptions. As a result, Web services can be found even if requests and offers are described using different terminologies provided there are semantical relationships such as subsumption among the terminologies. We argue in Chapter 1, that such approaches represent the first step in the right direction and there is need for more expressivity in order to achieve automated matchmaking. The main reason for this is that existing semantic Web service matchmaking approaches enable only limited automation, since they consider Web services as black boxes and match only their interfaces. Though interface matching is important to ensure that a Web service can be incorporated in a larger system, it is not sufficient Web service discovery. The main reason for this is that a user looking for Web services has some task that he wishes to accomplish by using a Web service. So, his primary interest is the functionality of the Web service, that is what a Web service actually does and not whether he can invoke a Web service in principle. Furthermore, we identify the need for two types of matchmaking approaches, namely goal based and simulation based. Existing matchmaking approaches fall into the category of goal based and to the best of our knowledge ours is the only work that has addressed the problem of simulation based matchmaking.

In Chapter 2, we presented requirements for expressive matchmaking of Web services. Since matchmaking algorithms work on the Web service descriptions, expressive matchmaking directly imposes requirements for an expressive Web service description formalism. In this work, we considered three aspects of Web services, namely, involved resources, credentials of involved actors and the dynamic behavior of the Web service process.

In Part II, we presented a formalism for specifying resources, dynamics and security aspects of invocable agents in a unified and interoperable way. We analyzed and compared available techniques for formalizing the three aspects and identified description logics, in particular $\mathcal{SHOIN}(\mathbf{D})$ with DL-Safe rule extensions for resources, SPKI/SDSI certificates for credentials and π-calculus as best suited for our purpose. In Chapter 3, we have shown how the resources of the actors involved in the execution of a Web service can be specified with expressive description logics, like $\mathcal{SHOIN}(\mathbf{D})$ [Aga04, AH04, AS06, AS07].

In Chapter 4, we have shown how the non-functional properties of the actors can be modeled in a way such that parties can build their trust in them as well as automatically reason about them. By modeling SPKI/SDSI certificates with description logics, we have given them a formal semantics. Furthermore, we have shown how certification policies, that are currently described in natural language, can be formalized with description logics axioms. Thus, reducing the task of verification of the eligibility of a user to answering a description logic query [ASW04, AS04, AS05].

In Chapter 5, we have shown how the dynamic behavior of Web services can be modeled. Since many Web service processes do not necessarily run entirely inside the Web, the problem of describing the functionality of Web service then became equivalent to describing any type of invocable agents and viewing Web services as special type of agents that that invocable via Web protocols. We showed how the notion of communication channels known from π-calculus and other process algebras can be augmented with protocol types to support mixed environments [Aga04]. Having a formalism for each of the three aspects that we considered in this work, we presented the overall formalism that combines the aspects of the Web services by connecting π-calculus names with descriptions logic ontologies, introducing local operations for modeling changes (updates) in the set of resources of the involved actors [AS06, AS07] and embedding credential based access control as conditions in the Web service process [ASW04, AS04, AS06, AS07].

The unified formalism has many useful properties. It can be used for describing executable processes as well compositions of Web services. So, in contrast to existing approaches, we do not require any additional formalism for describing Web service compositions. This is useful since while performing automatic composition of Web services, one anyway needs a way for "writing down" composed models. Furthermore, the formalism can be used to describe workflow systems in a formal way [Aga04, AS06, AS07].

Note, that describing workflow systems was not our primary focus in this work. However, the formalism we needed to develop for describing Web services is general enough and has formal semantics in contrast to most of the workflow languages like BPEL4WS.

Having a formalism for describing Web services in a very expressive way, we turned our attention to matchmaking of Web services in Part III. We considered two types of matchmaking approaches, namely goal based and simulation based. Goal based approach deals with specifying constraints for desired Web services, whereas simulation bases approach directly compares two Web services. So, the simulation based approach works directly with Web service descriptions, whereas the goal based approach needs a formalism for specifying desired constraints on the functionality of Web services. We developed such an expressive goal specification formalism in Chapter 6. We have bulit our goal specification formalims on the well known modal temporal logic μ-calculus, which is the most expressive decidable temporal logic. Our goal specification formalism emerges by using description logic queries in place of μ-calculus atomic propositions and augmenting μ-calculus atomic actions by structures. This enables support for different types of communication protocols, allows to differentiate between input and output actions and enables reasoning about static objects (resources) that are involved during the execution of a Web service [Aga04, AS06, AS07].

Having the formalisms for describing Web services and goals respectively at our disposal, we then developed algorithms for goal based matchmaking in Chapter 7 and for simulation based matchmaking in Chapter 8. The algorithm for goal based matchmaking builds on the technique known as local model checking that does not require explicit calculation of all the states of a system and hence avoids the state explosion problem to some extent. The algorithm is sound, complete and decidable. The algorithm for simulation based matchmaking gains from the theory of equivalence of processes, a thoroughly studied topic in the field of process algebras. However, since dynamic behavior is only one of the three aspects that we considered in this work, we needed to define notion of (bi)simulation of resources and credentials of agents [AS06]. We then presented the algorithm for simulation based matchmaking, that unifies the simulation relationships of behavior, resources and credentials. The algorithm checks for two given Web service descriptions whether the one Web service can be replaced by another without changing the overall behavior of the systems they are embedded in. The algorithm builds on the algorithms known from congruency theory of π-calculus augmented with the algorithms developed by us for checking simulation of resources and credentials [AS07].

