Jinghai Rao

Semantic Web Service Composition via Logic-based Program Synthesis

Department of Computer and Information Science
Norwegian University of Science and Technology
N-7491 Trondheim, Norway

Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer and Information Sciences
N-7491 Trondheim
Norway

Dr.Ing. Thesis 2004:121

ISBN 82-471-6464-7 (printed ver.)
ISBN 82-471-6463-9 (electronic ver.)
ISSN 1503-8181

Abstract

The ability to efficient selection and integration of inter-organizational heterogeneous Web services at runtime becomes an important requirement to the Web service provision. In an Web service application, if no single existing Web service can satisfy the functionality required by the user, there should be a program or an agent to automated combine existing services together in order to fulfill the request.

The aim of this thesis is to consider the Web service composition problem from the viewpoint of logic-based program synthesis, and to propose an agent-based framework for supporting the composition process in scalable and flexible manner. The approach described in this thesis uses Linear Logic-based theorem proving to assist and automate composition of Semantic Web services. The approach uses a Semantic Web service language (DAML-S) for external presentation of Web services, while, internally, the services are presented by extralogical axioms and proofs in Linear Logic. Linear Logic, as a resource conscious logic, enables us to capture the concurrent features of Web services formally (including parameters, states and non-functional attributes). The approach uses a process calculus to present the process model of the composite service. The process calculus is attached to the Linear Logic inference rules in the style of type theory. Thus the process model for a composite service can be generated directly from the complete proof. We introduce a set of subtyping rules that defines a valid dataflow for composite services. The subtyping rules that are used for semantic reasoning are presented with Linear Logic inference figures. The composition system has been implemented based on a multi-agent architecture, AGORA. The agent-based design enables the different components for Web service composition system, such as the theorem prover, semantic reasoner and translator to integrated to each other in a loosely coupled manner.

We conclude with discussing how this approach has been directed to meet the main challenges in Web service composition. First, it is autonomous so that the users do not required to analyze the huge amount of available services manually. Second, it has good scalability and flexibility so that the composition is better performed in a dynamic environment. Third, it solves the heterogeneous problem because the Semantic Web information is used for matching and composing Web services.

We argue that LL theorem proving, combined with semantic reasoning offers a practical approach to the success to the composition of Web services. LL, as a logic for specifying concurrent programming, provides higher expressive powers in the modeling of Web services than classical logic. Further, the agent-based design enables the different components for Web service composition system to integrated to each other in a loosely coupled manner.

The main contributions of this thesis is summarized as follows. First, an generic framework is developed for the purpose of presenting an abstract process of the automated Semantic Web service composition. Second, a specific system based on the generic platform has been developed. The system focuses on the translation between the internal and external languages together with the process extraction from the proof. Third, applications of the subtyping inference rules that are used for semantic reasoning is discussed. Fourth, an agent architecture is developed as the platform for Web service provision and composition.

Contents

List of Figures

List of Tables

Preface

This thesis is submitted to the Norwegian University of Science and Technology (NTNU) in partial fulfillment of the requirements for the degree *doktor ingeniør*. This work has been conducted at the Department of Computer and Information Sciences (IDI), NTNU, Trondheim, Norway, under the supervision of Professor Mihhail Matskin. Part of this work was conducted while I was a visiting researcher at the Department of Microelectronics and Information Technology, the Royal Institute of Technology, Stockholm, Sweden.

Acknowledgments

First and foremost, I would like to thank my supervisor Professor Mihhail Matskin for his expert guidance, constant encouragement and enduring patience during my doctoral research. I am lucky to work in his group and have been surrounded by knowledgeable and helpful co-workers. I am grateful to Peep Küngas for our wonderful cooperation and all the interesting discussions about Linear Logic. Thanks to Sobah Abbas Petersen and Amund Tveit for their discussions, information and inspiration.

During the first half of year 2003 I was given the opportunity to stay at the Department of Microelectronics and Information Technology, the Royal Institute of Technology. I would like to thank Professor Hannu Tehnunen who invited me for this visiting. I would also like to thank the colleagues there, in particular Mika Cohen and Christian Schulte for discussions and assistance. I thank my old friend Bo Cong, for helping me to find the apartment and sharing a lot of fun in Stockholm.

I would like to thank all people at IDI. Among them, my friends, Darijus, Sari, Raimundas and Wacek have provided me a lot of help. Warm thanks to my friends both in Norway and in China for the joy I shared with them and the help I received from them.

Finally, I would like to thank my parents and my brother in China for their love and support. My special thank goes to Xiaomeng for her encouragement and support. She has been helping me both with forming the ideas presented in this thesis, and with thoughtful comments on what has been written.

Jinghai Rao
November 10, 2004

Chapter 1

Introduction

1.1 Motivation and Aim

Recent progress in the field of Web services makes it practically possible to publish, locate, and invoke applications across the Web. This is a reason why more and more companies and organizations now implement their core business and outsource other application services over Internet. Thus the ability to efficient selection and integration of inter-organizational services on the Web at runtime becomes an important issue to the Web service provision. The general problem is about how to develop mechanisms to automatically locate the correct Web service in order to meet the user's requirements. In some cases, if no single Web service can satisfy the functionality required by the user, there should be a possibility to combine existing services together in order to fulfill the request.

The problem of Web service composition is a highly complex task. Here we underline some sources of its complexities:

- First, the amount of available Web services is huge, and it is already beyond the human's capability to analyze them manually.
- Second, Web services can be created and updated on the fly, thus the composition system needs to detect the updating at runtime and the decision should be made based on the up-to-date information.
- Third, the Web services are usually developed by different organizations that use different conceptual models for presenting services' characteristics. This requires utilization of relevant semantic information for matching and composition of Web services.

In this thesis, we focus on the automated composition of Web services at runtime. The result of the composite service is generated in the user interaction loop on the basis of the service requests and the available services. Since

Web services are pieces of software applications, Web service Composition can be conceived as a software synthesis problem [80]. This research trend has triggered a number of research efforts both in academia and in industry. (An overview of the literature is given in Chapter 2).

The aim of this thesis is to consider the Web service composition problem from the viewpoint of logic-based program synthesis, and to propose an agent-based framework for supporting the composition process. The logic-based method ensures the correctness and the completeness of the solution. The agent-based framework provides the scalability and flexibility to the composition system.

1.2 An Application Example

We consider how our service composition method can be applied to the composition of Web services. Here, we present an example based on value-added Web service composition. Value-added services differ from core services—they are not a part of core services but act as complements to the core services. In particular, they may stand alone in terms of operation and profitability as well as provide adds-on to core services. It is important to mention that value-added services may allow different their combinations and they may provide incremental extension of core services. For example, in online shopping, the core services range from product search, ordering, payment and shipment. The value-added services, such as currency exchange, measurement converter and language translation can also be required in cases when the core services cannot meet the users' requests exactly. Usually these value-added services are not designed for a particular core service but they rather extend abilities of core services or, in other words, add value to the core services.

As a working example we consider a ski selling Web service. In this example the core service receives characteristics of a pair of skis (such as, length, brand, model etc) as inputs and provides prices, availability and other requested characteristics as outputs. We assume that a user would like to use this service but there are gaps between the user's requirements and the functionalities the service provides.

The differences could exist, for example, in the following details:

- the user would like to receive prices in a local currency (for example, in Norwegian Crowns), however, the service provides price in US Dollars only;
- the user would like to use centimeters as length measurement units but the service uses inches;
- the user does not know what ski length or model are needed and he would like that these can be calculated from his height and weight;

- the user does not know which brand is most suitable and he would like to get a recommendation from a Web service.

We assume that the user provides the body height measured in centimeters (cm), the body weight measured in kilograms (kg), his skill level and the price limit. The user would like to get a price of a recommended pair of skis in Norwegian Crowns (NOK).

The core service *selectSkis* accepts the ski length measured in inches, ski brand, ski model and gives the ski price in US Dollars (USD).

Some available value-added services are as follows :

- *selectBrand*—given a price limit and a skill level, provides a brand;
- *selectModel*—given body height in cm and body weight in kg, provides ski length in cm and a model;
- *cm2inch*—given length in cm, provides length in inches;
- *USD2NOK*—given price in USD, provides price in NOK;
- *inch2cm*—given length in inches, provides length in cm;
- *NOK2USD*—given price in NOK, provides price in USD;
- *kg2lb*—given weight in kg, provides weight in pounds;
- *lb2kg*—given weight in pounds, provides weight in kg.

The core service and available value-added services are depicted respectively in Figure 1.1 and Figure 1.2. A required service is presented in Figure 1.3. The structure of a composite service solution is presented in Figure 1.4.

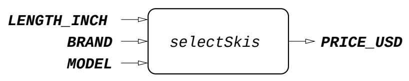


Figure 1.1: The core service for buying skis.

We would like to mention that our working example is intentionally made simpler than it is required for practical cases. This has been done in order to keep simplicity of presentation. In practice there can be more value-added services available and more parameters for the core service (in particular, there may exist many other converters for currency, measurements and other units), and the user may not always be able to find a solution intuitively. In addition, it may also be beyond the user's ability to search among a huge amount of available value-added services to find all possible solutions. In particular, if the set of possible solutions consists of all existing converters to all inputs

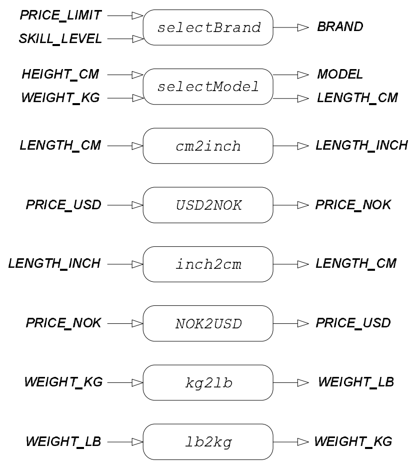


Figure 1.2: The available value-added services.

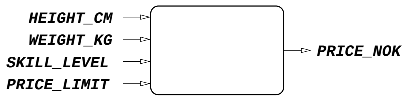


Figure 1.3: The required service for buying skis.

and outputs of all Web services (both core and value-added), this may cause big overhead in service provision. Taking this into account, we believe that automatic composition would be an efficient and a practical solution in this case.

Here we illustrate a case where the Web services are only specified by the input/output signatures. This is not enough to model the current Web services because the Web services have more behavior other than just receiving and sending messages, for example, the pre- and post-condition, the cost information, the security information, etc. We will give a complete overview of the properties to specify the Web Services in Chapter 3.

1.3 Research Questions

The overall research question this thesis tries to answer is:

How can we enable the intelligent agents to automatically retrieve and compose Web services to achieve the goals specified by their users?

In order to be able to answer this question, we define a set of research questions that address the problem in detail.

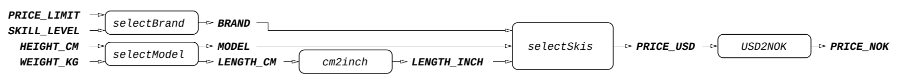


Figure 1.4: The final service structure for buying skis.

RQ1: What is Web services and how are they composed?

- What type of information is required to identify Web services at runtime?
- How do we specify the interaction within the composite service?
- What platforms are required to support the publishing, invocation and composition of Web services?

RQ2: How can we automatically compose the Web services via logic-based program synthesis?

- How can we specify the Web services in a logical language?
- What is the formal basis for representing the dynamic composition of services from the service specification?
- How can we extract the process model from the proof?

RQ3: Is it reasonable to present the Web service composition problem in the context of multi-agent framework?

- Can multi-agent framework improve the scalability and flexibility of the Web service composition systems?
- Can multi-agent framework leverage the semantics interoperation for heterogeneous Web services?

RQ4: How can we use Semantic Web markups for facilitating the Web service composition task?

- How can we deal with the difference between the Semantic Web languages and Web service languages?
- How can we integrate the semantic reasoner for Semantic Web with the logic theorem prover for program synthesis?

1.4 Background

According to the general research question, we are interested in the problem of how to automate the service provision and composition at runtime. When we chose this subject, four research directions inspired us. They are the researches in Web services, deductive program synthesis, software agents and Semantic Web. Those directions are presented here as the background of the research subject in this thesis.

- First of all, we were inspired by the Web service initiatives proposed both in academia and in industry. Those efforts provide platforms and languages that allow to discover, execute and integrate services. For example, Universal Description, Discovery, and Integration (UDDI) [19], Web Services Description Language (WSDL) [31], Simple Object Access Protocol (SOAP) [24] and parts of DAML-S [72] ontology (including ServiceProfile and ServiceGrounding) define standard ways for service discovery, description and invocation (message passing). Some other initiatives including Business Process Execution Language for Web service (BPEL4WS) [13] and DAML-S ServiceModel, are focused on representing service compositions where a process flow and bindings between services are known a priori. Some platforms have been developed to support the invocation of Web services that are specified by the above languages, such as bpws4j [57], axis [1], WebSphere [58] and .NET [84]. However, those initiatives do not provide means for the dynamic composition of existing services. If the required functionality cannot be realized by a single existing service, the composition of the available services to fulfill the request has to be specified by the human being by hand. The languages only provide the way to describe the the composite services when the process model is known.

- Second, we were inspired by research on deductive program synthesis using the intuitionistic propositional calculus. Program synthesis is a method of software engineering in order to generate programs automatically. In particular, deductive program synthesis is based on an observation that constructive proofs are equivalent to programs where each step of a proof can be interpreted as a step of a computation. The key ideas of the software composition approach, as well as correspondence between theorems and specifications and between constructive proofs and programs, are presented in [70]. We believe that an automated deductive program synthesis method serves best for dynamic Web Service composition given the correct specification of services in logic. The open problem is how to develop the translators between the standard Web service specification languages and the logical languages. The Web service specification languages are used by the users to describe the Web services in the problem domain. The logical languages are hidden from the users, although they are reachable, if needed. They are used by automated theorem prover to generate the process model for the composite service.

- The third source for our approach is agent technology [124,93]. The software agent is autonomous, reactive and proactive computational device that could interact through Internet. Using software agents to represent Web services gives a better flexibility, scalability and dynamicity to retrieval and composition problem. The reasons exist in three points. First, if the Web services are changed on the fly, the agents who represent those services is able to report the changes to the repository initiatively. Second, the distributed manner of software agent does not require a central controlled service repository. This can improve the system performance when the number of available Web Services is huge. Third, a composition program can be divided into a set of sub-programs that can be solved by different agents.

- At last, we were inspired by the research efforts in the Semantic Web [20]. The Semantic Web, as a vision of the evolution of the World Wide Web from a human interpretable document collection to a computer understandable one, increases automation of interoperation among Web applications. Generally, such interoperation is realized through the content annotation using XML based ontology language, such as DAML+OIL [2] and OWL [34]. The Semantic Web and ontologies are central to our research because they allow the Web services and software agent to agree on the terms that they use when advertising, searching and communicating. For Web Services, the semantic markup provide the shared meaning of the term using to describe the service capabilities. In particular, the synonym and subsumption relations between the terms are essential for the Web service composition, because they define the valid dataflow between the Web services. The same issue is also emphasized by the program synthesis community using type systems [69]. For the software agents, agents represent their ”view of the world” by explicitly defined ontologies. The interoperability of a multi-agent system is achieved through the reconciliation of these views by a commitment to common ontologies that permit agents to interoperate and cooperate. Thus, ontologies, if shared among stakeholders, will improve system interoperability across agent systems.

We can find a strong influence from the above research directions to our research result. An overview of the background researches is illustrated in Figure 1.5. In general, the Web Service efforts provide the specification language to the program synthesis methods that use logical languages to deduce and generate process model for composite service. The software agent, using program synthesis algorithm as decision making and inference method, acts as a more flexible and scalable platform for composition system. In the platform, the software agents are on behalf of Web services and the Web services present the requirements and capabilities of agents. Semantic Web supports

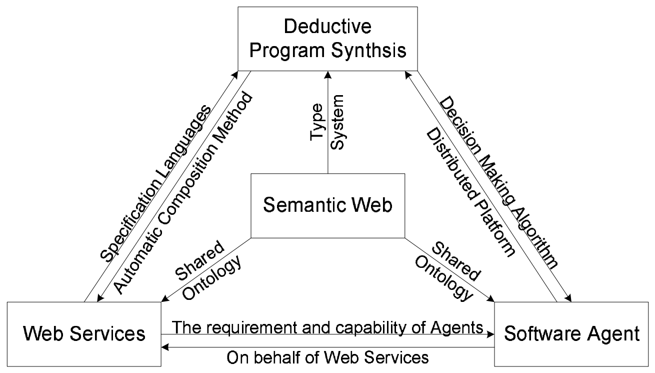

Deductive Program Synthesis
Specification Languages
Automatic Composition Method
Type System
Decision Making Algorithm
Distributed Platform
Semantic Web
Shared Ontology
Shared Ontology
Web Services
The requirement and capability of Agents
On behalf of Web Services
Software Agent

Figure 1.5: The relations between the background researches.

the interoperation of Web services and multiple agents by shared ontologies, while it also provides the type system for program synthesis.

Our research method integrates a deductive program synthesis technique with the standard semantic Web service specification language to provide automated service composition. The work is realized in the context of a multi-agent platform. The detail of our proposed solution is described in Section 1.5.

1.5 Proposed Solution

We have approached the problem through designing and implementing a prototype system for automated Web service composition. The prototype constitutes an essential part of the presented thesis work. The prototype is designed to combine the advantages of multi-agent system with positive sides of deductive program synthesis for supporting Web Service composition. The program synthesis part is based on the proof search in a fragment of propositional Linear Logic (LL) [44]. The prototype system is built on the software agent architecture, AGORA. We use both the AGORA notion and implementation in our solution.

This thesis has resulted in the implemented prototype with the following functionalities:

- First, the general idea of our service composition method is as follows. Given a set of existing Web services and a set of functionalities and non-functional attributes, the method finds a composition of existing services that satisfies the user requirements. The description of existing Web services is encoded as extralogical axioms in LL, and the requirements to the composite services are specified in form of a LL sequent to be proven. We use a LL theorem prover to determine whether the sequent can be proven by the axioms. If the answer is positive then the next step is to construct the dataflow from the generated proofs. The reason that we use LL theorem proving as Web service composition method is as follows. LL is a resource-conscious logic and this allows us to capture the concurrent features of Web services formally (including parameters, states and non-functional attributes). Because of soundness of the logic fragment used the correctness of composite services is guaranteed with respect to the initial specification. Completeness of the logic fragment ensures that all composable solutions would be found. Our method is strongly influenced by the service composition method using Structural Synthesis of Programs(SSP) proposed in [76, 64, 75]. Both SSP and our approach are based on the idea that the programs can be constructed automatically taking into account only their structural properties, namely the interfaces of the services. The idea is used to construct programs from existing modules. Modules are pre-programmed functions whose internal structure are not considered. Therefore, the result composite service can be modeled as a set of component services that are connected through dataflow dependencies between the interfaces. One main difference is that the method reported in this thesis considers more properties other than the input/output parameters, for example the pre-, post-conditions and the non-functional attributes.

- Second, our method distinguishes between the service specification languages and the logical languages when describing the Web services. The service specification languages are used by the users to enhance accessibility of the users in the sense that it is easier for the users to express what they want. The logical languages are different because they are used by the automated theorem prover. In reality, the users have already gotten used to the standard Web service languages, such as WSDL and DAML-S. Thus we have to develop the translation components between the Web service languages and the logical languages. In our system, we have presented that DAML-S ServiceProfile part is suitable for a declarative language to describe the Semantic Web service specification. The documents written in DAML-S ServiceProfile is able to be translated to LL axioms as the input of the LL theorem prover. A π-calculus [90] based process calculus is used to present the process model of composite service formally. It can be extracted from the proof directly. The calculus can be translated into DAML-S ServiceModel. In addition, because DAML-S lacks the support of invocation platform, WSDL and BPEL4WS are used as grounding of ServiceProfile and ServiceModel respectively. As a result, the translation between the WSDL, BPEL4WS and DAML-S is also developed.

- Third, we use a composition approach that allows reasoning with types from a service specification. The fundamental purpose of the types is to define the valid dataflow for the composite services, thus types impose constraints which help to enforce the correctness of the execution of composite service. The typing relations between the concepts used to specify the services are defined in the domain ontology. We developed a semantic reasoner to infer the missing type information automatically. The type information can be presented in form of LL implications. This ensures the interoperability between the LL theorem prover and the semantic reasoner. A set of interaction protocols between the LL theorem prover and the semantic reasoner is also developed.