In Part IV, the final part of the work, we presented the prototypical implementation of the formalism and algorithms developed in this work as well their applications in the context of research projects. In Chapter 9, we have presented the architecture of our open source prototypical implementation[1] and discussed its various components in detail. We have evaluated the performance of our algorithms on a fairly large set of Web

[1]The source code of the prototype is available at `http://kasws.sourceforge.net/`

service descriptions. For managing the set of Web service descriptions efficiently, we have first developed a technique to automatically generate Java API from an OWL-DL ontology [AV05]. Then, we presented an ontology for modeling agents, in particular Web services.

In Chapter 10, we presented applications in which the techniques developed in this work have been used among others also in the scope of the BMBF projects SeMIPort[AH04] and SESAM[EA04, AR05].

11.2. Open Problems Concerning Our Work

11.2.1. Annotating Web Services with Logic $\mathcal{B}$

In Chapter 7 we have developed a model-checking algorithm that checks whether a Web service description fulfills a formula $\phi \in \mathcal{B}$. The model checking algorithm checks whether the states of the transition system described by a Web service fulfill sub formulas of ϕ. It is worth considering whether we can achieve a more efficient model checking algorithm if the Web services are annotated explicitly with the sub formulas they satisfy.

Annotating a Web service w with a formula ϕ is actually nothing more than saying $w \models \phi$ explicitly. The semantics of the formulas of logic $\mathcal{B}$ is defined on the set of states of a process. So, we need to clarify the semantics of such annotations. A Web service process is a labeled transition system with a unique start state. Denoting the start state of a Web service w with $s(w)$, we can define the semantics of an annotation $w \models \phi$ as $s(w) \in [\![\phi]\!]$.

Another advantage of explicit annotations is that it makes possible to describe Web services semantically by simply categorizing them. Or in other words, our logic $\mathcal{B}$ can be used to describe the semantics of category based approaches like the MIT process handbook [2]. To achieve this, it is sufficient to give logic $\mathcal{B}$ formulas names. For example, one can define a book selling service as a process that eventually delivers a book as follows

$$\mathsf{BookSellingService} := \mathsf{true}\ \mathtt{until}\ [\bar{\mathsf{c}}(\mathsf{Book})]\mathtt{true}.$$

Now, if a Web service w is a book selling service, one would simply annotate the Web service w with BookSellingService instead of with the formula on the right side in the definition of BookSellingService.

However, note that annotating Web services with this technique that requires manual effort. An annotator has to find the categories (ideally all the categories) that his Web service may fit into and annotate his Web service with the categories. Finding appropriate categories is not only time consuming task but also requires human intelligence, since the annotator has to understand the semantics of a category by looking at its definition. In the above example, an annotator needs to understand what the category

[2]http://ccs.mit.edu/ph/

BookSellingService actually means before he can decide whether he can annotate his Web service with this category. On the other hand, while describing a Web service with the formalism presented in Chapter 5 a Web service provider only needs to describe what his Web service does, which is obviously known to him. Since the formalism for modeling Web services is a process oriented formalism and Web service are often implemented in procedural programming languages like Java, we hope that an initial description of a Web service can be automatically generated from the implementation code. Such an automatically generated pre-description then only needs to be fine tuned by the annotator. So, it is an interesting open problem to find how much efficiency we can gain from explicit annotations and whether the manual effort needed for annotations is worth it or not.

11.2.2. Semi-Automatic Support for Describing Web Services and Constraints

In this work, we have developed expressive formalisms for describing Web services and specifying constraints on Web services in Chapter 5 and Chapter 6 respectively. The prototypical implementation presented in Chapter 9 provides a user graphical interfaces for describing Web services as well as specifying constraints.

The WSDL2KASWS conversion tool can automatically generate first draft of the semantic description of Web services from their WSDL description. However, WSDL allow to model only minimal information about the behavior of a Web service and rest of the information about the dynamics is described inside the `documentation` tag in natural language. In this work, we did not deal with extracting formal descriptions from the information available in natural language. Another idea could be to go one step deeper than WSDL and extract Web service descriptions from the impmentation of the Web service.

Similarly, we did not deal with deriving the constraints on Web services a user may have (semi)-automatically. Since Web services are primarily meant for being incorporated in larger business processes, it might be possible to derive the goals from the business context.