- Fourth, we use an agent-based approach to support the cooperative activities, such as advertising, searching and composing Web services. The agent-based approach uses software agents to represent the service providers and the service requesters. In this approach, the Web services describe the requirements and capabilities of agents. The service composition process is presented as multi-agent communication, coordination and negotiation. Furthermore, the different components supporting the composition, such as the theorem provers, the semantic reasoners and the translators can also represented by software agents, which is called facilitator agents. The agent-based design enables these components to integrate with each other in a loosely coupled manner. We have designed and implemented the respective agents on an agent architecture, AGORA [74, 73]. The AGORA system is a multi-agent environment, which provides supports for cooperative work between the participating agents. The agent-based approach gives service providers a more proactive role in the service composition process. The architecture combining with a distributed Web service composition method, reduces the heterogeneity between different components.

- Finally, we take advantage of the existing LL programming languages as the automated theorem provers in our experiments. We have used both Forum [87] and RAPS [60]. Forum is a LL theorem prover based on intuitionistic LL. We have implemented a theorem prover based on its Prolog interpreter to enable the attachment of process model as the type of the propositional variables. RAPS is developed in our group as a LL planner to support reasoning over Web service composition problems both in propositional and first-order LL. It supports Partial Deduction but does not support multiplicative disjunction so far. We should emphasize that this thesis is not an effort to develop automated LL theorem provers, instead we take advantage of existing languages and systems for support our applications in Web service composition.

1.6 Research Activities and Contributions

The research activities used in this work consist of literature review, system analysis, development and evaluation. All together the phases include the following steps.

1. The *literature review* step includes:
   - The *survey of Web service languages and platforms* step includes an investigation of existing languages and platforms to support the description, provision and invocation of Web services.
   - The *survey of Web service composition* step includes an investigation of applicable part of the relevant research efforts, in particular, we concern the automated Web service composition methods based on AI planning or logic-based program synthesis.
2. The *analysis of requirements* step includes an inventory of the problems in Web service composition and an analysis of the raised requirements.
3. The *development* step includes:
   - The *development of upper ontologies of Web services* step includes the efforts to generalize the specification languages both for service profile and service process. The languages are based on DAML-S, but a reconstruction is needed when considering the translation to logical languages.
   - The *development of logical presentation of Web services* step includes the definition of the specific axioms and inference rules.
   - The *development of agent-based approach* step includes the design of agent architecture and interaction protocols to support the service composition based on the multi-agent platform, AGORA.
   - The *prototype application* step includes development and implementation of the prototypical environment.
4. The *applicability analysis and evaluation* step includes the experimental evaluation of using proposed Web service composition in examples.

A major contribution of this thesis is the development and specification of a formal approach to support the Semantic Web service composition problem. The work has been implemented on a multi-agent platform.

Some specific contributions of our work are listed as follows. The detail evaluation of the following contributions can be found in Section 8.2:

C1: Development and evaluation of a LL-based formal method for automated program synthesis to enable Semantic Web service composition.

C2: Development of the translation mechanism between the standard Web service languages and the LL formulae. The process calculus is attached to the proof figure as type system, thus the process can be extracted from the proof directly.

C3: Application of the subtyping inference rules that are used for semantic reasoning. The semantic relations are presented with LL inference figures. We propose a system architecture where the semantic parser, LL theorem prover and the semantic reasoner can operate together.

C4: Development of an agent architecture and interaction for the service provision and composition.

1.7 Thesis Outline

This thesis contains a survey of the Web service composition methods, a description of the work and an overview of the results and contributions. An outline of the structure of this thesis is shown as follows:

Chapter 2 provides an overview of literatures on Web service composition, including the composition languages, platforms and methods. The composition methods are separated by the topic of business process, software composition and AI planning.

Chapter 3 presents a method for the translation from DAML-S ServiceProfile to extralogical axioms in LL.

Chapter 4 discusses how to extract a process from the LL proof. We introduce a process calculus that is attached to the LL inference rules in the style of type theory. Thus the process model for the composite service can be generated from the process of the proof.

Chapter 5 presents the usage of type system to enable the composition using Semantic Web information. We used a set of subtyping rules for semantic reasoning. The rules are presented with LL inference figures so that the semantic reasoner and LL theorem prover can interact directly.

Chapter 6 introduces an agent-based approach to service composition problem, especially the agent architecture and multi-agent interaction.

Chapter 7 demonstrates the implementation issues for our service composition system.

Chapter 8 evaluates the expressive power for the fragment of LL we used to model the Web services. Discussion and comparition of the proposed method are presented in this Chapter too.

Chapter 9 summarizes and discusses the method and future directions for this work.

Chapter 2

Web Service Composition: State of the Art

2.1 Web Services: Standards and Related Technologies

The term "Web services" has been used very often nowadays. According to IBM [4], "*Web Services are self-contained, modular applications, accessible via the Web through open standard languages, which provide a set of functionalities to businesses or individuals*". This definition places the emphasis on two points. The first point is that a Web service is seen as an application accessible to other applications over the Web. Secondly, Web services are open, which means that services have published interfaces that can be invoked by message passing standards. This definition is very simple, but not precise enough. For instance, it is not clear what it is meant by a modular, self-contained application. A step further in refining the definition of Web service is the one provided by the World Wide Web consortium (W3C):

> A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML-based messages conveyed by Internet protocols. [121]

The W3C definition stresses that Web services should be capable of being "defined, described, and discovered," thereby clarifying how to access the Web services [9]. We should also emphasize that Web services do not merely provide static information, but allow one to affect some action or change in the world, e.g. the sale of a product, the control of a physical device, etc.

What makes the Web services attractive is the ability to integrate the Web services developed by different organizations together to fulfill the user's requirement. Such integration is based on the common standards of Web service interfaces, regardless of the languages that are used to implement the Web services, and the platforms where the Web services are executed.

In general, the Web services have the following features that make them better in integration inside the heterogeneous environments:

loosely coupled: In software development, coupling typically refers to the degree to which software components/modules depend upon each other. Comparing with the tightly coupled components (such as the Distributed Component Object Model (DCOM) [85] or the Common Object Request Broker Architecture (CORBA) [94]), the Web services are autonomous and can operate independently from one another. The loosely coupled feature enables Web services to locate and communicate with each other dynamically at runtime.

universal accessibility: The Web services can be defined, described and discovered through the Web that enables an easy accessibility. Not only the Web services users can locate appropriate services, but services can describe and advertise themselves so that they are possible to bind and interact with each other.

standards languages: Web services are described by standard XML languages that have been considered as parts of the Web technology. The Web services standards are of higher abstraction. Although the cores of Web services may implemented by different programming languages, the interface of Web services are described by uniform standard XML languages.

The standard may be of the most importance for the success of Web services. In fact, the Web service community has proposed dozens of standard languages and frameworks to help users to present the services in a uniformed matter. Figure 2.1 is adapted from [118], which provides an overview of commonly used standards and their positions in the application of Web services. In the following, we will elaborate the languages from three sources. First, we sketch the Web service languages proposed or co-proposed by IBM, which is considered as the most elaborated and the best described industry quasi-standard for Web service so far. The IBM Web service languages represent a traditional view of Web service languages. From this view, Web services have simple rather than rich descriptions, the environment is closed rather than open, the service requester is human rather than machine and data exchange are syntactic rather than semantic [113]. Second, we introduce a Semantic Web description of services, which provides some additional modifications to move the traditional languages closer to the vision of an openness and interoperability. We argue that the current description of DAML-S, the DAML+OIL based

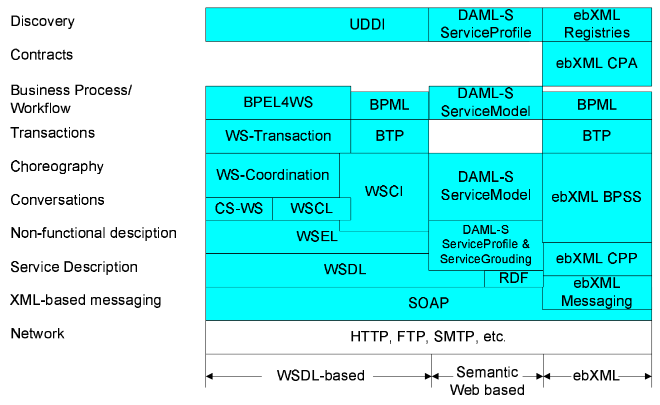


Figure 2.1: An overview of Web services standard languages, adapted from [118].

Semantic Web service ontology is of such kind of language. The third source of Web service languages are the languages concerning the business interaction, represented by ebXML. Finally, we will also list some Web service platforms that support the languages we have introduced.

2.1.1 IBM Web Service Languages

The general idea of Web service application from IBM is the following: A Web service provider offers services on the Web. He may choose to register his service at an online registry of a service broker. The registry also provides standardized description facilities, e.g. taxonomies that allow the description of 1) the functionality of a service, 2) the information about service provider, and 3) the way to access and interact with the service. The corresponding information about a particular service is registered by the provider at a broker.

The service requester searches for a service at the registry. He finds one by browsing or querying the registry. Then he uses the service description to create a binding for his application. The last step is to invoke or interact with the Web service using standard communication language. The IBM Web service languages that support the above procedure are UDDI, WSDL and SOAP.

UDDI [19] provides a registry where service providers can register and publish their services. The registry consists of three parts: white pages, yellow pages and green pages. Contact information, human readable information and related can be registered in the white pages. Keywords that characterize the service are registered in the yellow pages. Service rules and descriptions for application invocations are registered in the green pages (technical). UDDI does not support semantic descriptions of services and no content language for advertising is provided. WSDL is a candidate for such a content language.

WSDL [31] is a proposed W3C standard for describing network services. The descriptions work as a recipe for automating the details involved in communication between applications. WSDL defines an XML grammar where services are a set of ports (network endpoints). Messages are abstract descriptions of the exchanged data. Port types are abstract collections of operations supported by communication endpoints. A concrete protocol and data format for a port type constitutes to a reusable binding. SOAP is typically used to deploy the operations.

SOAP [24] is a proposed W3C standard for exchanging information in a decentralized and distributed environment. It offers a mechanism to define application semantics. This makes it possible to invoke methods on objects remotely. SOAP consists of three parts: envelope, encoding rules and Remote Procedure Call (RPC) representation. The envelope sets up a framework for what the message contains and responsibility. The encoding rules provide a serialization mechanism for exchanging instances of application specific data types. Finally, RPC makes it possible to encapsulate and represent remote procedure calls and responses.

2.1.2 Semantic Web Service Description with DAML-S

DAML-S [14] is an ontology that provides constructs for describing Web services. With DAML-S the properties of services and their capabilities are less ambiguous and described in computer-interpretable format. It can be regarded as a semantic-based effort for service description, service publication and service flow [113]. DAML-S descriptions enable improved matching of services [97]. The top level of the ontology has a service construct, related to three basic concepts: ServiceProfile, ServiceModel and ServiceGrounding. The ServiceProfile describes what the service does (what is provided), the ServiceModel describes how it works (what happens) and the ServiceGrounding describes how to access it (how to use). In a more detailed perspective, a composite Web service can be viewed as a process, which is specified by a subclass of ServiceModel called process ontology. A process can have a number of participating services as well as the control flow and data flow among the services. In the current version, the ServiceGrounding is defined as the mapping between the ServiceModel and the a service's interface description, WSDL.

2.1.3 ebXML

ebXML [38] is a standardization effort for business interactions, and is an initiative of UN/CEFACT and OASIS . It can be regarded as a more open and flexible successor of EDI (Electronic Data Interchange). To conduct business with each other, companies must do the following:

- Discover products and services offered.
- Determine how these products and services can be obtained by deciding on shared process and information exchange.
- Agree on form of communication and points of contact for exchange of documents.
- Agree on contractual terms like transactions, plans and security.

The ebXML Registry is the central server that stores the necessary data. Each company must register its profile, Collaboration Protocol Profile (CPP), which specifies some of the business processes of its business and some supported Business Service Interfaces. Business Processes are the activities companies can be involved in and the service interfaces describe how it can perform the transactions necessary in its processes. Companies use the collaboration protocol profile to agree on the contractual terms and establish a Collaboration Protocol Agreement (CPA). ebXML has two views to describe business interactions: Business Operational View and Functional Service View. The former addresses the semantics of business data transactions, in addition to: operational conventions, agreements, mutual obligations and requirements. The Functional Service View deals with supporting services like functional capabilities, business service interfaces and protocols.

ebXML can be regarded as complementary to the other technologies [110]. It takes a broader approach in terms of scope and functionality. A survey of ebXML specifications and other content standards are given in [36]. Such standards are highly relevant for semantic interoperability in a global sense, but will not be emphasized in this thesis.

2.1.4 Platforms

E-business infrastructure companies are beginning to announce platforms to support some levels of Web-service automation. Examples of such products include Hewlett-Packard's e-speak, a description, registration, and dynamic discovery platform for e-services; Microsoft's .NET and BizTalk tools; Oracle's Dynamic Services Framework; IBM's Application Framework for E-Business; and Sun's Open Network Environment. VerticalNet solutions, anticipating and wishing to accelerate the markup of services for discovery, is building ontologies and tools to organize and customize Web service discovery and - with its OSM Platform - is delivering an infrastructure that coordinates Web services for public and private trading exchanges.

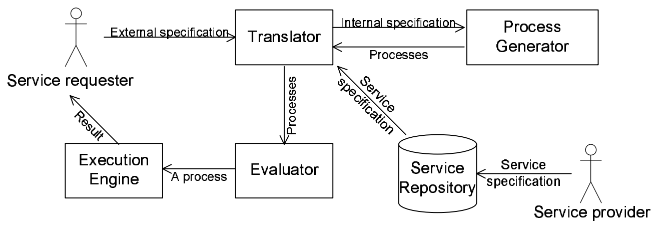


Figure 2.2: A framework of service composition system.

There are also some platforms proposed by academia. [109] is a service matching and invocation platform supported by software agents. [29] is a service composition platform based on HP workflow infrastructure.

2.2 An Abstract Model for Web Service Composition

Here, we propose a general framework for automatic Web services composition. This framework is in high-level abstraction, without considering a particular language, platform or method used in composition process. The framework is initially proposed in [105], and further improved in [106]. The aim of the framework is to give the basis to discuss similarities and differences of the available service composition methods. In addition, we also use the framework to unify the terms used in the following of the thesis.

A general framework of the service composition system is illustrated in Figure 2.2. The composition system has two kinds of participants, service provider and service requester. The service providers propose Web services for use. The service requesters consume information or services offered by service providers. The system also contains the following components: translator, process generator, evaluator, execution engine and service repository. The translator translates between the external languages used by the participants and the internal languages used by the process generator. For each request, the process generator tries to generate a plan that composes the available services in the service repository to fulfill the request. If more than one plan is found, the evaluator evaluates all plans and proposes the best one for execution. The execution engine executes the plan and returns the result to the service provider.

Most precisely, the process of automatic service composition includes the following phases:

Presentation of single service: firstly, the service providers will advertise their atomic services at a global market place. There are several languages available for advertising, for example, UDDI [19] or DAML-S ServiceProfile [72]. The essential attributes to describe a Web service include the signature, states and the non-functional values. The signature is represented by the service's inputs, outputs and exceptions. It provides information about the data transformation during the execution of a Web service. The states are specified by precondition and postcondition. We model it as the transformation from one state to another state in the world. Non-functionality values are those attributes that are used for evaluating the services, such as the description of cost, quality and response time.

Translation of the languages: most service composition methods distinguish between the external and internal service specification languages. The external languages are used by the users to enhance accessibility of the users in the sense that the users can express what they can offer or what they want in a relatively easy manner. They are usually different from the internal ones that are used by the composition process generator, because the process generator requires more formal and precise languages, for example, the logical programming languages. So far, the users have already get used to the standard Web service languages, such as WSDL and DAML-S. Thus the translation components between the standard Web service languages and the internal languages have to be developed.

Generation of composition process model: in the meantime, the service requester can also express the requirement in a service specification language. A process generator then tries to solve the requirement by composing the atomic services advertised by the service providers. The process generator usually takes the functionalities of services as input, and produces as output the process model that describes the composite service. The process model contains a set of selected atomic services together with the control flow and data flow among them.

Evaluation of composite service: it is quite common that many services have the same or similar functionalities. So it is possible that the planer generates more than one composite services fulfilling the requirement. In that case, the composite services are evaluated by their overall utilities using the information provided from the non-functional attributes. The most commonly used method is utility functions. The requester should specify weights to each non-functionality attributes and the best composite service is the one who is ranked on top.

Execution of composite service: after a unique composite process is selected, the composite service is ready to be executed. Execution of a composite Web service can be thought as a sequence of message passing according to the process model. The dataflow of the composite service is defined as the actions that the output data of a former executed service transfers to the input of a later executed atomic service.

Building composite Web services with an automated or semi-automated tool is critical to the success of the Web service applications. To that end, several methods for this purpose have been proposed. In particular, most researches conducted fall in the realm of workflow composition or AI planning.

For the former, one can argue that, in many ways, a composite service is similar to a workflow [30]. The definition of a composite service includes a set of atomic services together with the control and data flow among the services. Similarly, a workflow has to specify the flow of work items. The current achievements on flexible workflow, automatic process adaption and cross-enterprise integration provide the means for automated Web services composition as well. In addition, the dynamic workflow methods provide the means to bind the abstract nodes with the concrete resources or services automatically.

On the other hand, dynamic composition methods are required to generate the plan automatically. Most methods in such category are related to AI planning and deductive theorem proving. The general assumption of such kind of methods is that each Web service can be specified by its preconditions and effects in the planning context. Firstly, a Web service is a software component that takes the input data and produces the output data. Thus the preconditions and effects are the input and the output parameters of the service respectively. Secondly, the Web service also alters the states of the world after its execution. So the world state pre-required for the service execution is the precondition, and new states are generated after the execution is the effect. A typical example is a service for logging into a Web site. The input information is the username and password, and the output is a confirmation message. After the execution, the world state changes from "not_logged_in" to "logged_in". The "logged_in" state will be keeping until the "log_out" service is invoked. If the user can specify the preconditions and effects required by the composite service, a plan or process is generated automatically by logical theorem prover or AI planners without knowledge of predefined workflow. However, the business logic can provide constraints in the planning process.

In the following we will give a survey on the methods used for the process generator to generate the process. The methods can be either fully automated or semi-automated.

2.3 Web Service Composition via Business Process

In the workflow-based composition methods, we should distinguish the static and dynamic workflow generation. The static one means that the requester should build an abstract process model before the composition planning starts. The abstract process model includes a set of tasks and their data dependency.

Each task contains a query clause that is used to search the real atomic Web services to fulfill the task. In that case, only the selection and binding of atomic Web service are done automatically by program. The most commonly used static method is to specify the process model in graph. On the other hand, the dynamic composition both creates process model and selects atomic services automatically. This requires the requester to specify several constraints, including the dependency of atomic services, the user's preference and so on.

EFlow [29] is a platform for the specification, enactment and management of composite services. A composite service is modeled by a graph that defines the order of execution among the nodes in the process. The graph is created manually but it can be updated dynamically. The graph may include service, decision and event nodes. Service nodes represent the invocation of an atomic or composite service, decision nodes specify the alternatives and rules controlling the execution flow, and event nodes enable service processes to send and receive several types of events. Arcs in the graph denote the execution dependency among the nodes. Although the graph should be specified manually, EFlow provides the automation to bind the nodes with concrete services. The definition of a service node contains a search recipe that can be used to query actual service either at process instantiation time or at runtime. As the service node is started, the search recipe is executed, returning a reference to a specific service. In particular, the search recipe is resolved each time when a service node is activated. They do so because the availability of services may change very frequently in a highly dynamic environment. In [30], the authors further refine the service composition platform and propose a prototype of composite service definition language(CSDL). An interesting feature of CSDL is that it distinguishes between invocation of services and of operations within a service. It provides the adaptive and dynamic features to cope with the rapidly evolving business and IT environment in which Web services are executed.

Yang et. al. [126] proposed a composition logic which dictated how the component services could be combined, synchronized and co-ordinated. Composition logic is beyond conversation logic (which is modeled as a sequence of interactions between two services) and forms a sound basis for expressing the business logic that underlies business applications. The composition logic indicates the following features of a composite service:

- the sequence of atomic web services execution;
- the message dependency among the parameters of the atomic services.

The composition logic can be specified in XML document.

Tut et al. [119] introduced the use of patterns during the planning stage of service composition. Patterns represent a proven way of doing something. They could be business patterns such as how to model online store-fronts, or generic patterns such as project work process.

Polymorphic Process Model (PPM) [107] uses a method that combines the static and dynamic service composition. The static setting is supported by reference process-based multi-enterprise processes, the processes that consist of abstract subprocesses, i.e., subprocesses that have functionality description but lack implementation. The abstract subprocesses are implemented by service and binded at runtime. This is similar to the service binding in EFlow. The dynamic part of PPM is supported by service-based processes. Here, a service is modeled by a state machine that specifies that possible states of a service and their transitions. Transitions are caused by service operation(also called service activity) invocations or internal service transitions. In the setting, the dynamic service composition is enabled by the reasoning based on state machine.

2.4 Web Service Composition via AI Planning

Many research efforts tackling Web service composition problem via AI planning have been reported. In general, a planning problem can be described as a five-tuple $\langle S, S_0, G, A, \Gamma \rangle$, where S is the set of all possible states of the world, $S_0 \subset S$ denotes the initial state of the world, $G \subset S$ denotes the goal state of the world the planning system attempts to reach, A is the set of actions the planner can perform in attempting to change one state to another state in the world, and the translation relation $\Gamma \subseteq S \times A \times S$ defines the precondition and effects for the execution of each action.

In the terms of Web services, S_0 and G are the initial states and the goal states specified in the requirement of Web service requesters. A is a set of available services. Γ further denotes the state change function of each service.

DAML-S (also called OWL-S in the most recent versions) is the only Web service language that announces the directly connection with AI planning. The state change produced by the execution of the service is specified through the precondition and effect properties of the ServiceProfile in DAML-S. Precondition presents logical conditions that should be satisfied prior to the service being requested. Effects are the result of the successful execution of a service. Since DAML+OIL, the language used to build DAML-S, uses Description Logics [45] as its logical foundation, DAML+OIL has the express power allowing for logical expressions. The majority of the methods reported in this survey use DAML-S as the external Web service description language. There are also a couple of methods that use WSDL or their own languages.

In the following we introduces a list of Web service composition methods based on AI planning. This kind of methods have been reported frequently in recent years, so we do not claim that we have an exhaustive list of the methods. We further classify the methods into five categories, namely, the situation calculus, the Planning Domain Definition Language (PDDL), rule-based planning, the theorem proving and others.

2.4.1 Situation Calculus

McIlraith et. al. [81,92,80] adapt and extend the Golog language for automatic construction of Web services. Golog is a logic programming language built on top of the situation calculus. The authors address the Web service composition problem through the provision of high-level generic procedures and customizing constraints. Golog is adopted as a natural formalism for representing and reasoning about this problem.

The general idea of this method is that software agents could reason about Web services to perform automatic Web service discovery, execution, composition and inter-operation. The user request (generic procedure) and constraints can be presented by the first-order language of the situation calculus(a logical language for reasoning about action and change). The authors conceive each web service as an action - either a *PrimitiveAction* or a *ComplexAction*. Primitive actions are conceived as either world-altering actions that change the state of the world or information-gathering actions that change the agent's state of knowledge. Complex actions are compositions of individual actions. The agent knowledge base provides a logical encoding of the preconditions and effects of the Web service actions in the language of the situation calculus. The agents use procedural programming language constructs composed with concepts defined for the services and constraints using deductive machinery. A composite service is a set of atomic services which connected by procedural programming language constructs(if-then-else, while and so forth).

The authors also propose a way to customize Golog programs by incorporating the service requester's constraints. For example, the service requester can use the nondeterministic choice to present which action is selected in a given situation, or use the sequence construct to enforce the execution order between two actions. The generation of the plan have to obey the predefined constraint.

2.4.2 Planning Domain Definition Language (PDDL)

A strong interest to Web service composition from AI planning community could be explained roughly by similarity between DAML-S and PDDL representations. PDDL is widely recognized as a standardized input for state-of-the-art planners. Moreover, since DAML-S has been strongly influenced by PDDL language, mapping from one representation to another is straightforward (as long as only declarative information is considered). When planning for service composition is needed, DAML-S descriptions could be translated to PDDL format. Then different planners could be exploited for further service synthesis.

In presenting the Web service composition method based on PDDL, McDermott [77] introduces a new type of knowledge, called *value of an action*, which persists and which is not treated as a truth literal. From Web service construction perspective, the feature enables us to distinguish the information transformation and the state change produced by the execution of the service. The information, which is presented by the input/output parameters are thought to be reusable, thus the data values can be reused for the execution of multiple services. Contrarily, the states of the world are changed by the service execution. We interpret the change as that the old states are consumed and the new states are produced.

To deal with this issue is critical for Web service composition using AI planning because usually in AI planning, closed world assumption is made, meaning that if a literal does not exist in the current world, its truth value is considered to be *false*. In logic programming this approach is called *negation as failure*. The main trouble with the closed world assumption, from Web service construction perspectives, is that merely with truth literals we cannot express that new information has been acquired. For instance, one service requester might want to describe that after sending a message to a Web service, an identity number to the message will be generated. Thus during later communication the ID could be used.

2.4.3 Rule-based Planning

Medjahed [82] presents a technique to generate composite services from high-level declarative description. The method uses composability rules to determine whether two services are composable. The composition approach consists of four phases.: First,the specification phase enables high-level description of the desired compositions using a language called Composite Service Specification Language(CSSL). Then the matchmaking phase uses composability rules to generate composition plans that conform to service requester's specifications. If more than one plan is generated, in the selection phase, the service requester selects a plan based on quality of composition (QoC) parameters (e.g. rank, cost, etc.). The final phase is the generation phase. A detailed description of the composite service is automatically generated and presented to the service requester.

Here, we should pay more emphasis on the composability rules because it is the major issue to define how the plan is generated. The composability rules consider the syntactic and semantic properties of Web services. Syntactic rules include the rules for operation modes and the rules for binding protocols of interacting services. Semantic rules include the following subset: (1) message composability defines that two Web services are composable only if the output message of one service is compatible with the input message of another service; (2) operation semantic composability defines the compatibility between the domains, categories and purposes of two services; (3) qualitative composability defines the requester's preferences regarding the quality of operations for the composite service; and (4) composition soundness considers whether a composition of services is reasonable. To this end, the authors introduce the notion of composition templates that define the dependency between the different kinds of services.

The main contribution of this method is the composability rules, because they define the possible Web service's attributes that could be used in service composition. Those rules can be used as a guideline for other Web service methods based on planning.

SWORD [101] is another developer toolkit for building composite Web services using rule-based plan generation. SWORD does not deploy the emerging service-description standards such as WSDL and DAML-S, instead, it uses Entity-Relation (ER) model to specify the Web services. In SWORD, a service is modeled by its preconditions and postconditions. They are specified in a world model that consists of entities and relationships among entities. A Web service is represented in the form of a Horn rule that denotes the postconditions are achieve if the preconditions are true. To create a composite service, the service requester only needs specify the initial and final states for the composite service, then the plan generation can be achieved using a rule-based expert system. Besides the general composition methods, an interesting work done by SWORD is that the authors give a discussion on that the rule-based chaining can sometimes generate "uncertain" results if a precondition can not uniquely determines a postcondition. The authors argue that the uncertain results can avoid only when the preconditions are functionally depending on the postconditions inside a service. In fact, it may happen in most service composition methods described in this survey but not all authors explicitly declare it.

2.4.4 Other AI-planning Methods

Some other AI planning techniques are proposed for the automatic composition of Web services. In [125] the SHOP2 planner is applied for automatic composition of Web services, which are provided with DAML-S descriptions. SHOP2 is an Hierarchical Task Network(HTN) planner. The authors believe that the concept of task decomposition in HTN planning is very similar to the concept of composite process decomposition in DAML-S process ontology. The authors also claim that the HTN planner is more efficient than other planning language, such as Golog. In their paper, the authors give a very detailed description on the process of translating DAML-S to SHOP2. In particular, most control constructs can be expressed by SHOP2 in an explicit way.

Sirin et al [111] present a semi-automatic method for web service composition. Each time when a user has to select a Web service, all possible services, that match with the selected service, are presented to the user. The choice of the possible services is based both on functionalities and non-functional attributes. The functionalities (parameters) are presented by OWL classes and OWL reasoner is applied to match the services. A match is defined between two services when an output parameter of one service is the same OWL class or subclass of an input parameter of another service. The OWL inference engine can order the matched services so that the priority of the matches are lowered when the distance between the two types in the ontology tree increases. If more than one match is found, the system filters the services based on the non-functional attributes that are specified by the user as constraints. Only those services who pass the non-functional constraints can be presented to the service requester. The idea of semi-automatic service composition is quite interesting because it is very difficult to capture behavior in sufficient detail and compose the services in a fully automatic way, especially for the commercial-grade services. Although the proposed method is simple, it indicates the trend that automatic planner and human being can work together to generate the composite service for the user's request.

2.5 Service Composition using Program Synthesis

Program synthesis is a method of software engineering used to generate programs automatically. There are three different approaches to program synthesis: transformational, inductive and deductive program synthesis. Here, we only focus on the deductive program synthesis methods. Deductive program synthesis is based on the observation that proofs are equivalent to programs because each step of a proof can be interpreted as a step of a computation. This transforms program synthesis into a theorem-proving task. The key ideas of this approach, namely the correspondence between theorems and specifications and between constructive proofs and programs are presented in [70]. From some aspects, we can say that the Web service composition methods using deductive program synthesis are special cases of those using AI Planning, but the process generation has tightly connection with proof theories.

A general composition process using program synthesis case is shown in Figure 2.3. First, a description of existing Web Services is translated into logical axioms, and the requirements to the composite services are specified in form of a logical sequent to be proven. Second, a theorem prover is used to determine whether the sequent can be proven by the available axioms. If the answer is positive, the last step is to construct the process description from the complete proof.

Waldinger [122] elaborates an idea for service synthesis by theorem proving. The approach is based on automated deduction and program synthesis and has its roots in his earlier work [71]. Initially available services and user

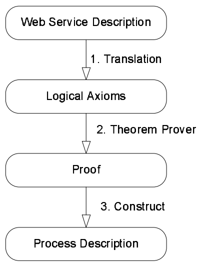


Figure 2.3: The process of Web service composition using program synthesis.

requirements are described in a first-order language, related to classical logic, and then constructive proofs are generated with SNARK theorem prover. Finally, service composition descriptions are extracted from particular proofs.

Lämmermann [64] applies Structural Synthesis of Program (SSP) for automated service composition. SSP is a deductive approach to synthesis of programs from specifications. The specifications of services only include the structural properties, i.e. the input/output information. SSP uses propositional variables as identifiers for input/output parameters and uses intuitionistic propositional logic for solving the composition problem. The composition is based on the proofs-as-programs property of intuitionistic logic. It equates the program of service composition to the problem of proof search. The author also takes advantage of disjunctions in classical logic to describe exceptions, which could be thrown away during service invocation.

2.6 Summary

This chapter has aimed to give an overview of recent progress in automated Web services composition techniques. Firstly, we propose a five-step model for Web services composition process. The composition model consists of service presentation, translation, process generation, evaluation and execution. Each step requires different languages, platforms and methods.

In these five steps, we concentrate on the methods of composite Web services process generation. We give the introduction and comparition of selected methods to support this step. The methods are enabled either by workflow research, AI planning or logic-based program synthesis. The workflow methods are mostly used in the situation where the request has already defined the process model, but automatic program is required to find the atomic services to fulfill the requirement. The AI planning methods is used when the requester has no process model but has a set of constraints and preferences. Hence the process model can be generated automatically by the program. The program synthesis methods are also fully automated. Theorem proving is used to generate the process model of the composite services.

Although the different methods provide different level of automation in service composition, we can not say the higher automation the better. Because the Web service environment is highly complex and it is not feasible to generate everything in an automatic way. Usually, the highly automated methods is suitable for generating the implementation skeletons that can be refined into formal specification. A discussion on this topic is presented by Hull et. al. [56].

Chapter 3

Logical Presentation of Web Services

3.1 Introduction

This chapter provides an analysis of the specification language for atomic Web services and its translation to LL. The reason that we distinguish the service specification language and the logical language is that they are implemented for different purposes when describing Web services. The service specification language is used by the users to enhance accessibility of the users in the sense that the users can easily express what they want. For this purpose, the specification should not only be easy to read and understand, but also easy to write by the user. Another main applicable purpose is that the specification language should allow for been exchanged and processed through Web. In reality, the users have already gotten used to the standard semantic Web service specification, DAML-S ServiceProfile, which is based on DAML, the W3C recommended XML markup language. The main problem is that DAML-S does not have explicitly defined semantics that can be used in its translation into logic. In order to present the translation in a formal way, we have defined an upper ontology for atomic Web services using the DAML+OIL notation. We can present in detail the translation mechanism from the defined classes, properties and relations in the upper ontology to the specific LL propositions.

The logical language is used to specify the Web services in a way that enables the automated inference. The description of existing web services is encoded as LL axioms, and the requirement to the composite service are specified in form of a LL sequent to be proven. We use an automated LL theorem prover to determine whether the sequent can be proven by the axioms. If the answer is positive, a process model of the composite service can be extracted from the generated proof. We use the propositional fragment of multiplicative, additive intuitionistic LL (MAILL) as the logical language to specify the Web services. Soundness and completeness of this fragment of LL have been reported in other publications in detail [66]. In this chapter, we will focus on the detail of presenting the Web service constructs by the LL literals and operators. In general, the Web service constructs include the functionalities, such as input, output, precondition, and effect as well as the non-functionality attributes.

The rest of this chapter is organized as follows: we introduce the LL and its features in Section 3.2, and further evaluate the expressive power of LL in the context of Web service in Section 3.3. Then we present the upper ontology of Web services and its general presentation in LL in Section 3.4. Section 3.5 and Section 3.6 discuss the detailed translation of service functionalities and non-functional attributes respectively. We will further present an example that considers all aspects of Web service specification in Section 3.7.

3.2 Linear Logic

LL is a refinement of classical logic introduced by J.-Y. Girard to provide a means for keeping track of "resources"—in LL two assumptions of a propositional constant A are distinguished from a single assumption of A. This feature forces the resources to be consumed exactly once. Therefore, resources in LL may only be supplied or consumed as they may not generally be copied or ignored. In other words, LL treats logical assumptions as consumable resources. In LL, the weakening and contraction rules that are supported in classical logic are available only for assumptions marked with the modality !. This means that, in general, an assumption not thus marked can only be used once in a branch of the search of a proof. Limited-use formulae can represent limited resources in some domain. Therefore LL is often described as a logic of resources rather than a logic of truth (such as classical logic). Since the assumptions in classical logic indicate the truth. Thus two truth values of the same fact are same as a single value. From this point, LL is not a logic about the truth, instead it is about the computation on amount.

Although LL is not the first attempt to develop resource-oriented logics (well-known examples are relevance logic [37] and Lambek calculus [63]), it is by now one of the most investigated one. The main reason is that LL greatly increases the expressiveness of logic programming, allowing direct and modular operational semantics specification of state, exceptions, continuations and concurrency in programming languages. Since its introduction LL has enjoyed increasing attention from researchers both in proof theory and computer science. Since LL embraces computational themes directly in its design, it often allows direct and declarative approaches to computational and resource sensitive specifications.

In this thesis, we use the propositional part of intuitionistic multiplicative additive fragment of LL (IMALL or MAILL). Lincoln [66] summarizes that the complexity result for the propositional MALL fragment(without !) is indicated to be PSPACE-complete. The syntax of the this LL fragment is presented by the following grammar:

$$A ::= \ P|A \multimap A|A \otimes A|A \& A|A \oplus A|!A|\mathbf{1}.$$

Here, P stands for a propositional variable and A ranges over formulae. The logic fragment consists of linear implication $\multimap$, multiplicative conjunction $\otimes$, additive conjunction $\&$, additive disjunction $\oplus$, "of course" (!) operator and trivial goal 1. We now introduce the LL connectives and their inference rules in detail.

Multiplicative Conjunction ($A \otimes B$**)** denotes that the literals A and B are consumed or achieved simultaneously. If $A \otimes B$ is a resource, we have to consume both A and B to achieve the goal. If $A \otimes B$ is a goal, we have to split our resources into two parts, Γ and $\Gamma^{'}$ and show that with resources Γ we can achieve A and with $\Gamma^{'}$ we can achieve B. The splitting of resources, viewed bottom-up, is a non-deterministic operation.

Additive Conjunction ($A \& B$**)** is called internal choice. if $A \& B$ is a resource, we have to make both A and B available before achieving the goal. However, they are not necessary to be consumed. If $A \& B$ is a goal, both A and B are achieved. Both literals are made available in both premises, since we have to make a choice on which one we want to achieve.

Additive Disjunction ($A \oplus B$**)** is equal to the disjunction in classical logic. If $A \oplus B$ is a resource, we can make either A or B available to achieve the goal. If $A \oplus B$ is a goal, either A or B are achieved.

Linear Implication ($A \multimap B$**)** internalizes the linear hypothetical judgment at the level of propositions. $A \multimap B$ means that the goal B is achievable only when resource A is available.

Of Course Modality ($!A$**)** means that we can use or generate a literal A as much as we want—the amount of the resource is infinite. While in classical logic literals may be copied by default, in LL this has to be stated explicitly. It can be used to present some kinds of resources, for example, duplicable informations, that are able to be duplicated arbitrarily.

Unit (1**)** presents the trivial goal which requires no resources.

The sequent of intuitionistic LL is an expression of the form $\Gamma; \Delta \vdash G$, where Γ and Δ are multisets of resource formulae, and G is a goal formula. Γ and Δ are called *intuitionistic context* and *linear context* respectively. The sequent can be interpreted as follows: given an unbounded set of resource Γ, the goal G can be achieved by consuming resource Δ. In other words, the sequent $\Gamma; \Delta \vdash G$ can be mapped to the LL sequent $!\Gamma, \Delta \vdash G$.

The inference rules of this fragment of LL are presented in Table 3.1.

LL provides us the expressive power to present Web services in some features that Classical Logic does not provide. In general, the reasons that LL is useful as a declarative language for Web service composition are presented as

Logical axiom and Cut rule:

$$A \vdash A \ (id) \qquad \frac{\Gamma \vdash A \quad \Gamma', A \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta} \ (Cut)$$

Rules for propositional constants:

$$\vdash 1 \qquad \frac{\Gamma \vdash A}{\Gamma, 1 \vdash A}$$

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C} \ (L\otimes) \qquad \frac{\Gamma \vdash A \quad \Gamma' \vdash B}{\Gamma, \Gamma' \vdash A \otimes B} \ (R\otimes)$$

$$\frac{\Gamma_1 \vdash A \quad \Gamma_2, B \vdash C}{\Gamma_1, (A \multimap B), \Gamma_2 \vdash C} \ (L \multimap) \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} \ (R \multimap)$$

$$\frac{\Gamma, A \vdash \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \oplus B \vdash \Delta} \ (L\oplus) \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A \oplus B} \ (R\oplus)(a) \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \oplus B} \ (R\oplus)(b)$$

$$\frac{\Gamma, A \vdash \Delta}{\Gamma, A \& B \vdash \Delta} \ (L\&)(a) \qquad \frac{\Gamma, B \vdash \Delta}{\Gamma, A \& B \vdash \Delta} \ (L\&)(b) \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \& B} \ (R\&)$$

Rules for exponential !:

$$\frac{\Gamma \vdash \Delta}{\Gamma, !A \vdash \Delta} \ (W!) \qquad \frac{\Gamma, A \vdash \Delta}{\Gamma, !A \vdash \Delta} \ (L!) \qquad \frac{\Gamma, !A, !A \vdash \Delta}{\Gamma, !A \vdash \Delta} \ (C!)$$

Table 3.1: Inference rules for MAILL fragment.

follows:

1. The multiplicative conjunction enable us to present the quantity of consumable resource in Web services, such as price, time and the size of cache. For example, Lincoln [66] uses LL multiplicative conjunction to present the amount of money in their examples.

2. Using or not using "of course" modality before a proposition enables us to distinguish two aspects of the service functionalities: the information transformation and the state change produced by the execution of the service. The information is presented by the input/output parameters. We assume information is reusable, thus the input values are used but not consumed after the execution of a service. The parameters are presented by propositions with "of course" modality. Contrarily, the states of the world are changed by the service execution. We interpret the change as that the old states are consumed and the new states are produced. Therefore, the multiple state values are presented by propositions without "of course" modality. It means the state values can be only consumed once. We will discuss this issue in more detail in Section 3.5.