11.2.3. Efficient Matchmaking

The matchmaking algorithms presented in Chapter 7 and Chapter 8 go through all Web service descriptions linear. That is, for n Web service descriptions in the repository, they check each of them one by one. In theory model checking algorithms work on one goal and one description and simulation algorithms on two descriptions. However, in the concrete practical problem of matchmaking, it might not be necessary. In typical discovery use cases, a requester is interested in all the matches. So, one might use efficient data structures for managing Web service descriptions and matchmaking algorithms that work on the whole repository in one shot rather than processing each Web service description one after another and independently of each other.

11.3. Further Research Ideas

11.3.1. Automatic Composition

In this work, we addressed the problem of matchmaking of Web services. We considered the situation that given a collection of Web service descriptions, how can the Web services from the collection can be detected that fulfill certain conditions or that offer the same functionality as a given Web service. In some cases, there may exist combinations of Web services that can fulfill the desired constraints or offer same functionality as a given Web service. The problem of finding such combinations is referred to as automatic composition of Web services and is a topic of big interest in the community. There has been already some work done in this field [MSZ01, MS02, SPW+04]. However, the results are still not satisfactory. In our opinion, one of the main problems with most of existing automatic composition techniques is that they incorporate AI planning techniques. AI planning techniques are typically backward chaining algorithms that given a desired output type or conditions in the real world (goal), a set of atomic actions (Web services) and client's knowledge base, compose a sequence of actions in order to reach the aimed situation. AI planning techniques are useful for example in controlling the movement of robots that have the task of moving from one position to other while having a fixed set of actions as its disposal. However, the main problem of Web service composition is different in that one is interested in finding a composition of Web services that has some desired behavior. Such a composition should be usable by any one and not only by the client who performed the composition. Note that AI planning based algorithms construct a client specific sequence of actions and thus the generated plans can be used only once. An approach that goes in the right direction in our opinion is presented in [BCG+05]. However, there are still some open issues that need further consideration. The approach presented in [BCG+05] is not goal driven but simulation driven. That is, given a Web service, the approach tries to build a mediator, which uses messages to interact with pre-existing Web services and the client such that overall behavior of the mediated system faithfully simulates the behavior of the goal service. The issue that remains open is the goal based automatic composition. We believe that our model checking algorithm can be generalized to construct mediators or synchronization skeletons as they are often called in the theory, to glue together Web services and thus obtaining Web service compositions. The goal based matchmaking approach that we have presented in this work can then be seen as a special case in which the mediator is empty, that is there is no need for glue. The approach presented in [BCG+05] is based on databases schemas and does not support complex type and mappings between schemas. An alternate approach could be to use ontologies in place of database schemas.

11.3.2. Preference Based Ranking

The matchmaking approaches presented in this work detect the set of Web service that satisfy a given goal or offer the same functionality as another Web service. The goals specification technique presented in this work does not allow specification of preferences on the desired properties. As a result, the matchmaking techniques presented in this work deliver a set and not a list ordered by the rank of Web services. Ranking of Web services can be important in practical settings, in which a user does not have hard constraints on properties but soft constraints described as preferences. For example, a user may wish to specify that he prefers if the credit card is charged after the delivery to if it is charged before the delivery. Having such preferences the matches can be automatically ranked and the top ranked match can be automatically incorporated in the business process of the user (cf. "I'm Feeling Lucky" button in Google). There is already some work done in this direction. There are approaches based on Fuzzy Logic, for example [AL05a, AL05b] and on utility theory [LAO+06]. However, it is still not quite clear how the specification of user preferences can be unified with our goal specification formalism. Another interesting work that goes in this direction is [BFM06], in which the authors address the problem of specifying and generating preferred plans using rich, qualitative user preferences with the help of a logic for specifying non-Markovian preferences over the evolution of states and actions associated with a plan. The approach supports qualitative rather than ordinal temporal preferences.

In this work we have presented a holistic approach for automatic reasoning about various aspects of Web services. Not only the holistic nature of our work but also the expressiveness of the developed formalisms represent a significant improvement to existing semantic Web service approaches. Considering the vision of semantic Web service markets, although there are still a few open problems that need to be solved, our formalisms build a solid basis and the tools developed in the scope of this work can be used to realize the first architecture of semantic Web service markets.

Bibliography

[ABFG04] Daniel Austin, Abbie Barbir, Christopher Ferris, and Sharad Garg. Web services architecture requirements. Technical report, W3C, February 2004.

[ABH$^+$02] Anupriya Ankolekar, Mark H. Burstein, Jerry R. Hobbs, Ora Lassila, David Martin, Drew V. McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne, and Katia Sycara. DAML-S: Web Service Description for the Semantic Web. In Horrocks and Hendler [HH02], pages 348–363.

[ACD$^+$03] Tony Andrew, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. Business process execution language for web services. Technical report, BEA Systems, IBM Corp., Microsoft Corp., SAP AG, Siebel Systems, 2003.

[AFM$^+$05] Rama Akkiraju, Joel Farrell, John Miller, Meenakshi Nagarajan, Marc-Thomas Schmidt, Amit Sheth, and Kunal Verma. Web Service Semantics-WSDL-S, A Joint UGA-IBM Technical Note, Version 1.0,. Technical report, IBM and LSDIS, April 2005.