3. In addition, the inference rules for the "of course" modality enable us to duplicate information in an explicit way. The rules are similar to the weakening and contraction in Classical Logic. Except the duplicable information, the "Of course" modality can also be used in some non-consumable non-functional attributes.

4. The additive conjunction and disjunction can distinguish the internal choice and the external choice. It is important in computation but has not specified explicitly in logic. For example, a dialog box that asks users to choose "yes" or "no" is an internal choice, because the user's choice will lead the program running in different branches. A typical external choice situation is that a service may produce one of several alternative outputs every time it is executed. In particular, this is the case with exception handling. The outputs produced by the service depend on the execution environment without any user's intervention. This issue will not be discuss deeply in this thesis. The main concern is that none of the available Web service specification languages supports external choice, so this issue is not usable in reality. However, we regard this an important direction of future development of Web service languages.

5. Last but not the least, LL has close connection with the concurrent processes that are foundations for modeling the composite Web services. In particular, the translation from proofs in LL into Milner's π-calculus [90] has been studied in many literatures. [8, 86, 18]. For example, the multiplicative conjunction($\otimes$) can present the "composition" in π-calculus; the

disjunction($\oplus$) presents “choice”, and the “of course” modality(!) presents “replication”.

In the following we justify and present in detail the procedure to translate the service specifications (encoded by DAML-S ServiceProfile) to logical axioms in LL.

3.3 The Expressive Power of LL

We have introduced the method for automated composition of the Web services by LL theorem proving. A main step for this task is the formal translation between LL and the Web service languages. Therefore, a fundamental question right now concerns the comparison of LL with other logic-based approaches for Web service composition. Here, we are going to evaluate the expressive power of LL. The purpose of the evaluation is to reveal whether the expressive power of LL is enough to present the features of Web services.

What are the features of Web services? Frankly, people are still arguing on what should be included in the Web service description to enable automated software to retrieve and compose Web services to achieve the goals specified by the end-users. Fortunately, there have been quite some work to reveal the formal foundation of Web services. In general, it has been agreed on two points. 1) the atomic services can be modeled as the state change problem, so those language that related to state-oriented programming can be used for the description of atomic services; 2) the process of the composite service can be described by the models of concurrent workflow. Accordingly, to evaluate the expressive power of LL, what we should do is to discuss the connection between LL and those two fields: state-oriented programming and models of concurrency.

Logic-based approaches for dealing with changes can be broadly classified into two categories: those designed for database or logic programming and those designed for reasoning about programs and actions [22]. Prolog is an early (and the base-known) example of the language in the first category. The situation calculus, Dynamic logic and Temporal Logic are the forefathers of the modern approaches to reasoning about actions.

Considering the specific requirement of Web services, the available Web services are modeled as black boxes, by which the services are viewed in terms of observable inputs and outputs with the internals of the system hidden from being viewed. Since Web service is a piece of modular software, we can model it as a transitional component through its interface. The transition made by the execution of the Web service includes two parts: 1) the Web service reads the information from the inputs and produces new information through the outputs; 2) besides the information change, the execution of the Web service will change the state of the environment. In a sense, the presentation of available

Web services is the modeling of the changes made by their execution, from the point of views for both information and environment.

The composite services have strong connection with the models of concurrency. The internal structure of a composite service is described by how control flow and data flow being formally separated within process models. The multiple subprocesses inside the composite service can execute in parallel. The interaction between the services and the environment is also concurrent.

We use Petri Nets as a mediator between the Web service languages and the LL. We select Petri Nets for its combination of compelling computational semantics, ease of implementation, and its ability to address both offline analysis tasks such Web service composition and online execution tasks such as deadlock determination resource satisfaction, and quantitative performance analysis. Most importantly, on the one hand, an operational semantics for DAML-S using Petri Nets has been developed in [92]; on the another hand, LL can be used to define a specification language for Petri nets, by giving precise correspondences, at different levels, between LL and Petri Nets.

Let us introduce Petri Nets briefly. Petri Nets are firstly introduced in Carl Adam Petri's PhD thesis(1962) to address the problems of concurrency in systems, including events, agents and actors. Petri Nets use a special class of generalized graphs or nets for a mathematical description of the system structure that can then be investigated analytically. Been represented as graphs, the Petri Nets consist of the basic notions from graph theory, including nodes, edges, and the manner in which the nodes and edges are interconnected. In Petri Nets, we have two types of nodes. By convention, the first type of node is called a *place* that represents the types of resources. The second type is called a *transition* that represents how resources are consumed or produced by actions. The edges of a Petri Net are called *arcs*. Usually, Petri Nets require that an arc can connect only two nodes that belong to different types. Therefore, there can only be one arc from a place to a transition, or from a transition to a place. In the former case, the place acts as an input for the transition, which in the latter case the place is the output.

Formally, a net $\mathcal{N} = (P, T, pre, post)$, where P is a set of places, T is a set of transitions, as well as two functions *pre* and *post* which map each transition $t \in T$ to a multiset of P, called the pre- and post multiset of t respectively.

The Semantic Web markup for Web services, DAML-S has been influenced by Petri Nets from the very beginning. In [92], the authors show the mapping from the DAML-S service description to the corresponding Petri Net structure. For the DAML-S atomic services, the effect axioms specify all and only the conditions for actions specify all and only the conditions under which a fluent can change; and the necessary conditions for actions specify all and only the conditions under which an action is possible to execute. Thus, Petri Nets provides a computational mechanism for achieving this completion. The graph structure defines the completion and the computation over the graph structure achieves the computational completion semantics. Hence, the solution to the frame problem is captured in the computational semantics of Petri Nets.

The DAML-S composite process are compositions of sub-process, all of which stem out in atomic processes. The DAML-S *composedOf* property specifies the control flow and data flow of its sub-processes, yielding constraints on the ordering and conditional execution of these sub-processes.

The Petri Nets structure for composite Web services represent a process which changes the state from the pre-condition to the post-condition. The pre-condition and post-condition corresponds to the start and finish items in Petri Nets. Inside the process, the atomic processes are connected by control constructs. Each construct is considered as an appropriate Net structure that captures a possible execution semantics of that construct. The paper gives the Petri Nets semantics for the basic control constructs *sequence*, *parallel*, *condition*, *choice* and *iterate*.

The DAML-S atomic processes correspond to transition and embedded composite processes are recursively built up from their ground atomic processes in a Petri Net. The overall system has a distributed operational semantics. Each transition is fired based on its local input conditions, and transition firings correspond to system evolution.

The translation from a Petri Net to a LL formula is straightforward. The idea is: each place of a Petri Net is regarded as an atomic proposition of LL, and transitions as provability relation. In [39], the authors conclude that intuitionistic LL is expressive enough to be considered seriously as a specification logic for parallel processes. The fragment of LL used here is equal to MAILL, which includes the following operators:

$$\otimes, \multimap, \oplus, \&, ! \text{ and } 1$$

The judgments of the correspondence between Petri Nets and LL are based on their interpretation of LL on Petri Nets, and with the implicit assumption that Petri Nets are a good general model of parallel processes. The paper shows how Petri Nets can naturally be made into models of MAILL fragment in such a way that many properties that one might wish to state of Petri Nets become expressible in LL.

With respect to a net N, the formulas of LL are interpreted as follows:

$$\begin{aligned}
[\![1]\!]_N &= \{m|m \to \underline{0}\} \\
[\![a]\!]_N &= \{m|m \to a\} \\
[\![A \otimes B]\!]_N &= \{m|\exists m_A \in [\![A]\!]_N, m_B \in [\![B]\!]_N.m \to m_A + m_B\} \\
[\![A \multimap B]\!]_N &= \{m|\forall m_A \in [\![A]\!]_N.m + m_A \in [\![B]\!]_N\} \\
[\![A\&B]\!]_N &= [\![A]\!]_N \cap [\![B]\!]_N \\
[\![A \oplus B]\!]_N &= [\![A]\!]_N \cup [\![B]\!]_N \\
[\![!A]\!]_N &= \bigcup\{q|q \text{ a postfixed point of } x \to 1 \cap [\![A]\!]_N \cap (x \otimes x)\}
\end{aligned}$$

In the interpretation, an atomic proposition a is interpreted as the downwards closure of the associated place. The downward closure is defined as a set of places that can reach place a after applying a series of transitions. Formally, let $m[t\rangle m'$ stand for transition $t \in T$ has multisets of input m and output m', the relation $m_0 \to m_t$ is defined as follows:

$$\exists t_1, \ldots, t_n \in T, m1, \ldots, m_n \in M, n \geq 0.m_0[t_1\rangle m_1 \ldots [t_n\rangle m_n = m_t$$

The soundness and completeness of the above interpretation are proved by [39]. Soundness states that all provable properties in LL hold in Petri Nets, while completeness states that properties which hold in any Petri Net can be proved in LL.

In summary, through the above analysis, we have proven that the MAILL fragment of LL has enough expressive power for current features of Web Services. The reason is that the Web service features can be formally presented by the semantics of Petri Nets, while the mapping between Petri Nets and LL is also presented.

3.4 The Upper Ontology of Web Services and LL

Figure 3.1 shows the upper ontology for the declarative specification of a Web service, namely the ServiceProfile. The upper ontology can be used as a specification framework to present either the request to the service or the advertisement of existing service. For the existing service, we do not distinguish the atomic service from composite service, because we model an existing service as a black-box. Black-box view of Web services regards a service purely in terms of observable interfaces with the internals of the system hidden from being viewed. One can make no assumption about its behavior or state beyond what is specified by its interface. Therefore, the service process that denotes

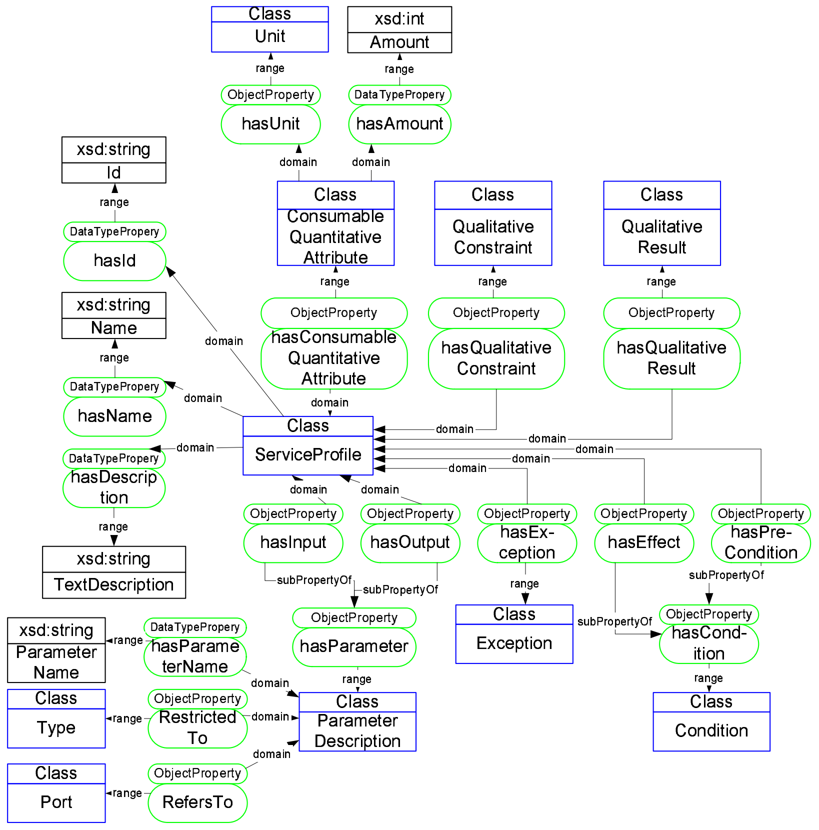


Figure 3.1: The upper ontology for Web service declarative specification.

the internal control- and data-flow inside the composite service is not considered when selecting the service.

The specification describes a service by general information, functionalities and non-functional attributes. The general information of a Web service includes the service Id, service name and an optional free text description. The functionalities include inputs, outputs, precondition, effects and exceptions. The inputs and outputs are signature definition of the parameters, which are specified by ParameterDescription class. This class is composed of three fields: the parameter name, the parameter type and the port where the data should be send or received. We do not elaborate the data structure for the values of precondition, effect and exception, because we just treat them as DAML classes. The non-functional attributes are classified into three categories: consumable quantitative attributes, qualitative constraints and qualitative results. Qualitative constraints and qualitative results are simply defined as DAML classes.

While the description of consumable quantitative attributes includes the unit and the amount.

The syntactic definition of the ServiceProfile presentation is represented by Extended BNF [91] as follows:

$$
\begin{aligned}
\langle ServiceProfile\rangle ::={}& '\langle'\textbf{hasId}\langle Id\rangle'\rangle' \\
& ['\langle'\textbf{hasName}\langle Name\rangle'\rangle'] \\
& ['\langle'\textbf{hasDescription}\langle TextDescription\rangle'\rangle'] \\
& \{'\langle'\textbf{hasInput}\langle Input\rangle'\rangle'\} \\
& \{'\langle'\textbf{hasOutput}\langle Output\rangle'\rangle'\} \\
& \{'\langle'\textbf{hasException}\langle Exception\rangle'\rangle'\} \\
& \{'\langle'\textbf{hasEffect}\langle Effect\rangle'\rangle'\} \\
& \{'\langle'\textbf{hasPrecondition}\langle Precondition\rangle'\rangle'\} \\
& \{'\langle'\textbf{hasConsumableQuantitativeAttribute}\langle ConsumableQuantitativeAttribute\rangle'\rangle'\} \\
& \{'\langle'\textbf{hasQualitativeConstraint}\langle QualitativeConstraint\rangle'\rangle'\} \\
& \{'\langle'\textbf{hasQualitativeResult}\langle QualitativeResult\rangle'\rangle'\} \\
\langle Id\rangle ::={}& \langle \%STRING\rangle \\
\langle Name\rangle ::={}& \langle \%STRING\rangle \\
\langle TextDescription\rangle ::={}& \langle \%STRING\rangle \\
\langle Input\rangle ::={}& \langle ParameterDescription\rangle \\
\langle Output\rangle ::={}& \langle ParameterDescription\rangle \\
\langle Effect\rangle ::={}& \langle Condition\rangle \\
\langle Precondition\rangle ::={}& \langle Condition\rangle \\
\langle ParameterDescription\rangle ::={}& '\langle'\textbf{ParameterName}\langle ParameterName\rangle'\rangle' \\
& '\langle'\textbf{RestrictedTo}\langle Type\rangle'\rangle' \\
& '\langle'\textbf{RefersTo}\langle Port\rangle'\rangle' \\
\langle Exception\rangle ::={}& \langle DAMLClass\rangle \\
\langle Condition\rangle ::={}& \langle DAMLClass\rangle \\
\langle ConsumableQuantitativeAttribute\rangle ::={}& '\langle'\textbf{hasUnit}\langle Unit\rangle'\rangle' \\
& '\langle'\textbf{hasAmount}\langle Amount\rangle'\rangle' \\
\langle QualitativeConstraint\rangle ::={}& \langle DAMLClass\rangle \\
\langle QualitativeResult\rangle ::={}& \langle DAMLClass\rangle \\
\langle ParameterName\rangle ::={}& \langle \%STRING\rangle \\
\langle Type\rangle ::={}& \langle DAMLClass\rangle \\
\langle Port\rangle ::={}& \langle DAMLClass\rangle \\
\langle Unit\rangle ::={}& \langle DAMLClass\rangle \\
\langle Amount\rangle ::={}& \langle \%INTEGER\rangle
\end{aligned}
$$

Here, $\%STRING$ and $\%INTEGER$ refer to the XSD datatype $xsd : string$ and $xsd : int$ respectively. $DAMLClass$ refer to the definition of the classes in DAML+OIL language. We interpret a DAML class by the URI that points to the address where the DAML class is defined.

The above ServiceProfile specification can be translated into LL formula. Generally, a requirement to a Web service (including atomic service or composite service) can be expressed by the following LL formula:

$$\Gamma; (\Delta_{cn} \otimes \Delta_{qc}) \vdash (P \otimes I \multimap (F \otimes O) \oplus E) \otimes \Delta_{qr}$$

where Γ is a set of extralogical axioms representing available web services. $\Delta_{cn} \otimes \Delta_{qc}$ is a multiplicative conjunction of non-functional constraints, consisting of consumable quantitative attributes and qualitative constraints. Δ_{qr} is a multiplicative conjunction of non-functional qualitative results. Some of the qualitative attributes may denoted by "of course" modality if necessary. We will introduce the detail in Section 3.6. $P \otimes I \multimap (F \otimes O) \oplus E$ is a functionality description of the required service. Both I and O are multiplicative conjunctions of literals, where I represents a set of input parameters for the service and O represents output parameters produced by the service. Because both of them are information, they are denoted by "of course" modality before them. P and F are multiplicative conjection of preconditions and effects respectively. E is a presentation of an exception. Intuitively, the formula can be explained as follows: given a set of available atomic services and non-functional attributes, try to find a combination of services that computes O from I as well as changes the world state from P to F. If the execution of the service fails, an exception is thrown. Every element in Γ is in the form of $(\Delta_{cn} \otimes \Delta_{qc}) \vdash (P \otimes I \multimap (F \otimes O) \oplus E) \otimes \Delta_{qr}$, where meanings of Δ_*, I, O, P, F and E are the same as described above.

More concretely, we define a translation $\mathcal{T}$ from the ServiceProfile specification language to the corresponding LL formula. Considering the syntax of the specification language, a translation is defined by the following:

$$
\begin{aligned}
\langle ServiceProfile\rangle^{\mathcal{T}} \equiv &(\langle ConsumableQuantitativeAttribute\rangle^{\mathcal{T}} \otimes \langle QualitativeConstraint\rangle^{\mathcal{T}}) \vdash \\
&(\langle Input\rangle^{\mathcal{T}} \otimes \langle Precondition\rangle^{\mathcal{T}} \multimap_{\langle Id\rangle^{\mathcal{T}}} (\langle Output\rangle^{\mathcal{T}} \otimes \langle Effect\rangle^{\mathcal{T}}) \oplus \langle Exception\rangle^{\mathcal{T}}) \\
&\otimes \langle QualitativeResult\rangle^{\mathcal{T}} \\
\langle ConsumableQuantitativeAttribute\rangle^{\mathcal{T}} \equiv &\langle Unit\rangle^{\mathcal{T}\langle Amount\rangle^{\mathcal{T}}} \\
\langle QualitativeConstraint\rangle^{\mathcal{T}} \equiv &\langle DAMLClass\rangle^{\mathcal{T}} \\
\langle QualitativeResult\rangle^{\mathcal{T}} \equiv &\langle DAMLClass\rangle^{\mathcal{T}} \\
\langle Input\rangle^{\mathcal{T}} \equiv &\langle ParameterDescription\rangle^{\mathcal{T}} \\
\langle Output\rangle^{\mathcal{T}} \equiv &\langle ParameterDescription\rangle^{\mathcal{T}} \\
\langle Exception\rangle^{\mathcal{T}} \equiv &\langle DAMLClass\rangle^{\mathcal{T}} \\
\langle Effect\rangle^{\mathcal{T}} \equiv &\langle Condition\rangle^{\mathcal{T}} \\
\langle Precondition\rangle^{\mathcal{T}} \equiv &\langle Condition\rangle^{\mathcal{T}} \\
\langle Id\rangle^{\mathcal{T}} \equiv &\langle \%STRING\rangle^{\mathcal{T}} \\
\langle Unit\rangle^{\mathcal{T}} \equiv &\langle DAMLClass\rangle^{\mathcal{T}} \\
\langle Amount\rangle^{\mathcal{T}} \equiv &\langle \%INTEGER\rangle^{\mathcal{T}} \\
\langle ParameterDescription\rangle^{\mathcal{T}} \equiv &\langle Type\rangle^{\mathcal{T}}(\langle Port\rangle^{\mathcal{T}}) \\
\langle Condition\rangle^{\mathcal{T}} \equiv &\langle DAMLClass\rangle^{\mathcal{T}} \\
\langle Type\rangle^{\mathcal{T}} \equiv &\langle DAMLClass\rangle^{\mathcal{T}} \\
\langle Port\rangle^{\mathcal{T}} \equiv &\langle DAMLClass\rangle^{\mathcal{T}} \\
\langle \%STRING\rangle^{\mathcal{T}} \equiv &\textbf{the value of the string} \\
\langle \%INTEGER\rangle^{\mathcal{T}} \equiv &\textbf{the value of the integer} \\
\langle DAMLClass\rangle^{\mathcal{T}} \equiv &\textbf{the URI of the DAMLClass}
\end{aligned}
$$

We will give the detail explanation of the concrete translation through examples in the consequent sections.

3.5 Transformation of Functionalities

We use DAML-S ServiceProfile as the declarative specification language for Web services. An essential component of the ServiceProfile is the specification of what functionality the service provides and the specification of the conditions that must be satisfied for a successful result. We have to distinguish two aspects of the functionality of the service: the information transformation and the state change produced by the execution of the service. The information transformation is represented as a transformation from the input parameters required by the service to the output parameters produced by the service. It provides information about the data flow during the execution of a Web service. The state change provides information about what the Web service actually does. We model it as the transformation from one state to another state in the world. A typical example is a service for logging in a web site. The input information is the username and password, and the output is a confirmation message. After the execution, the world state changes from "not login" to "login". The "login" state will be kept until the "logout" service is invoked.

Both aspects are important for automated service provision and composition, and they are specified differently in DAML-S. The parameters are encoded by their name, type and the reference to the process model, while the states are presented by a collection of DAML Classes. In the logical translation, the parameter types are denoted by a proposition and the references are denoted by the proposition's proof term. The states are encoded by a set of proposition variables that are connected by multiplicative conjunction operators. Unlike the parameters, the states do not provide information for generating the dataflow. So they have no attached proof terms. In the following, we will discuss the translation strategy for those two aspects in greater detail.

3.5.1 Information Transformation

In DAML-S ServiceProfile, the information transformation is represented by ObjectProperties with the names "input" and "output". Both input and output are subproperties of "parameter". The input property specifies the information that the service requires to proceed with the computation. The output property specifies the result of the computation by the service. The value of "parameter" property ranges over the instances of the class "ParameterDescription". The class is a collection of the name of the parameter that can be used as an identifier, the parameter's type and a reference to the corresponding parameter in the process model.

parameterName is the name of the actual parameter, which is presented by a string.

restrictedTo points to the class that is the type of the described parameter. The type is allowed to be specified by either one DAML class or one XSD datatype. In other words, a parameter is restricted to only one concept in the domain ontology.

refersTo provides a reference to the parameter defined in the DAML-S ServiceModel.

We present an example of the functionalities presented in DAML-S for a measurement converter service as follows. We will explain the translation from DAML-S ServiceProfile to LL axiom by this example.

```
<profileHierarchy:InformationService
       rdf:ID="MesurementTransformation">
  <!-- reference to the service specification -->
  <service:presentedBy rdf:resource="&service;#cm2inch"/>
```

```
    <profile:serviceName>Transform_CM_to_Inch</profile:serviceName>
    <profile:textDescription>
        This service transform a length measured by CM into a length measuered by inch
    </profile:textDescription>

    <profile:input>
      <profile:ParameterDescription rdf:ID="LengthCM">
        <profile:parameterName>Length_By_CM</profile:parameterName>
        <profile:restrictedTo rdf:resource="&onto;#Centimeter"/>
        <profile:refersTo rdf:resource="&model;#LengthCM"/>
      </profile:ParameterDescription>
    </profile:input>

    <profile:output>
      <profile:ParameterDescription rdf:ID="LengthIn">
        <profile:parameterName>Length_By_Inch</profile:parameterName>
        <profile:restrictedTo rdf:resource="&onto;#Inch"/>
        <profile:refersTo rdf:resource="&model;#LengthIn"/>
      </profile:ParameterDescription>
    </profile:output>
  </profileHierarchy:TransformationService>
```

Here, we use entity types as a shorthand for URIs. For example, "&*onto*; #*Centimeter*" refers to the URI of the definitions for measurement converting services, for example: http://reliant.teknowledge.com/DAML/SUMO.owl#Centimeter. We translate the input parameter into the following LL term. To avoid the conflict with the reserved LL operators, we remove "&" and ";" in the RDF resource identifier. The removal does not effect the expression of both DAML and LL.

$$onto\#Centimeter(model\#LengthCM)$$

The value of "restrictedTo" property is translated into a LL proposition and the value of "refersTo" is translated into a proof term which identifies the proposition. The proof term is used to guarantee the proposition, which are introduced during the reasoning process, and are not used outside the proposition's scope. In the term of program, it means that the proof term is a program of type that is restricted by the proposition. This denotion has been used in functional programming and the literatures about *propositions-as-types* or *proofs-as-programs* paradigm. In particular, the Curry-Howard [55] isomorphism specifies that a proposition is identified with the type of its proofs – every proposition is a type specifying a construction that counts as evidence for it. For the input parameter, a proof term just refers to the input variable, which is parameter definition in ServiceModel in our case. For the output parameter, the proof term presents the process that calculates the outputs from the given inputs. For the atomic service, the proof term is a combination of the service id and the output parameters. For example, the output of the above example is able to be presented as:

$$service\#cm2inch : onto\#Inch(model\#LengthIn)$$

The proof term for the output parameter is different from the input parameter because we should denote how the output is calculated. The term means the invocation of service "cm2inch" produces the output "LengthIn". The term in front of the ":" mark is denoted by the name of a service.

From the computation point of view, this service in the above example requires an input that has type "&*onto*; #*Centimeter*"(the value of length measured in Centimeter) and produces an output that has type "&*onto*; #*Inch*"(the value of length measured in Inch). A complete LL formula that describes the above presented DAML-S document is as follows:

$$\begin{aligned}&\vdash !onto\#Centimeter(model\#LengthCM) \multimap \\ &\quad service\#cm2inch :!onto\#Inch(model\#LengthIn)\end{aligned}$$

The multiple input/output parameters are connected by multiplicative conjunctions and each proposition has "of course" modality in front. The reason that we use "of course" modality is that we believe that the service consumes and produces information, that can be either discarded or duplicated by the user. Therefore we have to enable weakening and contraction in logic. Since the "of course" modality supports the weakening and contraction in its inference rules, we do not necessarily need further justification. The main points will be sufficiently clear from the following examples.

Example. Consider the following two LL sequents:

$$(!A \multimap !C, !A \otimes !C \multimap !D); . \vdash !A \multimap !D \tag{3.1}$$

$$(!A \multimap !C \otimes !D, !C \multimap !E); . \vdash !A \multimap !E \tag{3.2}$$

Neither of these sequents are provable if we remove all "of course" modality "!", because (3.1) uses A twice, (3.2) discards E. These are common in computation. By using "of course" modality, they become provable.

The proof for sequent (3.1)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{axiom}}{\vdash !A \multimap !C} \qquad \dfrac{\dfrac{\text{id}}{!A \vdash !A} \quad \dfrac{\text{id}}{!C \vdash !C}}{!A, !A \multimap !C \vdash !C}\ L\multimap}{!A \vdash !C}\ cut \qquad \dfrac{\dfrac{\text{axiom}}{\vdash !A \otimes !C \multimap !D} \qquad \dfrac{\dfrac{\dfrac{\text{id}}{!A \vdash !A} \quad \dfrac{\text{id}}{!C \vdash !C}}{!A, !C \vdash !A \otimes !C}\ R\otimes \quad \dfrac{\text{id}}{!D \vdash !D}}{!A, !C, (!A \otimes !C \multimap !D) \vdash !D}\ L\multimap}{!A, !C \vdash !D}\ cut}{!A, !A \vdash !D}\ cut}{!A \vdash !D}\ C!}{\vdash !A \multimap !D}\ R\multimap$$

The proof for sequent (3.2)

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{axiom}}{\vdash !A \multimap !C \otimes !D} \qquad \dfrac{\dfrac{\text{id}}{!A \vdash !A} \quad \dfrac{\text{id}}{!C \otimes !D \vdash !C \otimes !D}}{!A, (!A \multimap !C \otimes !D) \vdash !C \otimes !D)}\ L\multimap}{!A \vdash !C \otimes !D}\ cut \qquad \dfrac{\dfrac{\dfrac{\dfrac{\text{axiom}}{\vdash !C \multimap !E} \qquad \dfrac{\dfrac{\text{id}}{!C \vdash !C} \quad \dfrac{\text{id}}{!E \vdash !E}}{!C, !C \multimap !E \vdash !E}\ L\multimap}{!C \vdash !E}\ cut}{!C, !D \vdash !E}\ W!}{!C \otimes !D \vdash !E}\ L\otimes}{!A \vdash !E}\ cut}{\vdash !A \multimap !E}\ R\multimap$$

3.5.2 State Change

Instead of modeling Web services by functions or relations, it is more important to take account of the possible interactions between the services and their environment during the course of their invocation. The main concern is that the services, after the execution, will change the state of their environment. Many research efforts tackling Web service composition problem via AI planning consider the state changes produced by Web Services [80, 125, 108]. In general, a planning problem can be described as a five-tuple $\langle S, S_0, G, A, \Gamma \rangle$, where S is the set of all possible states of the world, $S_0 \subset S$ denotes the initial state of the world, $G \subset S$ denotes the goal state of the world the planning system attempts to reach, A is the set of actions the planner can perform in attempting to change one state to another state in the world, and the translation relation $\Gamma \subseteq S \times A \times S$ defines the precondition and effects for the execution of each action.

In the term of Web services, S_0 and G are the initial states and the goal states specified in the requirement of Web service users. A is a set of available services. Γ further denotes the state change function of each service.

The state change produced by the execution of the service is specified through the precondition and effect properties of the profile. Precondition presents logical conditions that should be satisfied prior to the service being requested. Effects are the result of the successful execution of a service. In DAML-S, both precondition and effect take the value of a "Condition" class. An instance of "Condition" is a logical formula that is evaluated to be true or false. Since DAML+OIL, the language used to build DAML-S, uses Description Logic [45] as its logical foundation, the DAML+OIL has the express power allowing for logical expressions.

In our translation, we encode the classes as propositions in LL. Multiple conditions are connected by multiplicative conjunction because DAML-S enforces that all conditions must be fulfilled before the execution of a Web service. We further assume that all preconditions are replaced by the effects after the execution.

A variety of works in AI on planning languages are considered in developing the DAML-S ontology. The people in DAML-S coalition has shown how DAML-S inherited characteristics from Petri Nets [92] and PDDL [77]. Both languages are used to as formal languages to express the states and transitions in a process. On the another hand, LL is a powerful tool for reasoning about state. LL is able to be used as a logical language to present both Petri Nets and PDDL. Thus it is no difficult to use LL as a method for formally representing state-based reasoning. We present each condition by a LL proposition. The meaning of a proposition is specified by the detail of definition inside the condition class.

In the following, we present an example considering both information transformation and state change. A service that enables the user to log on a web site receives the input as username and password. If the login is successfully, a “LoginOk” message is outputed. The further requirement for this service is: before the service is executed, the user can not have been logged in the web site. So the precondition is “NotLogin”. After the successful execution of the service, the state changes from “NotLogin” to “Login”. We present the DAML-S code as follows:

```
<profileHierarchy:InformationService rdf:ID="LoginToWebSite">
  <!-- reference to the service specification -->
  <service:presentedBy rdf:resource="&service;#login"/>

  <profile:serviceName>Login_to_A_Web_site</profile:serviceName>
  <profile:textDescription>
      This service enables the user to log on a web site by username and password
  </profile:textDescription>

  <profile:input>
    <profile:ParameterDescription rdf:ID="Username">
      <profile:parameterName>UserName</profile:parameterName>
      <profile:restrictedTo rdf:resource="&onto;#Username"/>
      <profile:refersTo rdf:resource="&model;#User"/>
    </profile:ParameterDescription>
  </profile:input>

  <profile:input>
    <profile:ParameterDescription rdf:ID="Password">
      <profile:parameterName>PassWord</profile:parameterName>
      <profile:restrictedTo rdf:resource="&onto;#Password"/>
      <profile:refersTo rdf:resource="&model;#Passwd"/>
    </profile:ParameterDescription>
  </profile:input>

  <profile:output>
    <profile:ParameterDescription rdf:ID="Confirmation">
      <profile:parameterName>Login_Confirmation</profile:parameterName>
      <profile:restrictedTo rdf:resource="&onto;#LoginOk"/>
      <profile:refersTo rdf:resource="&model;#LoginOk"/>
    </profile:ParameterDescription>
  </profile:output>

  <profile:precondition>
    <profile:ConditionDescription rdf:ID="NotLogin">
      <profile:parameterName>NotLogin</profile:parameterName>
      <profile:condition rdf:resource="&onto;#NotLogin"/>
    </profile:ConditionDescription>
  </profile:precondition>

  <profile:effect>
    <profile:ConditionDescription rdf:ID="Login">
      <profile:parameterName>Login</profile:parameterName>
      <profile:condition rdf:resource="&onto;#Login"/>
    </profile:ConditionDescription>
  </profile:precondition>

</profileHierarchy:TransformationService>
```

The above DAML-S code is able to be translated into the following LL axiom. In front of the propositions that represent parameters are “of course” modality. The propositions for state values has no "of course" modality.

$$\vdash !onto\#Username(model\#User) \otimes !onto\#Password(model\#Passwd)$$
$$\otimes onto\#NotLogin \multimap !onto\#LoginOk(model\#LoginOk) \otimes onto\#Login$$

To simplify the problem and improve the efficiency of the prover, we discard the conditional effects that are supported by DAML-S. Only the effect when the service executes successfully is presented. Otherwise, an exception is thrown and the state of the world has no change. The detail of the failure can be given in the exception. In that manner, the conditional effects are presented implicitly in the exception information.

3.6 Transformation of Non-functional Attributes

Non-functional attributes are considered to be constraints over the functionalities of the service [32]. In the thesis, we use non-functional attributes to evaluate and select services when there are several services having the same functionalities. In this section, we discuss how to represent the non-functional attributes in term of LL propositions. First of all, we will propose a general enough model in which the non-functional attributes are classified into three categories:

- **Consumable Quantitative Attributes:** These attributes limit the amount of resources that can be consumed by the composite service. The total amount of resources is the sum of all resources of atomic services that form the composite service. For example, the price of composite service is the sum of prices for all included atomic services. This kind of attribute includes total cost, total execution time, etc. In LL, we use a^n to present the amount of resource a is n. In the definition in LL, a^n is an abbreviation of n times of multiplicative conjunction: $\underbrace{a \otimes \ldots \otimes a}_{n}$, for $n > 0$, so n has to be positive integer. But the type can be extended by revising the LL interpreter.

- **Qualitative Constraints:** Attributes which can not be expressed by quantities are called qualitative attributes. Qualitative Constraints are qualitative attributes which specify requirements to the execution of a Web service. In other words, constraint indicates the service provider's demand that the requester should provide. For example, some services may response only to some authorized calls. The constraints are regarded as prerequisite resource in LL.

- **Qualitative Results:** Another kind of qualitative attributes (such as service type, service provider or geographical location) specify the results regarding the services' context. Contrary to the service constraint, service result indicates the service requester's demand that the service provider should provide. These attributes can be regarded as goals in LL.

We give a tip to distinguish the constraints and results. If a service is selected when the required value is more general than the service's value given an attribute, the attribute is a result. Otherwise, it is a constraint. A typical example is the location of a service. A service concerns two kinds of location information. One is the location of the service, the other is the location of the request that the service will respond. If a service is located in Norway, and a requester requires a service located in Europe. Because the request is more general than the service's value, the location of the service is a result. Contrarily, if a service only responds the request from Europe, the request located in Norway can get response, because the service's value is more general than the request value.

In the following, we have to justify that the most common used non-functional attributes can be put into the three categories. Although there is no agreed conclusion on a comprehensive list of non-functional attributes used to describe the Web services, a survey of non-functional service properties has been reported in [95]. In this survey, the authors classify non-functional attributes into the following categories: temporal and spatial availability, channels, charging styles, settlement models, settlement contracts, payment, service quality, security, trust and ownership. We have noticed that all the attributes are very complex if we want to accurately describe the detail of them. Fortunately, the Web services community does not use all advanced features of those attributes. The non-functional attributes of Web Services just express condition in which the Web services are available. In the following, we will discuss the detail of those attributes from three aspects: (1), the meaning and purpose of a given category of non-functional attributes; (2) the ontology that has been constructed for the given category; and (3) the map of the attributes to the LL context.

Temporal and spatial availability of a Web service is specified by the constraints of when and where the service is invocable. For example, a Web service may be used at a given duration of time. The spatial constraint of Web services have two attributes. First, it is necessary to indicate the location of a Web service. Second, the Web Service may have constraint to respond only to the invocation request from a given location. A DAML ontology for time and space is defined at http://www.cs.rochester.edu/~ferguson/daml/. The ontology defines whether an attribute A is more spatially available or more temporally available than an attribute A'. As we have discussed above, the location of service is represented by result and the requester's location is constraint in LL. The temporal availability is a constraint to the invocation time from the the requester, so it is also a qualitative constraint in LL presentation. [33]

Channels describe the endpoints where the information being transmitted. The port address of input and output parameters may be a part of description of service functionalities, but more information is required to invoke the service. For example, we have to denote the service is supported by RPC or HTTP protocol in WSDL. An overview of invocation protocol for Web services can be found at [117].The channel information is represented as qualitative constraint because it is used to enforce the format of request.

Charging styles describe the charging technique applied by a service provider for the use of its service. In general, the charging styles for Web services are (1) per service request; and (2) by unit of measure and granularity (e.g. by length, volume, weight, area or time). Charging style is regarded as a constraint to the service requester.

Settlement is a process that reflects the mutual obligations of the provider and requester. If a settlement is an obligation of the requester, it is a constraint. Otherwise, it is a result.

Payment describe the price and the currency for the cost of executing the Web services. Payment is a quantitative value that is reduced after the execution of the service.

Service quality is a measure of the difference between expected and actual service provision. The most notable work on measuring customer perceptions of service quality is SERVQUAL [98]. From the viewpoint of the requester, it measures the competence of the provider to deliver a service. So it is represented by service goal in LL.

Security and trust concerns the issues of identity, privacy, alteration and repudiation of information transferred between the service providers and service requesters. A security annotation for DAML Web services has been reported in [35]. In this paper, the authors distinguish the security-related capabilities and requirements of Web services and exploit them for reasoning and matching against service requests. We consider the capabilities and requirements from the service provider's perspective. Therefore, security capabilities are those features provided by the Web Services, so they are represented by service goal. The security requirements are features that the consumer of a Web service should have in order to invoke the service, so they are service constraint in LL presentation.

	Consumable Quantitative Attributes	*Qualitative Constraints*	*Qualitative results*
Temporal and spatial information		temporal availability, spatial availability	service location
Channels		invocation protocol	
Charging styles		charging style	
Settlement		obligation of the requester	obligation of the provider
Payment	cost		
Service quality			quality
Security and trust		requirement	capability
Ownership		owner	

Table 3.2: The classification of non-functional attributes.

Ownership provides detail information of the service provider, such as company name, telephone number, address, etc. It is a fact of the service and expressed by service goal.

A summary of the non-functional attributes and their category is shown in table 3.2. Given this classification, all non-functional attributes listed here are able to be presented by LL propositions.

In DAML-S, all non-functional attributes are presented by a general expandable property in ServiceProfile: serviceParameters. In this property, the range of each attributes is unconstrained, i.e. no range restrictions are placed on the service parameters at present. Specific service parameters will specialize this property by restricting the range appropriately and using subPropertyOf. The presentation of qualitative constraints and results are similar to those of service precondition and effects. In LL, we use a propositional variable referring to the URI of the DAML class that presents the attributes. The consumable quantitative attributes are presented by two parts. One is the description of the attributes that is presented by a propositional variable in LL and a class in DAML. The other is the amount value of the attribute and presented by a positive integer.

The different categories of non-functional attributes have different presentation in extralogical axioms. As it was mentioned before, the non-functional attributes can be described either as constraints or results and they can be presented as follows:

- **The constraints for the service:**

$$\Delta_c = Consumable^x \otimes !Constraint$$

- **The results produced by the service:**

$$\Delta_r = !Fact$$

3.7 Example

In order to illustrate the LL presentation of Web services, let us consider a simplified service for recommending skis to the customer according to his/her size and skiing skill level. Available atomic services are specified in DAML-S ServiceProfile documents which can be found at http://bromstad.idi.ntnu.no/services. An example of service *CM2INCH* has already been presented in Section 3.5.1. These documents are automatically translated to the following axioms by a translater that uses the corresponding concepts we describe in the upper ontology. The example takes both functionalities and non-functional attributes into consideration. For the sake of readability, we omit the namespace of the parameters.