[Aga04] Sudhir Agarwal. Specification of Invocable Semantic Web Resources. In Zhang [Zha04], pages 124–131.

[AH04] Sudhir Agarwal and Peter Haase. Process-Based Integration of Heterogeneous Information Sources. In *informatik04-Workshop Semantische Technologien für Informationsportale*, volume 51 of *LNI*, pages 164–169, Ulm, Germany, September 2004. Springer.

[AHS02] Anupriya Ankolekar, Frank Huch, and Katia Sycara. Concurrent Execution Semantics for DAML-S with Subtypes. In Horrocks and Hendler [HH02], pages 14–21.

[AHS03] Sudhir Agarwal, Siegfried Handschuh, and Steffen Staab. Surfing the Service Web. In Fensel et al. [FSM03], pages 211–226.

[AHS05] Sudhir Agarwal, Siegfried Handschuh, and Steffen Staab. Annotation, Composition and Invocation of Semantic Web Services. *Journal of Web Semantics*, 2(1):1–24, 2005.

[AL05a] Sudhir Agarwal and Steffen Lamparter. sMart - A Semantic Matchmaking Portal for Electronic Markets. In Guenter Mueller and Kwei-Jay Lin, editors, *Proceedings of the 7th International IEEE Conference on E-Commerce Technology 2005*, pages 405–408, Munich, Germany, July 2005. IEEE Computer Society.

[AL05b] Sudhir Agarwal and Steffen Lamparter. User Preference based Automated Selection of Web Service Compositions. In Kunal Verma; Amit Sheth; Michal Zaremba; Christoph Bussler, editor, *ICSOC Workshop on Dynamic Web Processes*, pages 1–12, Amsterdam, Netherlands, DEC 2005. IBM.

[APS05] Anupriya Ankolekar, Massimo Paolucci, and Katia P. Sycara. Towards a formal verification of owl-s process models. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *International Semantic Web Conference*, volume 3729 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 2005.

[AR05] Sudhir Agarwal and Daniel Rolli. SESAM Project Deliverable AP-2.5: Prozessmodellierung für Auktionen und Verhandlungen (Process Modeling for Auctions and Negotiations). Technical report, University of Karlsruhe (TH), December 2005.

[AS04] Sudhir Agarwal and Barbara Sprick. Access Control for Semantic Web Services. In Zhang [Zha04], pages 770–773.

[AS05] Sudhir Agarwal and Barbara Sprick. Specification of Access Control and Certification Policies for Semantic Web Services. In Kurt Bauknecht, Birgit Prll, and Hannes Werthner, editors, *6th International Conference on Electronic Commerce and Web Technologies*, volume 3590 of *LNCS*, pages 348–357, Copenhagen, Denmark, August 2005. Springer.

[AS06] Sudhir Agarwal and Rudi Studer. Automatic Matchmaking of Web Services. In Liang-Jie Zhang, editor, *IEEE 4th International Conference on Web Services*, pages 45–54, Chicago, USA, September 2006. IEEE Computer Society.

[AS07] Sudhir Agarwal and Rudi Studer. Semantic Matchmaking of Web Service Functionalities. *International Journal of Web Services Research (JWSR)[AS06]*, 2007. Invited submission for special issue on selected papers of ICWS'06. Currently in fast review track.

[ASW04] Sudhir Agarwal, Barbara Sprick, and Sandra Wortmann. Credential Based Access Control for Semantic Web Services. In Payne et al. [PDL$^+$04].

[AV05] Sudhir Agarwal and Max Voelkel. Towards more efficient software engineering with formal mda. In Evan Wallace, Jeff Z. Pan, Phil Tetlow, and Elisa F. Kendall, editors, *Proceedings of Workshop on Semantic Web Enabled Software Engineering (SWESE)*, Galway, Ireland, November 2005.

[Bai06] Tian Bai. Automatische Extraktion von Semantischen Beschreibungen von Web Services (Automatic Extraction of Semantic Description of Web Services). Student Research Project supervised by Sudhir Agarwal and Rudi Studer, October 2006.

[Bar85] Hendrik Pieter Barendregt. *The Lambda Calculus - Its Syntax and Semantics.* Elsevier, 1985.

[BCG$^+$05] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Richard Hull, and Massimo Mecella. Automatic Composition Of Transition-Based Semantic Web Services With Messaging. In Klemens Bhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, and Beng Chin Ooi Per-ke Larson, editors, *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 613–624, Trondheim, Norway, August-September 2005. ACM.

[BCM$^+$03] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory Implemenation and Applications.* Cambridge University Press, 2003.

[BFIK99] Matt Blaze, Joel Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The Role Of Trust Management In Distributed Systems Security. In *Secure Internet Programming: Issues in Distributed and Mobile Object Systems*, volume 1603 of *Lecture Notes in Computer Science*, pages 183–210. Springer Verlag, Berlin, July 1999.