The available services are specified as follows:

$$\Gamma = \begin{array}{l} NOK^{10} \vdash !PRICE\_LIMIT \otimes !SKILL\_LEVEL \multimap_{SelectModel} !BRAND \otimes !MODEL \\ \vdash !HEIGHT\_CM \otimes !WEIGHT\_KG \multimap_{SelectLength} !LENGTH\_CM \\ NOK^{20} \vdash !LENGTH\_CM \multimap_{CM2INCH} !LENGTH\_IN \\ CA\_MICORSOFT \vdash (!PRICE\_USD \multimap_{USD2NOK} !PRICE\_NOK) \otimes LOC\_NORWAY \\ \vdash !LENGTH\_IN \otimes !BRAND \otimes !MODEL \multimap_{SelectSki} !PRICE\_USD \oplus !OUT\_STOCK \end{array}$$

The axioms can be explained as follows:

- *SelectModel* — This service recommends a brand and a model of skis given the preferences on price and skill level. NOK^{10} on the left hand side of this service denotes $\underbrace{NOK \otimes \ldots \otimes NOK}_{10}$. This means that 10 Norwegian Krones are consumed by executing this service;

- *SelectLength* — This service provides the recommended ski length given body height and body weight. This service costs 20 NOK;

- *CM2INCH* — This service transforms a length measured by centimeter to a length measured by inch;

- *USD2NOK* — This is a currency exchange service that can calculated the amount in Norwegian Krones given the amount in US Dollar. It is located in Norway, and it only responses to the execution requests that have been certificated by Microsoft. Here $CA\_MICROSOFT$ is a non-functional constraint and LOC_NORWAY is a non-functional result.

- *SelectSki* — This service provides the skis price in US Dollar given a model, a brand and a length of the skis. If the required skies do not exist, an "out of stock" exception is returned.

Let's consider a required composite service specified by the following formula:

$$(\Gamma);\Delta_1 \vdash (\mathit{PRICE\_LIMIT} \otimes \mathit{SKILL\_LEVEL} \otimes \mathit{HEIGHT\_CM} \otimes \mathit{WEIGHT\_KG} \multimap \mathit{PRICE\_NOK} \oplus \mathit{OUT\_STOCK}) \otimes \Delta_2$$

The constraints for the composite service are as follows:

$$\Delta_1 = \mathit{NOK}^{35} \otimes !\mathit{CA\_MICROSOFT}$$
$$\Delta_2 = !\mathit{LOC\_NORWAY}$$

They mean that we would like to spend at most 35 NOK for the composite service. The composite service consumer has certification from Microsoft (!CA_MICROSOFT) and it requires that all location-aware services are located within Norway (*!LOC_NORWAY*). ! symbol describes that we allow an unbound number of atomic services in the composite service. We consider quantitative constraints (for example, price) as regular resources in LL. If the total number of resources required by services (which is determined by functionality attributes) is less than the number of available resources, the services can be included into composite service. Otherwise, if, for example, the *SelectModel* service would cost 20 NOK instead of 10 NOK then the total required amount would be 40 NOK and the composition is not valid.

For the qualitative constraints (for example, location), the service uses a literal (for example, *LOC_NORWAY*) to present its value, and we can determine in the set of requirements Δ_1 whether a service meets the requirement. However, if there is no such literal in the service description, the constraint is not applied to this service at all.

3.8 Summary

In this chapter we have discussed how DAML-S ServiceProfile can be translated into LL language. A service specification in DAML-S ServiceProfile includes the functionalities and non-functional attributes of Web services. Further, functionalities consist of the denotion of information transformation and state change produced by the execution of the service. The non-functional attributes are classified into three categories: consumable quantitative attributes, qualitative constraints and qualitative results. We have reviewed the common used non-functional attributes and put them into different categories. In DAML-S, the information about Web services is presented by DAML classes and properties. They are translated to LL propositions that refer to the specific classes and properties. The meaning of the propositions and the semantic relation among the propositions are defined by the ontology relations in DAML.

After the ServiceProfile is translated into LL axioms and LL sequents, the next steps are theorem proving in LL and derivation of the process model from proof of the requested composite service. If it has been proven that the requested composite service can be generated using available services, the process model can be extracted from the proof guaranteed by the inference rules. A resulting dataflow of the selected atomic services can also be presented to the user for inspection. In the next chapter, we will discuss the method to extract the dataflow process from the proof.

Chapter 4

Extraction of a Process Model from Proof

4.1 Introduction

Chapter 3 introduces the translation from the DAML-S ServiceProfile to LL language. After service specifications in DAML-S are translated into LL extralogical axioms and a service request from the customer is translated into a form of theorem to be proven, the LL theorem prover is invoked. The task of the LL theorem prover is to prove the specified theorem using LL inference rules. If the answer is positive, a process model of the composite service can be extracted from the generated proof. The process model is presented by a process calculus formally and it is attached to the proof as a type system. The process calculus process can be further translated into either DAML-S ServiceModel or BPEL4WS.

This chapter focuses on the process model extraction and the translation from the process calculus to Web service process languages. For the former, the method of process extraction from the proof is introduced. Most the previous work have been reported on the program extraction from the proof that uses λ-calculus to model the sequential algorithmic processes. The λ-calculus theory is about modeling systems which have no or little interactions with their environment. On the contrary, the π-calculus theory developed by Robin Milner [90] in the late 1980s is about the modeling of concurrent communicating systems that interact with each other through explicitly defined ports and channels. The correspondence between Web services and π-calculus has been addressed by research efforts in [83, 123].

For the latter, the translation between the internal and the external process languages is presented. Internally, the process for the generated composite service is presented by a process calculus stemmed from π-calculus. This calculus is a formal language that is designed to be easily attached to LL formulae, while the external languages aim to be understood by the Web service users and supported by the execution engine. For this purpose, both DAML-S ServiceModel and BPEL4WS are good candidates for the external languages. DAML-S ServiceModel is the process description part of DAML-S, which provides the Semantic Web markup to Web service description. Using DAML-S ServiceModel for composite service enables the compatibility with the DAML-S ServiceProfile that is used to present more precisely the available services that form the composite service. The problem, however, is that no platform is available so far to support the invocation of the services that are described by DAML-S ServiceModel language. A possible solution is to use BPEL4WS, the process language based on WSDL, as the grounding of DAML-S ServiceModel since DAML-S uses WSDL as the service grounding to define the invocable interface of the atomic service [72]. To enable the translation from the process calculus to either DAML-S ServiceModel or BPEL4WS, we propose an upper ontology for service process. Since the upper ontology contains the essential constructs in both DAML-S ServiceModel and BPEL4WS, the translation from the upper ontology to the process language is straightforward. Further, the process calculus can be translated into the upper ontology, then the upper ontology can be translated into either DAML-S ServiceModel or BPEL4WS upon user's request. Therefore, the upper ontology is used as a mediator language between the process calculus and the process language. Although the upper ontology does not take full advantage of all supported functions in DAML-S ServiceModel and BPEL4WS, it supports enough constructs for the composite service that can be generated by the automated composition method proposed in this thesis.

The rest of this chapter is organized as follows. Section 4.2 introduces the π-calculus and its use in Web service composition. Section 4.3 presents the upper ontology for composite service process. Section 4.4 discusses the concrete method to extract the process model from the proof. At last, an example is presented in Section 4.5.

4.2 The π-calculus: a Formality of Web Service Process

The process calculus we use to present composite service process model is inspired by π-calculus. π-calculus is a process calculus that is used to describe dynamically changing networks of concurrent processes. The fundamental activities in π-calculus are processes, which exchange information over channels and ports. In π-calculus, name is a basic notion: values, variables, ports and processes are all referred by names. Usually, the names starting with small letters range from variables to ports, and the names starting with capital letter refer to the processes. The grammar of polyadic π-calculus is as follow.

$$P \quad ::= \quad \mathbf{0} \,\Big|\, a(x).P \,\Big|\, \overline{a}\langle x\rangle.P \,\Big|\, (\nu a)P \,\Big|\, !a(x).P \,\Big|\, P.P \,\Big|\, P|P \,\Big|\, P+P$$

Polyadic π-calculus is built from the operators of inaction, input prefix, output prefix, restriction, sequence, parallel composition, and global choice. $\mathbf{0}$ is an inactive process and it does not perform any action. An input prefixed process $a(x).P$ receives a variable or message x through port a then executes process P. An output $\overline{a}\langle x\rangle.P$ emits message x at port a then executes process P. The restriction $(\nu a)P$ defines a name a local to process P. Unlike the global name, the name a is private and its scope is limited to P. A replication $!a(x).P$ stands for a countably infinite number of copies of channel a in parallel. The rest are control operators. "." represents two processes execute in sequence. The execution of two processes is synchronous, while the second process is prevented from executing until the first process has completed. "|" represents two processes execute concurrently. The two processes can performs independently and also communicate each other. "+" represents a non-deterministic choice which either the first process or the second process will execute.

The structural congruence relation defined in the π-calculus will allow us to rewrite a process in a semantically equal means. Here we introduce the following congruence relations:

Commutative law:

$$P|Q \equiv Q|P \qquad P+Q \equiv Q+P$$

Associative law:

$$(P|Q)|R \equiv P|(Q|R) \qquad (P+Q)+R \equiv P+(Q+R)$$

Inactive law:

$$P \equiv P|\mathbf{0} \equiv P+\mathbf{0} \equiv P.\mathbf{0} \equiv \mathbf{0}.P$$

Replication law:

$$!a(x).P \equiv a(x).P|!a(x).P$$

The π-calculus is a very powerful concurrency model, which has been applied to design communication-based programming languages. It is the foundation of two of the main Web service process markup languages: BPML [25] from the BPMI consortium and XLANG [116] from Microsoft. In particular, XLANG was used as the internal processing language of the Microsoft BizTalk engine. BPEL4WS, the industrial standard language for composite Web services, drew much of its inspiration from the version of XLANG. The Semantic Web service markup, DAML-S ServiceModel, can be also adapted to π-calculus, although it does not announce π-calculus as its logical foundation.

The intended interpretation is that π-calculus describes a theory of processes concurrently communicating through distinguished ports and channels.

If we regard the Web services as processes, the invocation of those services is the message passing through their interfaces. In general, the following concurrent features of Web services result in that π-calculus is beneficial in presenting composite Web Services.

1. Processes in π-calculus are built in terms of synchronization constraints over I/O requests(messages), at a collection of channels. A channel is built on an output port and an input port. The function of a channel is transferring data from the output port to the input port. Whilst, the Web service notion provides an abstraction for autonomous computational entities that are based on the end-points and message-passing. End-points, presenting the service interfaces, are the ports for transferring messages. The service requester accesses a Web service by sending messages to the service's input end-points, and then, by fetching result from the output end-points.

2. π-calculus, unlike λ-calculus which is used to model the sequential single-thread computation, provides a general theory of concurrent interaction within and among multiple computational threads. This feature enables us to present the complex control activities in composite Web services, such as "sequence", "parallel" and "branch".

3. π-calculus has close relationship with LL, which is applied as the logical presentation of the available Web services in this thesis. The connection between LL and π-calculus was taken up formally by Abramsky [8], and further elaborated by Bellin and Scott [18]. Abramsky views proofs as processes. The key observation is that proof-theoretical communication (i.e. the Cut rule) is modeled by communication along an output port and an input port in the process-calculi world. Bellin and Scott give the formal translation between π-calculus and LL together with proving the soundness and completeness of the translation.

π-calculus is a general specification to model all kinds of concurrent systems. Considering the features of Web services, we can generalize some patterns that are specific for the concepts in Web service specification. The process calculus presented in this thesis is adapted from π-calculus by incorporating and rewriting the patterns. In the following, we list three kinds of patterns. They are channels, composite ports and services.

Channels

Firstly, we present how to use π-calculus to present the channels. To have a better understanding, we should distinguish the different notations when presenting the service from the provider's side and the requester side. From the provider's side, the Web services are concurrent processes and they are independent from each other. The services are invoked as soon as all their input ports receive data, and the service end after the data are sent to the output ports. The following example provides an illustration of two services, P and Q, and their presentation from the view of service provider's side.

$$(\nu a)(\nu b)inputP(a).P.\overline{outputP}\langle b\rangle \;\Big|\; (\nu c)(\nu d)inputQ(c).Q.\overline{outputQ}\langle d\rangle$$

We can see that the presentation from the service provider's side does not describe the connection between two services in an explicit way. The reason is that Web service are viewed independently from each other and the connection of the multiple services is not the concern of the service providers. Let's assume that a service requester may ask the above two services to be invoked in sequence, while the data outputed from service P is sent to service Q as input. Thus, the following process presented at the requester's side is different from that at the provider's side.

$$(\nu x)(\nu y)(\nu z)(inputP(x).P.\overline{outputP}\langle y\rangle.inputQ(y).Q.\overline{outputQ}\langle z\rangle)$$

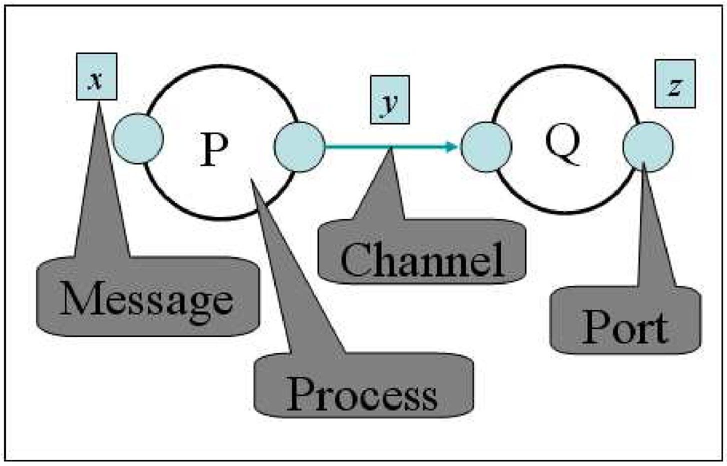


Figure 4.1: π-calculus flow graph.

This presentation emphasizes the message transferring between ports $outputP$ and $inputQ$, which denotes the workflow between two services that form the composite service. The workflow is illustrated in Figure 4.1. The process of workflow includes six sequential steps. First, the data stored in local variable x is sent to the input port $inputP$. Second, the process P is invoked. Third, the result is transmitted to the output port $outputP$ and is assigned to a local variable y. Fourth, the data in y is sent to the input port $inputQ$. Fifth, the process Q is invoked. Finally, the output from the $outputQ$ port is assigned to variable z. A channel between the port $outputP$ and $inputQ$ is defined as $\overline{outputP}\langle y\rangle.inputQ(y)$. In the channel, the message passing is represented by the sending/receiving actions in the fourth and fifth steps. The variable y is a local variable that indicates the data transferring between the $outputP$ port and the $inputQ$ port, both of which are global accessible. Because the local variable can be assigned to arbitrary name, we simply use a set of pair to include the correlated output and input parameters. Thus the above process can be rewritten as follows.

$$inputP.P.(\overline{outputP}inputQ).Q.\overline{outputQ}$$

We call $\overline{outputP}.inputQ$ a channel pair that denotes a connection between the output port $outputP$ and the input port $inputQ$. Each channel pair represents a means of synchronous interaction between two services. A channel pair represents a sequence of two actions. The first action reads the information from an output port, then the second action sends the same information to an output port of another services. The items in the pairs can be either ports or variables. For a variable a, $\overline{a}$ represents an action reading data from the variable, while a represents an action assigning a value to the variable.

The channel pair is not a standard presentation in π-calculus, although it can be rewritten using π-calculus formulae. Such presentation idea has been applied in many Web service process languages. For example, DAML-S uses property "sameValues" and Class "ValueOf" to state that the input to one subprocess should be the output of the previous one within a sequence. The presentation is enabled by a data set that connects one output parameter with one input parameter. XLANG uses a business process contract for stitching together the individual service descriptions using a map that defines the connections between the ports of the services involved. In XLANG, ports represent communication endpoints. Each port of each service must be mapped to at least one port of another service in such a way that the transport endpoints for the mapped ports are identical. In addition, a port that supports only outgoing operations must be mapped to exactly one incoming port of a partner service.

The channel pairs are the foundation to specify the dataflow within the generated composite service. What is believed to be one of the most important results of the automated service composition is the dataflow within the generated composite service. The method to generate the dataflow automatically is introduced in Section 4.4.

Composite ports

The second pattern is the presentation of composite ports. The composition of two ports a and b is presented in the form of (a, b). The port composition enables us to construct a new port that includes two parts. Each part is an existing port defined previously. The two parts are organized in an order that is defined by the initial ServiceProfile document. As we have introduced in Chapter 3, we use propositions to represent the parameters' types, and the addresses of the parameters are associated with the propositions as the proof terms. Thus one service may have several independent input and output ports. The composite port is a way to represent the multiple ports that belong to one service or one process. Moreover, we use $(a + b)$ to denote a new port that consists of two parts, but only one part is selected in execution. The selection is externally. It means which one is selected is not decided by the user. Instead, it depends on the execution result. An obvious example is the exception. A service may have two output ports, one of which is used to output the result if the service executes correctly, and another of which is used to output the exceptional information if the execution of the service fails. Since each service has only one exception and no other optional output defined in the upper ontology of service profile, an external choice includes only one $+$ symbol. The external choice is associated with the additive disjunction in LL.

The channel that is formed by two composite ports can be rewritten as a composite channel given the channel pairs are arranged in the same order as the ports. For example, a channel $\overline{(a_1, b_1, c_1)}(a_2, b_2, c_2)$ is equal to a composite channel $(\overline{a_1}a_2, \overline{b_1}b_2, \overline{c_1}c_2)$. The optional ports work in the same way. For example, the channel $\overline{(a_1 + b_1)}(a_2 + b_2)$ is equal to $\overline{a_1}a_2 + \overline{b_1}b_2$.

Services

Thirdly, we model all Web services as processes in the process calculus, where an atomic Web service is a process that cannot be decomposed. To distinguish the ports and services, we use the names starting with small letters refer to the ports, including the inputs, output, variables and exceptions. And the names starting with capital letters refer to the services in our setting. The names of services are available from the information that is provided in LL axioms.

Therefore, we present the syntax of the process calculus specification for Web services as follows. Comparing to the syntax of π-calculus at the begin of this section, this syntax is slightly different and strengthens the specific features considering the concepts of Web services.

$$
\begin{array}{rl}
\langle \mathit{Process} \rangle ::= & \langle \mathit{Inputs} \rangle . \langle \mathit{Process} \rangle \\
 & \big|\ \langle \mathit{Process} \rangle . \langle \mathit{Outputs} \rangle \\
 & \big|\ \langle \mathit{Process} \rangle . (\langle \mathit{Outputs} \rangle + \overline{\langle \mathit{Exception} \rangle}) \\
 & \big|\ \langle \mathit{Channels} \rangle . \langle \mathit{Process} \rangle \\
 & \big|\ (\nu \langle \mathit{Variable} \rangle) \langle \mathit{Process} \rangle \\
 & \big|\ \langle \mathit{Process} \rangle . \langle \mathit{Process} \rangle \\
 & \big|\ \langle \mathit{Process} \rangle | \langle \mathit{Process} \rangle \\
 & \big|\ \langle \mathit{Process} \rangle + \langle \mathit{Process} \rangle \\
 & \big|\ \langle \mathit{Service} \rangle \\
 & \big|\ \mathbf{0} \\
\langle \mathit{Inputs} \rangle ::= & (\langle \mathit{InputPort} \rangle , \langle \mathit{Inputs} \rangle) \\
 & |(\langle \mathit{InputPort} \rangle) \\
\langle \mathit{Outputs} \rangle ::= & (\overline{\langle \mathit{OutputPort} \rangle}, \langle \mathit{Outputs} \rangle) \\
 & |(\langle \mathit{OutputPort} \rangle) \\
\langle \mathit{Channels} \rangle ::= & (\langle \mathit{Channel} \rangle , \langle \mathit{Channels} \rangle) \\
 & |(\langle \mathit{Channel} \rangle + \langle \mathit{Channel} \rangle) \\
 & |(\langle \mathit{Channel} \rangle) \\
\langle \mathit{Variable} \rangle ::= & \langle \mathit{Port} \rangle \\
\langle \mathit{InputPort} \rangle ::= & \langle \mathit{Port} \rangle \\
\langle \mathit{OutputPort} \rangle ::= & \langle \mathit{Port} \rangle \\
\langle \mathit{Exception} \rangle ::= & \langle \mathit{Port} \rangle \\
\langle \mathit{Port} \rangle ::= & [!a - z][A - Za - z0 - 9\#]* \\
\langle \mathit{Service} \rangle ::= & [A - Z][A - Za - z0 - 9\#]* \\
\langle \mathit{Channel} \rangle ::= & \overline{\langle \mathit{OutputPort} \rangle} \langle \mathit{InputPort} \rangle \\
 & |\overline{\langle \mathit{OutputPort} \rangle} \langle \mathit{Variable} \rangle \\
 & |\overline{\langle \mathit{Variable} \rangle} \langle \mathit{InputPort} \rangle \\
 & |\overline{\langle \mathit{Exception} \rangle} \langle \mathit{Variable} \rangle \\
 & |\overline{\langle \mathit{Variable} \rangle} \langle \mathit{Variable} \rangle
\end{array}
$$

4.3 An Upper Ontology for the Process Model

A composite Web service can be viewed as a process that is described in terms of a process model. The process model, which details both the control structure and data flow structure of the service, is specified by a process language. There are a variety of process languages available, including the ServiceModel of the Semantic Web service markup, DAML-S; the industrial standard language for service process, BPEL4WS; and the formal concurrency modeling language π-

DAML-S	*BPEL4WS*	*π-calculus*	*Explanation*
Range	Type	Type(Sort)	data type definitions used to describe the exchanged messages
Data	Message	Message	an abstract definition of the data being transmitted
Process	Operation	Action	an abstract description of an action supported by the service
Range	PortType	Type(Sort)	a set of abstract operations. Each operation refers to an input message or output message
sameValues	Binding	Interaction	the concrete protocol and data format specifications for the operations and messages defined by a particular PortType
Parameter	Port	Port	an address for a binding, a single communication endpoint
Process	Services	Process	aggregation of a set of invocable applications

Table 4.1: The concepts in process specification.

calculus. These languages have a lot features in common. In Table 4.1, we give the mappings of some related concepts used in these process languages.

A successful Web service composition system should consider the different purpose of the above languages and enable the interoperation among these languages. In short, π-calculus, or the process calculus in this thesis, is used to specify the process formally. It is essential when generating the process from the proof in an automated way. DAML-S ServiceModel is the Semantic Web markup for the service process. It is naturally used to present the service process since we use DAML-S ServiceProfile to specify the interface of existing services. BPEL4WS is used as the grounding for DAML-S ServiceModel in order to support the invocation of generated composite service. Although people has noticed that DAML-S ServiceModel and BPEL4WS are not totally translatable to each other [79], we can enable the translation between these two languages by discarding some features that is not used in our automated service composition approach.

To enable the interoperation among the different sources of process languages, we design a process model that describe the service process in a highly abstract way. At conceptual level the process model is specified by an upper ontology in Figure 4.2. We do not expect the process model to serve for specifying all possible concepts in a wide array of services. Instead, the process model is a minimal set of the concepts of the process that the automated service composition method can generate. Moreover, the process model should be able to be specified by the process calculus formally as well as both by DAML-S and by BPEL4WS languages to present to the users.

The control flow in the process model is recursively defined by the class "ServiceProcess", that can be either "AtomicProcess" or "CompositeProcess". A "CompositeProcess" contains one or more sub processes. The sub processes are arranged in order and are executed in a given manner, which is indicated by the attribute of the type of control constructs, such as, "sequence", "choice" and "split". Each process has a set of parameters, including "inputs", "outputs", "variables" and an "exception". The "CompositeProcess" is also specified by its internal workflow activities. Such activities, including "Send, "Receive", "Connect" and "Copy" indicates the data transferring between the parameters.

More precisely, we summarize the concepts in the upper ontology as below. We will also explain their counterparts in DAML-S ServiceModel, BPEL4WS and the process calculus respectively.

ServiceProcess: A "ServiceProcess" refers to either an atomic Web service or a composite Web service. Accordingly, a "ServiceProcess" is either an "AtomicProcess" or a "CompositeProcess". An "AtomicProcess" is directly invocable and has no subprocesses. A "CompositeProcess" is a collection of "ServiceProcesses", each of which can be either "AtomicProcess" or "CompositeProcess". The "ServiceProcesses" that form a "CompositeProcess" are organized by using both "ControlActivities" and "Dataflow". The former indicates the ordering execution of the subprocesses, and the latter indicates end-to-end data exchanging. The term "ServiceProcess" is equal to the term "process" that is defined in DAML-S, in BPEL4WS and in the process calculus.

Parameters: "Parameters" include "Input", "Output", "Exception" and internal "Variable". Here, "Exception" can be regarded as a special kind of "Output", so both "Exception" and "Output" are defined as subclasses of a more general concept "AllOutput". The data exchange among the parameters is specified by "Dataflow" that we will discuss later. We present the "Parameters" as the value of a property of "ServiceProcess" class. DAML-S uses exact the same specification except that it does not support internal variable. One possible solution is to use variables as inputs or outputs inside an abstract process in DAML-S. BPEL4WS and the process calculus present the parameters in a similar matter presented above.

ControlActivity: A "CompositeProcess" includes a "ControlActivity" that indicates the ordering and the conditional execution of the subprocesses from which the "CompositeProcess" is composed. In the upper ontology,

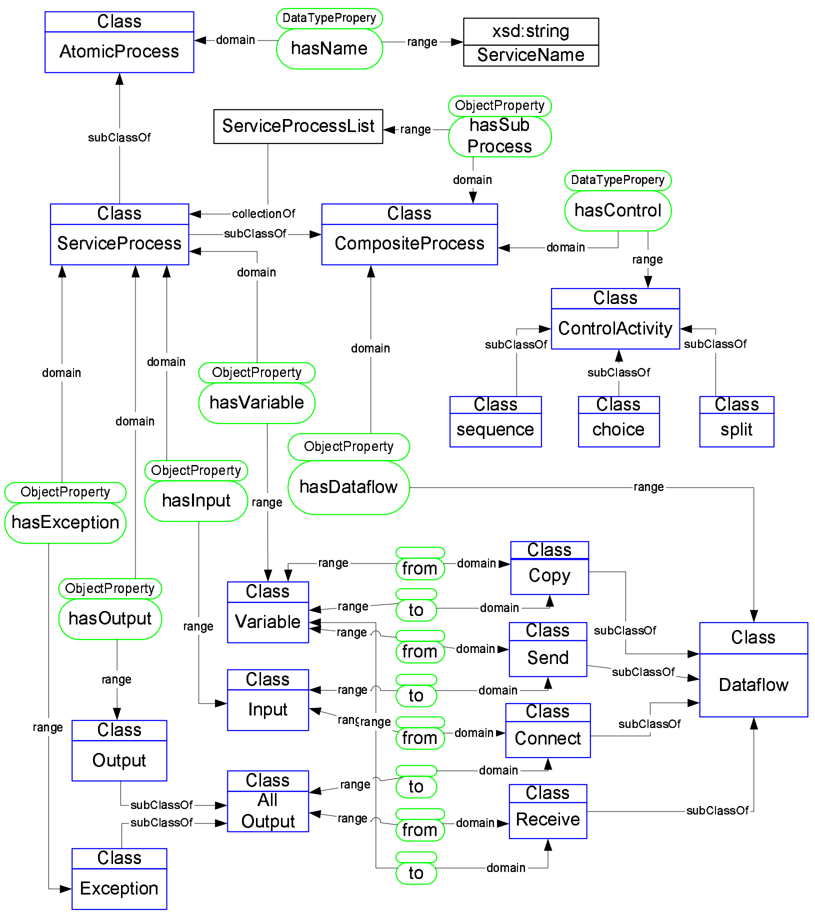


Figure 4.2: The upper ontology for the process model.

we have attempted to come up with a minimal set of activities that are potentially supported by the automated service composition approach. The activities consist of “sequence”, “split” and “choice”:

Sequence: allows us to define a collection of “ServiceProcesses” that are executed sequentially. This is equal to the “sequence” construct in both DAML-S and BPEL4WS. In the process calculus, this relation is denoted by sequential operator “.”.

Split: allows us to specify a collection of “ServiceProcesses” to be executed concurrently. DAML-S uses the same name for this activity. This activity is equal to the “flow” construct in BPEL4WS. In the process calculus, this is represented by parallel composition “|”.

Choice: allows us to select exactly one branch from a collection of “ServiceProcesses”. This selection is made externally by the program. DAML-S, although uses the same name, enables to select multiple processes. This activity can be represented by “switch” construct in BPEL4WS. In the process calculus, the choice operator “$+$” has the same function.

Dataflow: “Dataflow” defines the the data exchange inside the “CompositeProcess”. The “Dataflow” class is defined as a correlation pair which contains two parameters. These activities in the “Dataflow” include “Send”, “Receive”, “Connect” and “Copy”. “Send” emits the data stored in a local “Variable” to an “Input”. “Receive” takes the data from an “AllOutput” and assigns it to a local “Variable”. “Connect” defines that the data read from an “AllOutput” is sent to an “Input” in a synchronous manner. “Copy” assigns the data stored in one local “Variable” to another local “Variable”. We have discussed the method to represent the correlation pairs in both DAML-S and the process calculus. The “Send” and “Receive” activities are embedded in “invoke” construct in BPEL4WS, since the document specifies both the input variable and the output variable inside the “invoke” construct. “Assign” construct in BPEL4WS is equal to “Copy” activity. “Connect” activity is represented as the invocation of two services, if the value of an “Output” of one “ServiceProcess” is equal to the value of one “Input” of another “ServiceProcess”.

In the following, we define the grammar for the upper ontology of the process model. The concepts used in the grammar are exactly equal to what we have presented above. The grammar is specified by Extended BNF [91]:

$$
\begin{aligned}
\langle \mathit{ServiceProcess}\rangle &::= \langle \mathit{AtomicProcess}\rangle|\langle \mathit{CompositeProcess}\rangle \\
\langle \mathit{AtomicProcess}\rangle &::= ['(' \mathbf{hasName}\langle \%STRING\rangle ')'] \\
&\quad \{'('\mathbf{hasInput}\langle \mathit{Input}\rangle ')'\} \\
&\quad \{'('\mathbf{hasOutput}\langle \mathit{Output}\rangle ')'\} \\
&\quad \{'('\mathbf{hasException}\langle \mathit{Exception}\rangle ')'\} \\
\langle \mathit{CompositeProcess}\rangle &::= \{'('\mathbf{hasInput}\langle \mathit{Input}\rangle ')'\} \\
&\quad \{'('\mathbf{hasControl}\langle \mathit{ControActivity}\rangle ')'\} \\
&\quad \{'('\mathbf{hasSubProcess}\langle \mathit{ServiceProcessList}\rangle ')'\} \\
&\quad \{'('\mathbf{hasDataflow}\langle \mathit{Dataflow}\rangle ')'\} \\
&\quad \{'('\mathbf{hasOutput}\langle \mathit{Output}\rangle ')'\} \\
&\quad \{'('\mathbf{hasException}\langle \mathit{Exception}\rangle ')'\} \\
&\quad \{'('\mathbf{hasVariable}\langle \mathit{Variable}\rangle ')'\} \\
\langle \mathit{Input}\rangle &::= \langle \mathit{DAMLClass}\rangle \\
\langle \mathit{Output}\rangle &::= \langle \mathit{AllOutput}\rangle \\
\langle \mathit{Exception}\rangle &::= \langle \mathit{AllOutput}\rangle \\
\langle \mathit{ControlActivity}\rangle &::= \mathbf{sequence}|\mathbf{split}|\mathbf{choice} \\
\langle \mathit{ServiceProcessList}\rangle &::= '[' \langle \mathit{ServiceProcess}\rangle, \langle \mathit{ServiceProcessList}\rangle ']' \\
\langle \mathit{AllOutput}\rangle &::= \langle \mathit{DAMLClass}\rangle \\
\langle \mathit{Dataflow}\rangle &::= \langle \mathit{Copy}\rangle|\langle \mathit{Send}\rangle|\langle \mathit{Connect}\rangle|\langle \mathit{Receive}\rangle| \\
\langle \mathit{Copy}\rangle &::= '(' \mathbf{from}\langle \mathit{Variable}\rangle \mathbf{to}\langle \mathit{Variable}\rangle ')' \\
\langle \mathit{Send}\rangle &::= '(' \mathbf{from}\langle \mathit{Variable}\rangle \mathbf{to}\langle \mathit{Input}\rangle ')' \\
\langle \mathit{Connect}\rangle &::= '(' \mathbf{from}\langle \mathit{Input}\rangle \mathbf{to}\langle \mathit{AllOutput}\rangle ')' \\
\langle \mathit{Receive}\rangle &::= '(' \mathbf{from}\langle \mathit{AllOutput}\rangle \mathbf{to}\langle \mathit{Variable}\rangle ')' \\
\langle \mathit{Variable}\rangle &::= \langle \mathit{DAMLClass}\rangle
\end{aligned}
$$

Here we should emphasize that the upper ontology is not a new Web service process language. Instead, the upper ontology describes the concepts used to specify the service process and shows the connection of these concepts with their counterparts in other Web service languages, including the process calculus, DAML-S ServiceModel and BPEL4WS. Therefore, the process calculus that is extracted from the complete proof is able to be translated into the Web service languages for presentation or execution. An example of such translation will be presented in Section 4.5.

4.4 From the Proof to the Process Model

Here we propose a concrete method for the extraction of the process model presented in the process calculus from the completed proof. We do it by attaching proof terms to the deduction rules in the style of type theory. Each inference rule of the LL has been assigned a computational meaning. This makes the relationship of deduction rules and the process model more clear. It is also easier to be extended for dealing with any extension of the LL fragments.

A type theory has been defined for LL in [7]. In previous publication, proof terms are seen as programs in λ-calculus. We have, however, expressed the concern that λ-calculus does not provide enough expressive power for specifying the process of composite Web services, because it is not possible to present the concurrency of multiple Web services. Therefore, we choose the π-calculus-based process calculus in the proof terms.

Fortunately, people have concerned that the LL should have deeper connections with the parallelism and concurrent computation. This concern has further been taken up more formally by Abramsky [8] in an influential series of lectures. The Abramsky's view is essentially a modification of the formulae-as-types(Curry-Howard) isomorphism. In stead of proofs being functions presented by λ-calculus, Abramsky views proofs as processes(e.g. π-calculus or CCS terms). The Abramsky's view is to transfer the propositions as types paradigm to concurrency, so that concurrent processes, rather than functional programs, become the computational counterparts of proofs. The key observation of this view is that the cut-elimination in deductive inference is modeled by the construction of a channel for the communication along two ports in process-calculi world.

The Abramsky's translation between MALL rules and π-calculus is further discusses by Bellin and Scott [18]. In their paper, the authors give a detailed treatment of information flow in proof-nets and show how to mirror various evaluation strategies for proof normalization. The authors also give soundness and completeness results for the process-calculus translations of the MALL fragment. Bellin and Scott's translation is based on one-sided sequents in Classical Linear Logic(CLL), while the service presentation in this thesis is based on Intuitionistic Linear Logic. Considering this, we make some modification for Abramsky's translation to fit our presentation. Yet the results of Abramsky, Bellin and Scott can be applied to the intuitionistic version of MALL fragment, MAILL. As we have discussed in Chapter 3, MAILL is the fragment of LL that we use to present the existing Web services.

In the intuitionistic version of LL the cut rule is asymmetric. The left premise is distinguished from the right by the fact that the cut formula appears in the output position (i.e. as a result) in one, and in input position(i.e. as a resource) in the other. This is exactly reflected on the programming level by the fact that cut rule is interpreted by the operation of function composition(expressed syntactically as substitution).

We summarize the inference rules with the attached proof terms. Some rules are different from the MAILL inference rules introduced in Table 3.1. We discuss those differences in the following:

Firstly, "Shift" is not a default MAILL inference rule. But it can be proven from the inference rules in Table 3.1:

$$\dfrac{\dfrac{}{\Gamma \vdash A \multimap B}\,axiom \qquad \dfrac{\dfrac{}{A \vdash A}\,id \quad \dfrac{}{B \vdash B}\,id}{A, A \multimap B \vdash B}\,L\multimap}{\Gamma, A \vdash B}\,cut$$

Secondly, the composition ports are associated with the multiplicative conjunction $\otimes$ and disjunction &. Since, the propositions are used to represent the parameters' types, and the addresses of the parameters are associated with the propositions as the proof terms, a new proposition that composes two existing propositions by an operator is built when the introduction rule are applied during the inference. For example, $A \otimes B$ is a joint composition of propositions A and B. From the process point of view, the new proposition is a new port type, where the new port is a composition of two exist ports. Thus the formula $A(a) \otimes B(b)$ is equal to $A \otimes B(a, b)$.

Thirdly, the "choice" operator $+$ is associated with additive disjunction $\oplus$ in LL. From the process point of view, the introduction of additive disjunction leads to the construction of a composite port with external choice. Therefore, the formula $A(a) \oplus B(b)$ is equal to $A \oplus B(a + b)$

For attaching the process calculus formulae in the LL inference rules, we have to introduce a set of structural congruence rules that consider both the LL and the process calculus. The structural congruence rules used in the inference rules is presented as following.

Commutative law:

$$A \otimes B(a, b) \equiv B \otimes A(b, a) \qquad A\&B(a, b) \equiv B\&A(b, a)$$
$$(P + Q) : A \oplus B(a + b) \equiv (Q + P) : B \oplus A(b + a)$$

Associative law:

$$(A(a) \otimes B(b)) \otimes C(c) \equiv A(a) \otimes (B(b) \otimes C(c)) \equiv A \otimes B \otimes C(a, b, c)$$
$$(A(a)\&B(b))\&C(c) \equiv A(a)\&(B(b)\&C(c)) \equiv A\&B\&C(a, b, c)$$
$$(A(a) \oplus B(b)) \oplus C(c) \equiv A(a) \oplus (B(b) \oplus C(c)) \equiv A \oplus B \oplus C(a + b + c)$$

Interaction law:

$$\overline{(a_1, a_2, \ldots, a_n)}(b_1, b_2, \ldots, b_n) \equiv (\overline{a_1}b_1, \overline{a_2}b_2, \ldots, \overline{a_n}b_n)$$

Replication law:

$$!(A(a)) \equiv !A(!a)$$

The updated inference rules concerning the proof term are presented in Table 4.2. Each inference rule has a computational interpretation in the context of Web service composition process and the process calculus. In the following, we discuss the interpretation of each inference rule of MAILL fragment of LL we used in service presentation.

Logical axiom and Cut rule:

$$A(x) \vdash (\nu x)0 : A(x) \ \ (Id) \qquad \frac{\Gamma \vdash P : A(a_1) \quad \Gamma', A(a_2) \vdash Q : C}{\Gamma, \Gamma' \vdash (P.\{\overline{a_1}a_2\}.Q) : C}\ (Cut)$$

Rules for propositional constants:

$$\vdash 1 \qquad \frac{\Gamma \vdash A}{\Gamma, 1 \vdash A}$$

$$\frac{\Gamma, A(a), B(b) \vdash P : C}{\Gamma, A \otimes B(a,b) \vdash P : C}\ (L\otimes) \qquad \frac{\Gamma \vdash P : A(a) \quad \Gamma' \vdash Q : B(b)}{\Gamma, \Gamma' \vdash (P|Q) : A \otimes B(a,b)}\ (R\otimes)$$

$$\frac{\Gamma \vdash A(a) \multimap_P B(b)}{\Gamma, A(a) \vdash P : B(b)}\ (Shift) \qquad \frac{\Gamma, A(a) \vdash P : B(b)}{\Gamma \vdash A \multimap_{a.P.\overline{b}} B}\ (R \multimap)$$

$$\frac{\Gamma, A(a) \vdash P : C(c_1) \quad \Gamma, B(b) \vdash Q : C(c_2)}{\Gamma, A \oplus B(a+b) \vdash (P+Q) : C(c_1 + c_2)}\ (L\oplus)$$

$$\frac{\Gamma \vdash P : A(a)}{\Gamma \vdash P : A \oplus B(a)}\ (R\oplus)(a) \qquad \frac{\Gamma \vdash Q : B(b)}{\Gamma \vdash Q : A \oplus B(b)}\ (R\oplus)(b)$$

$$\frac{\Gamma, A(a) \vdash \Delta}{\Gamma, A\&B(a,0) \vdash \Delta}\ (L\&)(a) \qquad \frac{\Gamma, B(b) \vdash \Delta}{\Gamma, A\&B(0,b) \vdash \Delta}\ (L\&)(b)$$

$$\frac{\Gamma \vdash P : A(a) \quad \Gamma \vdash Q : B(b)}{\Gamma \vdash (P|Q) : A\&B(a,b)}\ (R\&)$$

Rules for exponential !:

$$\frac{\Gamma \vdash \Delta}{\Gamma, !A(0) \vdash \Delta}\ (W!) \qquad \frac{\Gamma, A(a) \vdash \Delta}{\Gamma, !A(!a) \vdash \Delta}\ (L!) \qquad \frac{\Gamma, !A(!a), !A(!a) \vdash \Delta}{\Gamma, !A(a) \vdash \Delta}\ (C!)$$

Table 4.2: Inference rules for process extraction.

id The identity corresponds to a transition to the same parameter type, and it requires no action. So the corresponding process is empty, 0.

cut The cut rule corresponds to the sending and receiving operations in the process calculus. It results in the sequential composition of two process. In the composition, the output of one process is connected to the input of the other process.