[BFL96] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized Trust Management. In *Proc. of 1996 IEEE Symposium of Security and Privacy*, pages 164–173, Oakland, CA, May 1996.

[BFM06] Meghyn Bienvenu, Christian Fritz, and Sheila A. McIlraith. Planning with Qualitative Temporal Preferences. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning*, pages 134–144, Lake District of the United Kingdom, June 2006. AAAI Press.

[BH91] Franz Baader and Philipp Hanschke. A Schema for Integrating Concrete Domains into Concept Languages. In *Proceedings of the Twelfth International*

Joint Conference on Artificial Intelligence (IJCAI-91), pages 452–457, Sydney, 1991.

[BHS04] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description Logics. In Staab and Studer [SS04], pages 3–28.

[Bis03] Matt Bishop. *Computer Security – Art and Science.* Addison Wesley, 2003.

[BK02] Joachim Biskup and Yücel Karabulut. A Hybrid PKI Model With An Application For Secure Mediation. In Ehud Gudes and Sujeet Shenoi, editors, *Research Directions in Data and Applications Security, IFIP Proceedings of the 16th Annual IFIP WG 11.3 16th International Conference on Data and Application Security*, volume 256 of *IFIP Conference Proceedings*, pages 271–282. King's College, Cambridge, UK, Kluwer, July 2002.

[Bra00] Stefan Brands. *Rethinking Public Key Infrastructures and Digital Certificates.* MIT Press, Cambridge-London, 2000.

[CE82] Edmund M. Clarke and E. Allen Emerson. Design And Synthesis Of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Logic of Programs, Workshop*, pages 52–71, London, UK, 1982. Springer-Verlag.

[CEE$^+$01] Dwaine E. Clarke, Jean-Emile Elien, Carl M. Ellison, Matt Fredette, Alexander Morcos, and Ronald L. Rivest. Certificate Chain Discovery In SPKI/SDSI. *Journal of Computer Security*, 9:285–322, 2001.

[CES86] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic Verification Of Finite State Concurrent System Using Temporal Logic. *ACM Transactions on Programming Languages and Systems*, 8:244–263, 1986.

[CFL$^+$97] Yang-Hua Chu, Joan Feigenbaum, Brian LaMacchia, Paul Resnik, and Martin Strauss. REFEREE: Trust Management For Web Applications. *World Wide Web Journal*, 2(3):706–734, 1997.

[Cha85] David Chaum. Security Without Identification: Transaction Systems To Make Big Brother Obsolete. *Communication of the ACM*, 10(28):1030–1044, 1985.

[CI01] Daniel T. Chang and Sridhar Iyengar. CWM Web Services Specification - Joint Submisstion By IBM And Unisys. http://www.omg.org/docs/ad/01-10-07.pdf, October 2001.

[Den82] Dorothy E. Denning. *Cryptography and Data Security.* Addison Wesley, 1982.

[DKF+03] Grit Denker, Lalana Kagal, Tim Finin, Massimo Paolucci, and Katia Sycara. Security For DAML Web Services: Annotation and Matchmaking. In Fensel et al. [FSM03], pages 335–350.

[EA04] Andreas Eberhart and Sudhir Agarwal. SmartAPI - Associating Ontologies and APIs for Rapid Application Development. In *Ontologien in der und für die Softwaretechnik*, March 2004.

[EC80] E. Allen Emerson and Edmund M. Clarke. Characterizing Correctness Properties Of Parallel Programs Using Fixpoints. In *Proceedings of ICALP'80*, volume 85, pages 169–181, 1980.

[EF03] Andreas Eberhart and Stefan Fischer. *Web Services. Grundlagen und Praktische Umsetzung mit J2EE und .NET.* Hanser, 2003.

[EFL+99a] Carl M. Ellison, Bill Frantz, Butler Lampson, Ronald L. Rivest, Brian M. Thomas, and Tatu Ylonen. Simple public key certificate. http://world.std.com/˜cme/html/spki.html, July 1999.

[EFL+99b] Carl M. Ellison, Bill Frantz, Butler Lampson, Ronald L. Rivest, Brian M. Thomas, and Tatu Ylonen. SPKI certificate theory. Internet RFC 2693, September 1999.

[EL86] E. Allen Emerson and Chin-Laung Lei. Efficient Model Checking in Fragments of the Propositional mu-Calculus. In *Proceedings 1st IEEE LICS*, 1986.

[Eme81] E. Allen Emerson. *Branching Time Temporal Logics and the Design of Correct Concurrent Programs.* PhD thesis, Division of Applied Sciences, Harvard University, 1981.

[ESS05] Marc Ehrig, Steffen Staab, and York Sure. Bootstrapping Ontology Alignment Methods with APFEL. In Y. Gil, E. Motta, V. R. Benjamins, and M. A. Musen, editors, *Proceedings of the 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005.*, volume 3729 of *LNCS*, pages 186–200. Springer, November 2005. Nominated for best paper award!

[FB97] Warwick Ford and Michael S. Baum. *Secure Electronic Commerce: Building the Infrastructure for Digital Signatures and Encryption.* Prentice Hall, 1997.

[FB02] Dieter Fensel and Christoph Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2), 2002.

[FHBF⁺01] Warwick Ford, Phillip Hallam-Baker, Barbara Fox, Blair Dillaway, Brian LaMacchia, Jeremy Epstein, and Joe Lapp. XML Key Management Specification (XKMS). Technical report, W3C, March 2001.

[FL79] Michael J. Fischer and Richard E. Ladner. Propositional Dynamic Logic of Regular Programs. *Journal of Computer and Sytem Science*, 18:194–211, 1979.

[FSM03] Dieter Fensel, Katia Sycara, and John Mylopoulos, editors. *Proceedings of the 2nd International Semantic Web Conference*, volume 2870 of *LNCS*, Sandial Island, Fl, USA, October 2003. Springer.

[GHVD03] Benjamin Grossof, Ian Horrocks, Raphael Volz, and Stephan Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proceedings of WWW 2003*, Budapest, Hungary, May 2003.

[GMP04] Stephan Grimm, Boris Motik, and Chris Preist. Variance in e-Business Service Discovery. In *Semantic Web Services: Preparing to Meet the World of Business Applications, workshop at ISWC 2004*, 2004.

[HBEV04] Peter Haase, John Broekstra, Andreas Eberhart, and Raphael Volz. A Comparison of RDF Query Languages. In *Proceedings of the Third International Semantic Web Conference, Hiroshima, Japan, 2004.*, NOV 2004.

[HCM$^+$05] Armin Haller, Emilia Cimpian, Adrian Mocan, Eyal Oren, and Christoph Bussler. WSMX - A Semantic Service-Oriented Architecture. In *Proceedings of the International Conference on Web Service (ICWS 2005)*, pages 321–328, Orlando, Florida, 2005.

[HFPS99] Russ Housley, W. Ford, William T. Polk, and D. Solo. Internt X.509 Public Key Infrastrucure Certificate and CRL Profile. RFC 2459 Edition, January 1999. http://www.ietf.org/rfc/rfc2459.txt.

[HH02] Ian Horrocks and James A. Hendler, editors. *Proceedings of the First International Semantic Web Conference: The Semantic Web (ISWC 2002)*, volume 2342 of *Lecture Notes in Computer Science (LNCS)*, Sardinia, Italy, 2002. Springer.

[HHK$^+$05] Pascal Hitzler, Peter Haase, Markus Krötzsch, York Sure, and Rudi Studer. DLP isn't so bad after all. In *Proceedings of the Workshop OWL - Experiences and Directions, Galway, Ireland, November 2005*, NOV 2005.

[HM80] Mathew Hennessy and Robin Milner. On Observing Non-Determinism and Concurrency. In *Proceedings of ICALP'80*. LNCS, 1980.

[HM05] Peter Haase and Boris Motik. A Mapping System for the Integration of OWL-DL Ontologies. In Axel Hahn, Sven Abels, and Liane Haak, editors, *IHIS 05: Proc.s of the 1st Int. Workshop on Interoperability of Heterogeneous Information Systems*, pages 9–16. ACM Press, NOV 2005.

[HPS04a] Ian Horrocks and Peter F. Patel-Schneider. A Proposal for an OWL Rules Language. In *Proceedings of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 723–731. ACM, 2004.

[HPS04b] Ian Horrocks and Peter F. Patel-Schneider. Reducing OWL Entailment to Description Logic Satisfiability. *Journal of Web Semantics*, 1(4):345–357, 2004.

[HPSvH03] 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), 2003.

[JJ89] Claude Jard and Thierry Jron. On-line Model Checking for Finite Linear Time Spefications. In *International Workshop on Automatic Verification Methods for Finite State Systems*, number 407 in LNCS. Springer, 1989.

[Kel04] Stefan Kelm. The Pki Page – Extensive List Of Certification Authorities. http://www.pki-page.org/, 2004.

[KFD$^+$04] Lalana Kagal, Tim Finin, Grit Denker, Massimo Paolucci, Katia Sycara, and Naveen Srinivasan. Authorization and Privacy for Semantic Web Services. In Payne et al. [PDL$^+$04].

[KFJ03] Lalana Kagal, Tim Finin, and Anupam Joshi. A Policy Language for A Pervasive Computing Environment. In *IEEE 4th International Workshop on Policies for Distributed Systems and Networks*. IEEE, June 2003.

[KFJ04] Lalana Kagal, Tim Finin, and Anupam Joshi. Declarative Policies for Describing Web Service Capabilities and Constraints. In *W3C Workshop on Constraints and Capabilities for Web Services*, Oracle Conference Center, Redwood Shores, CA, USA, October 2004. W3C.

[KLP$^+$04] Uwe Keller, Ruben Lara, Axel Pollers, Ioan Toma, Michel Kifer, and Dieter Fensel. WSMO Web Service Discovery, November 2004. http://www.wsmo.org/TR/d5/d5.1/v0.1.

[Koz83] D Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[LAO$^+$06] Steffen Lamparter, Anupriya Ankolekar, Daniel Oberle, Rudi Studer, and Christof Weinhardt. A Policy Framework for Trading Configurable Goods and Services in Open Electronic Markets. In Bruce Spencer, Mark S. Fox, Weichang Du, Donglei Du, and Scott Buffett, editors, *Proceedings of the 8th Int. Conference on Electronic Commerce (ICEC'06)*, pages 162 – 173, Fredericton, New Brunswick, Canada, AUG 2006.

[Len02] Maurizio Lenzerini. Data Integration: A Theoretical Perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246. ACM Press, 2002.

[LH03] Lei Li and Ian Horrocks. A Software Framework For Matchmaking Based On Semantic Web Technology. In *Www '03: Proceedings Of The Twelfth International Conference On World Wide Web*, pages 331–339. ACM Press, 2003.

[MCK03] KIRK MCKUSICK. A Conversation with Adam Bosworth. *ACM Queue*, 1(1), MArch 2003.

[Mil80] R. Milner. *A Calculus for Communicating Processes*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

[MM04] Frank Manola and Eric Miller. Resource Description Framework (RDF) Model and Syntax Specification, February 2004. http://www.w3.org/TR/rdf-primer/.

[MP69] Zohar Manna and Amir Pnueli. Formalization of Properties of Recursively Defined Functions. In *ACM STOC*, pages 201–210, 1969.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes, Part I+II. *Journal of Information and Computation*, pages 1–87, September 1992.

[MS02] Sheila A. McIlraith and Tran Cao Son. Adapting Golog for Composition of Semantic Web Services. In *8th International Conference on Principles of Knowledge Representation and Reasoning*, Toulouse, France, April 2002.

[MSS04] B. Motik, U. Sattler, and R. Studer. Query Answering for OWL-DL with Rules. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *Proc. of the 3rd. Int. Semantic Web Conference (ISWC 2004)*, volume 3298 of *LNCS*, Hiroshima, Japan, November 2004. Springer.

[MSZ01] S. McIlraith, T.C. Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems, Special Issue on the Semantic Web*, 16(2):46–53, March/April 2001.

[NS06] Kioumars Namiri and Nenad Stojanovic. Towards Business Level Verification of Cross-Orgranizational Business Processes. In *Proceedings of the Workshop on Semantics for Business Process Management*, pages 101–112, Budva, June 2006.

[Par69] David Park. Fixpoint Induction And Proofs Of Program Properties. *Machine Intelligence*, 5:59–78, 1969.

[PDL$^+$04] Terry Payne, Keith Decker, Ora Lassila, Sheila McIlraith, and Katia Sycara, editors. *AAAI Spring Symposium - Semantic Web Services*, Stanford, California, USA, March 2004.

[PKKJ04] Anand Patwardhan, Vladimir Korolev, Lalana Kagal, and Anupam Joshi. Enforcing Policies in Pervasive Environments. In *International Conference on Mobile and Ubiquitous Systems: Networking and Services*, Cambridge, MA, August 2004. IEEE.

[PKPS02] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara. Semantic Matching of Web Service Capabilities. In Horrocks and Hendler [HH02].

[Pnu77] Amir Pnueli. A Temporal Logic of Programs. In *18th Annual IEEE-CS Symposium on Foundations of Computer Science*, pages 46–57, 1977.

[Pra76] Vaughan R. Pratt. Semantical Considerations of Floyd-Hoare logic. In *Proceedings of 16th IEEE FOCS*, pages 109–121, 1976.

[Pra82] Vaughan R. Pratt. A Decidable Mu-Calculus. In *Proceedings of 22nd IEEE FOCS*, pages 421–427, 1982.

[QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and Verificiation of Concurrent Programs in CESAR. In *Proceedings of 5th International Symposium on Programming*, number 137 in LNCS, pages 195–220. Springer, 1982.

[RA03] Daniel Rolli and Sudhir Agarwal. SESAM Project Deliverable AP-2.1: Minimales Marktmodell (Minimal Market Model). Technical report, University of Karlsruhe (TH), September 2003.

[RE05] Daniel Rolli and Andreas Eberhart. An Auction Reference Model for Describing and Running Auctions. In *Proceedings of the Wirtschaftsinformatik Conference*, Bamberg, Germany, 2005.

[Ric04] Mike Ricciuti. Will Jini-like wishes come true? http://news.com.com/2010-1071-5170275.html, March 2004.

[RKL$^+$05] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Ruben Lara, Michael Stollberg, Axel Pollers, Cristina Feier, Christoph Bussler, and Dieter Fensel. Web Service Modeling Ontology. *Applied Ontology*, 1(1):77–106, 2005.

[RL96] Ronald L. Rivest and Butler Lampson. SDSI - A Simple Distributed Security Infrastructure. http://theory.lcs.mit.edu/˜cis/sdsi.html, April 1996.

[RNW04] Daniel Rolli, Dirk Neumann, and Christoph Weinhardt. A Minimal Market Model in Ephermal Markets. In *TheFormEMC*, pages 86–100, Toledo, Spain, 2004.

[Sam02] Pierangela Samarati. Enriching Access Control To Support Credential-Based Specifications. In Bernd Reusch Sigrid Schubert and Norbert Jesse, editors, *"Informatik bewegt" Proc. of the 32. Jahrestagung der Gesellschaft für Informatik*, volume P-19 of *Lecture Notes in Informatics*, pages 114–119. German Informatics Society (GI), Dortmund, Germany, October 2002. http://ls6-www.cs.uni-dortmund.de/issi/cred_ws/index.html.de.

[SBH$^+$05] York Sure, Stephan Bloehdorn, Peter Haase, Jens Hartmann, and Daniel Oberle. The SWRC Ontology - Semantic Web for Research Communities. In Carlos Bento, Amilcar Cardoso, and Gael Dias, editors, *Proceedings of the 12th Portuguese Conference on Artificial Intelligence - Progress in Artificial Intelligence (EPIA 2005)*, volume 3803 of *LNCS*, pages 218 – 231, Covilha, Portugal, DEC 2005. Springer.

[SC01] Pierangela Samarati and Sabrina de Capitani di Vimercati. Access Control: Policies, Models, And Mechanisms. In Riccardo Focardi and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design (FOSAD)*, volume 2171 of *Lecture Notes in Computer Science*, pages 137–196. FOSAD 2000, Bertinoro, Italy, Springer Verlag, Berlin, October 2001.

[SPAS03] Katia Sycara, Massimo Paolucci, Anupriya Ankolekar, and Naveen Srinivasan. Automated Discovery, Interaction and Composition of Semantic Web Services. *Journal of Web Semantics*, 1(1):27–46, December 2003.

[SPW$^+$04] Evren Sirin, Bijan Parsia, Dan Wu, James Hendler, and Dana Nau. HTN planning for web service composition using SHOP2. *Journal of Web Semantics*, 1(4):377–396, 2004.

[SS04] Steffen Staab and Rudi Studer, editors. *Handbook on Ontologies.* International Handbooks on Information Systems. Springer, 2004.

[SSS91] Manfred Schmidt-Schauss and Gert Smolka. Attributive Concept Descriptions With Complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[SW91] Colin Stirling and David Walker. Local Model Checking in the Modal mu-Calculus. *Theoretical Compututer Science*, 89(1):161–177, 1991.

[SW01] Davide Sangiorgi and David Walker. *PI-Calculus: A Theory of Mobile Processes*. Cambridge University Press, New York, NY, USA, 2001.

[SW03] Michael Stroebel and Christof Weinhardt. The Montreal Taxonomy for Electronic Negotiations. *Group Decision and Negotiation*, 12(2):143–164, 2003.

[Tch04] Dimitri Tcherevik. Managing Web Services with CA Web Services Distributed Management. Technical report, Computer Associates, May 2004.

[Tur36] Alan M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. In *Proceedings of the London Mathematical Society*, volume 2, pages 230–265, 1936.

[UDD01] UDDI. UDDI Executive White Paper. Technical report, UDDI.org, November 2001.

[vdA01] Wil M. P. van der Aalst. Exterminating the Dynamic Change Bug: A Concrete Approach to Support Workflow Change. *Information Systems Frontiers*, 3(3):297–317, 2001.

[vdAvK02] W.M.P. van der Aalst and Hee van Kees. *Workflow Management: Models, Methods and Systems*. The MIT Press, 2002.

[VOSS03] Raphael Volz, Daniel Oberle, Steffen Staab, and Rudi Studer. Ontolift prototype. WonderWeb Deliverable D11, 2003. http://wonderweb.semanticweb.org.

[VW83] Moshe Y. Vardi and P. Wolper. Yet Another Process Logic. In *Logic of Programs*, volume 820, pages 501–512. LNCS, 1983.

[W3C02] W3C. Web service activity. http://www.w3.org/2002/ws/, 2002.

[W3C03] W3C. Web Service Description Language (WSDL) Version 1.2, March 2003. http://www.w3.org/TR/wsdl.

[W3C05] *Accepted Papers of the W3C Workshop on Rule Languages for Interoperability, 27-28 April 2005, Washington, DC, USA*, 2005. http://www.w3.org/2004/12/rules-ws/accepted.

[WVV$^+$01] Holger Wache, Thomas J. Voegele, Ubbo Visser, Heiner Stuckenschmidt, Gerhard Schuster, Holger Neumann, and Sebastian Huebner. Ontology-Based Integration of Information – A Survey of Existing Approaches. In

IJCAI-01 Workshop: Ontologies and Information Sharing, pages 108–117, 2001.

[Zha04] Liang-Jie Zhang, editor. *Proceedings of the 2nd International Conference on Web Services*, San Diego, California, USA, July 2004. IEEE Computer Society.