$L\otimes$ The $L\otimes$ rule generates a new port that is the composition of two separate ports.

$R\otimes$ The $R\otimes$ rule generates a new process that consists of two sub-processes in parallel.

$shift$ Initially, a Web service is specified by an axiom in the form of $\vdash A(a) \multimap_p B(b)$. Here A and B are sets of input and output parameters, while p is the name of the service. The $shift$ rule changes the initial presentation into a sequent that can be used for the future proof.

$R \multimap$ The $R \multimap$ reverses the change made by $shift$ rule.

$L\oplus$ The $L\oplus$ introduces an external choice of two processes.

$R\oplus$ The $R\oplus$ indicates if the port a has type A, it also has type $A \oplus B$. This is used in cut elimination with an external choice.

$L\&$ The $L\&$ produces a redundant input parameter that is not used by the process. However, it can be used in cut elimination with an internal choice.

$R\&$ The $R\&$ introduces an internal choice of two processes.

$W!$ The $W!$ presents a logical weakening. It is used when an output parameter is discarded.

$L!$ The $L!$ presents dereliction. It is used for explicit presentation of the information duplication.

$C!$ The $C!$ presents contraction. It is used to discarding the multiple use of a single piece of information.

After applying the above inference rules, the complete proof contains a proof term that is specified by the process calculus introduced in Section 4.2. The calculus should be translated into Web service process language for execution. In the Section 4.3, we have shown the relation between the upper ontology and the Web service process languages, including DAML-S ServiceModel and BPEL4WS. Therefore, if we develop the translation from the process calculus to the upper ontology, we can translate it to other Web service process languages.

The syntax of both the process calculus and the upper ontology have already been introduced in the previous sections. Here, we define a translation $\mathcal{R}$ from the process calculus to the specification of the upper ontology. Considering the syntax of the specification languages, a translation is defined by the following:

$$
\begin{aligned}
\langle Process\rangle^{\mathcal{R}} \equiv\ & \mathbf{if}(\langle Inputs\rangle.\langle Process\rangle)^{\mathcal{R}}\mathbf{then}\langle Process\rangle^{\mathcal{R}}\langle Inputs\rangle^{\mathcal{R}}\\
& \mathbf{if}(\langle Process\rangle.\langle Outputs\rangle)^{\mathcal{R}}\mathbf{then}\langle Process\rangle^{\mathcal{R}}\langle Outputs\rangle^{\mathcal{R}}\\
& \mathbf{if}(\langle Channels\rangle.\langle Process\rangle)^{\mathcal{R}}\mathbf{then}\langle Process\rangle^{\mathcal{R}}\langle Channels\rangle^{\mathcal{R}}\\
& \mathbf{if}(\langle Process\rangle.(\langle Outputs\rangle + \langle Exception\rangle)^{\mathcal{R}}\mathbf{then}\langle Process\rangle^{\mathcal{R}}\langle Outputs\rangle^{\mathcal{R}}{}'\langle'\mathbf{hasException}\langle Exception\rangle^{\mathcal{R}}{}'\rangle'\\
& \mathbf{if}(\nu\langle Variable\rangle.\langle Process\rangle)^{\mathcal{R}}\mathbf{then}\langle Process\rangle^{\mathcal{R}}{}'\langle'\mathbf{hasVariable}\langle Variable\rangle^{\mathcal{R}}{}'\rangle'\\
& \mathbf{if}(\langle Process\rangle.\langle Process\rangle)^{\mathcal{R}}\mathbf{then}'\langle'\mathbf{hasControl\,sequence}'\rangle''\langle'\mathbf{hasSubProcess}'['\langle Process\rangle^{\mathcal{R}},\langle Process\rangle^{\mathcal{R}}{}']''\rangle'\\
& \mathbf{if}(\langle Process\rangle|\langle Process\rangle)^{\mathcal{R}}\mathbf{then}'\langle'\mathbf{hasControl\,split}'\rangle''\langle'\mathbf{hasSubProcess}'['\langle Process\rangle^{\mathcal{R}},\langle Process\rangle^{\mathcal{R}}{}']''\rangle'\\
& \mathbf{if}(\langle Process\rangle + \langle Process\rangle)^{\mathcal{R}}\mathbf{then}'\langle'\mathbf{hasControl\,choice}'\rangle''\langle'\mathbf{hasSubProcess}'['\langle Process\rangle^{\mathcal{R}},\langle Process\rangle^{\mathcal{R}}{}']''\rangle'\\
& \mathbf{if}(\langle Service\rangle)^{\mathcal{R}}\mathbf{then}'\langle'\mathbf{hasName}\,\textit{the string of service name}'\rangle'\\
& \mathbf{if}(0)^{\mathcal{R}}\mathbf{then}\,\textit{do nothing}\\
(\langle Inputs\rangle)^{\mathcal{R}} \equiv\ & '\langle'\mathbf{hasInput}\langle InputPort\rangle^{\mathcal{R}\prime}\rangle'\langle Inputs\rangle^{\mathcal{R}}\\
(\langle Outputs\rangle)^{\mathcal{R}} \equiv\ & '\langle'\mathbf{hasOutput}\langle OutputPort\rangle^{\mathcal{R}\prime}\rangle'\langle Outputs\rangle^{\mathcal{R}}\\
(\langle Channels\rangle)^{\mathcal{R}} \equiv\ & \mathbf{if}(\langle Channel\rangle,\langle Channels\rangle)\mathbf{then}'\langle'\mathbf{hasDataflow}\langle Channel\rangle^{\mathcal{R}\prime}\rangle'\langle Channels\rangle^{\mathcal{R}}\\
& \mathbf{if}(\langle Channel\rangle + \langle Channel\rangle)\mathbf{then}'\langle'\mathbf{hasDataflow}\langle Channel\rangle^{\mathcal{R}\prime}\rangle''\langle'\mathbf{hasDataflow}\langle Channel\rangle^{\mathcal{R}\prime}\rangle'\\
(\langle InputPort\rangle)^{\mathcal{R}} \equiv\ & \langle Port\rangle^{\mathcal{R}}\\
(\langle OutputPort\rangle)^{\mathcal{R}} \equiv\ & \langle Port\rangle^{\mathcal{R}}\\
(\langle Exception\rangle)^{\mathcal{R}} \equiv\ & \langle Port\rangle^{\mathcal{R}}\\
(\langle Variable\rangle)^{\mathcal{R}} \equiv\ & \langle Port\rangle^{\mathcal{R}}\\
\langle Channel\rangle^{\mathcal{R}} \equiv\ & '\langle'\mathbf{from}\langle Port\rangle^{\mathcal{R}}\mathbf{to}\langle Port\rangle^{\mathcal{R}\prime}\rangle'\\
\langle Port\rangle^{\mathcal{R}} \equiv\ & \mathbf{the\ string\ of\ port\ name}
\end{aligned}
$$

4.5 An Example Proof

In this Section we show an example about how the process is generated from the service presentation. We reuse the example that has already been shown in Chapter 3. To make the proof simple and easy to read, we discard the non-functional attributes in the example, but the proving process is the same after introducing the non-functional attributes.

Applied to our motivating example, the predefined axioms and the goal to be proven can be presented as follows:

Axioms:

$$
\Gamma = \begin{array}{l}
\vdash PRICE\_LIMIT(smp) \otimes SKILL\_LEVEL(sms) \multimap_{SelectModel} BRAND(smb) \otimes MODEL(smm)\\
\vdash HEIGHT\_CM(slh) \otimes WEIGHT\_KG(slw) \multimap_{SelectLength} LENGTH\_CM(sll)\\
\vdash LENGTH\_CM(cic) \multimap_{CM2INCH} LENGTH\_IN(cii)\\
\vdash PRICE\_USD(unu) \multimap_{USD2NOK} PRICE\_NOK(unn)\\
\vdash LENGTH\_IN(ssl) \otimes BRAND(ssb) \otimes MODEL(ssm) \multimap_{SelectSki} PRICE\_USD(ssp) \oplus EXCEPTION(sse)
\end{array}
$$

Goal/Theorem

$$\vdash PRICE\_LIMIT \otimes SKILL\_LEVEL \otimes HEIGHT\_CM \otimes WEIGHT\_KG \multimap PRICE\_NOK \oplus EXCEPTION$$

Using the inference rules from the Table 4.2 the proof is as shown in Figure 4.3. In order to make the proof easier to read, we use abbreviations to represent the propositions. Here, PL, SL, BR, MO, HC, WK, LC, LI, PU, PN and EX stand for PRICE_LIMIT, SKILL_LEVEL, BRAND, MODEL, HEIGHT_CM, WEIGHT_KG, LENGTH_CM, LENGTH_IN, PRICE_USD, PRICE_NOK and EXCEPTION, respectively.

The generated process calculus formula is presented as follows:

$$(\nu x_1)(smp,sms,slh,slw).((SelectModel|SelectLength.\overline{sll}cic.CM2INCH)$$
$$.(\overline{smb}ssb,\overline{smm}ssm,\overline{cii}ssl).SelectSki.(\overline{ssp}unu+\overline{sse}x_1).USD2NOK.\overline{unn+x_1}$$

The above formula of the process calculus can be translated into the following code in the language of the upper ontology using the translation mechanism introduced above:

```
<hasInput smp>
<hasInput sms>
<hasInput slh>
<hasInput slw>
<hasControl sequence>
<hasSubProcess [
  <hasControl split>
  <hasSubProcess [
    <hasControl sequence>
    <hasSubProcess [SelectLength, CM2INCH]>
    <hasDataflow from sll to cic>,
    SelectModel
  ]
<hasDataflow from smb to ssb>
<hasDataflow from smm to ssm>
<hasDataflow from cii to ssl>
<hasDataflow from smb to ssb>
<hasDataflow from ssp to unu>
<hasDataflow from sse to x1>
<hasOutput unn>
<hasException x1>
<hasVariable x1>
]>
```

$$\dfrac{\dfrac{\dfrac{\vdash HC(slh)\otimes WK(slw)\multimap_{SelectLength}\ LC(sll)}{HC(slh)\otimes WK(slw)\vdash SelectLength : LC(sll)}\ Shift \qquad \dfrac{\vdash LC(cic)\multimap_{CM2INCH}\ LI(cii)}{LC(cic)\vdash CM2INCH : LI(cii)}\ Shift}{HC(slh)\otimes WK(slw)\vdash SelectLength.\overline{sll}cic.CM2INCH : LI(cii)}\ cut}{HC\otimes WK(slh,slw)\vdash SelectLength.\overline{sll}cic.CM2INCH : LI(cii)}\ Structural\,congruence \tag{4.1}$$

$$\cdots$$

$$\dfrac{\dfrac{\dfrac{\vdash PL(smp)\otimes SL(sms)\multimap_{SelectModel}\ BR(smb)\otimes MO(smm)}{PL(smp)\otimes SL(sms)\vdash SelectModel : BR(smb)\otimes MO(smm)}\ Shift}{PL\otimes SL(smp,sms)\vdash SelectModel : BR\otimes MO(smb,smm)}\ Structural\,congruence}{PL\otimes SL\otimes HC\otimes WK(smp,sms,slh,slw)\vdash (SelectModel|SelectLength.\overline{sll}cic.CM2INCH) : BR\otimes MO\otimes LI(smb,smm,cii)}\ R\otimes(with(4.1)) \tag{4.2}$$

$$\cdots$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash PU(unu)\multimap_{USD2NOK}\ PN(unn)}{PU(unu)\vdash USD2NOK : PN(unn)}\ Shift}{PU(unu)\vdash USD2NOK : PN\oplus EX(unn)}\ R\oplus(a) \qquad \dfrac{\dfrac{}{EX(x_1)\vdash(\nu x_1)0 : EX(x_1)}\ id}{EX(x_1)\vdash(\nu x_1)0 : PN\oplus EX(x_1)}\ R\oplus(b)}{PU\oplus EX(unu+x_1))\vdash(\nu x_1)(USD2NOK+0) : PN\oplus EX(unn+x_1)}\ L\oplus}{PU\oplus EX(unu+x_1)\vdash(\nu x_1)(USD2NOK) : PN\oplus EX(unn+x_1)}\ Structural\,Congruence \tag{4.3}$$

$$\cdots$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash BR(ssb)\otimes MO(ssm)\otimes LI(ssl)\multimap_{SelectSki}\ PU(ssp)\oplus EX(sse)}{BR(ssb)\otimes MO(ssm)\otimes LI(ssl)\vdash SelectSki : (PU(ssp)\oplus EX(sse))}\ Shift}{BR\otimes MO\otimes LI(ssb,ssm,ssl)\vdash SelectSki : PU\oplus EX(ssp+sse)}\ StructualCongruence}{BR\otimes MO\otimes LI(ssb,ssm,ssl)\vdash(\nu x_1)(SelectSki.\overline{ssp+sse}(unu+x_1).USD2NOK) : PN\oplus EX(unn+x_1)}\ cut(with(4.3))}{PL\otimes SL\otimes HC\otimes WK(smp,sms,slh,slw)\vdash(\nu x_1)((SelectModel|SelectLength.\overline{sll}cic.CM2INCH).\overline{(smb,smm,cii)}(ssb,ssm,ssl).SelectSki.\overline{ssp+sse}(unu+x_1).USD2NOK) : PN\oplus EX(unn+x_1)}\ cut(with(4.2))}{PL\otimes SL\otimes HC\otimes WK(smp,sms,slh,slw)\vdash(\nu x_1)((SelectModel|SelectLength.\overline{sll}cic.CM2INCH).(\overline{smb}ssb,\overline{smm}ssm,\overline{cii}ssl).SelectSki.(\overline{ssp}unu+\overline{sse}x_1).USD2NOK) : PN\oplus EX(unn+x_1)}\ StructrualCongruence}{\vdash PL\otimes SL\otimes HC\otimes WK\multimap_{(\nu x_1)(smp,sms,slh,slw).((SelectModel|SelectLength.\overline{sll}cic.CM2INCH).(\overline{smb}ssb,\overline{smm}ssm,\overline{cii}ssl).SelectSki.(\overline{ssp}unu+\overline{sse}x_1).USD2NOK.\overline{unn+x_1}}\ PN\oplus EX}\ R\multimap \tag{4.4}$$

Figure 4.3: The example proof.

At the conceptual level, the above code is equal to the process illustrated in Figure 4.4. The figure uses the denotation of concepts presented in the process upper ontology. Therefore, it can be further translated into DAML-S or BPEL4WS by the translation framework we proposed before.

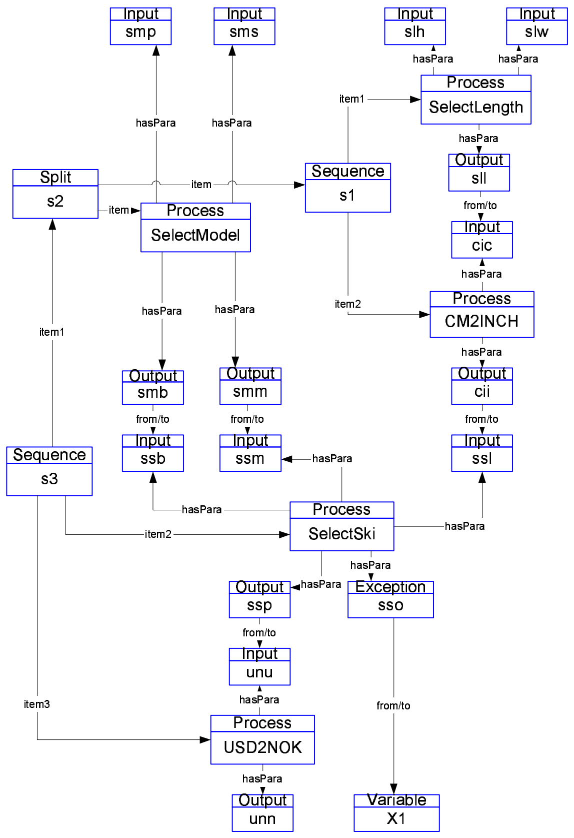


Figure 4.4: The illustration of result process for example.

4.6 Summary

This Chapter presents the approach to extract the process description of the generated composed service from the complete LL proof. In order to do so we need to extend the inference rules in Table 3.1 with terms. We study the LL inference rules from the viewpoint of giving each rule a concrete computational interpretation in the context of composite Web services. The updated inference rules supplied with proof terms enable us to construct the process at each inference step.

Because LL has deep connections with parallelism and concurrent computation, it is natural to use LL proof terms to present the process calculi that is used to model Web services, which has put much concern in concurrency. In this chapter, we use a π-calculus based process calculus to present the proof terms. The process calculus is attached to the LL inference rules in the style of type theory. A complete process presented by the process calculus can be extracted from the complete proof.

We use the process calculus to present the composite service formally, but the final result is presented to the users in DAML-S ServiceModel or BPEL4WS languages. We use both DAML-S and BPEL4WS here because DAML-S lacks the support for service invocation. Therefore, we use BPEL4WS as the grounding of DAML-S ServiceModel. To enable the translation among the process calculus, DAML-S and BPEL4WS, we propose an upper ontology that is general enough to cover the essential aspects of the different languages, thus the upper ontology can be used as a mediator for translation.

The composite service that satisfies the customer's request is generated only if the theorem prover can find a service with an input that has the exactly same type as the output of another existing service. This puts too much restriction because different DAML classes for representing the parameter types may refer to the same concept. In addition, if the type of an output parameter of one service is the subtype of the type of an input parameter of another service, it is also safe to transfer data from the output to the input. In the next Chapter, we will discuss how to apply the Semantic Web information to enable a more flexible composition of Web services.

Chapter 5

Semantic Web Service Composition

5.1 Introduction

In the previous Chapter, we have introduced a Web service composition approach based on LL proof. A general idea of such approach is inspired by the cut rule in LL inference. Since we use LL proposition to refer to the the classes in DAML+OIL (also called DAML classes) that represent the type of parameters, the computational meaning of the cut rule is: if an output port of one service is restricted to the same DAML class as an input port of another service, the output port and the input port can form a channel pair. A channel pair represents a means of synchronous interaction between two services. In the interaction, the information is read from an output port, then the same information is sent to the input port.

The above method should obey a very restricted constraint that the input and the output have to be restricted to the same DAML class. Actually, the researches in Object-Oriented programming have provided the support of subclass polymorphism, where when a value or instance of class A is expected, a value or instance of any subclass B of A can also be accepted [15]. As a result, we can also model subsumption-based semantic inferencing as it has been done in Object-Oriented programming.

The classes in DAML-S are defined through DAML+OIL language, which owes to its foundations in RDF Schema, providing a typing mechanism for Web resource. DAML+OIL enables some measure of semantic inferencing to be made by an semantic reasoner. Here, we model subsumption-based semantic inferencing as it has been done in Object-Oriented programming. The major difference between a DAML class and a class in a typical Object-Oriented programming language is that DAML classes are meant primarily for data modeling and they contain no methods.

The subsumption inference of DAML+OIL language is possible because DAML+OIL is an XML binding to a formal Description Logic(DL) model [51]. DLs are a general class of logic designed to model vocabularies and their relationships. Several DL algorithms have been developed to ensure logical consistency of the model developed by DAML+OIL language and to answer logic queries including satisfiability, equivalence and subsumption.

In this Chapter, we will introduce the class subsumption into the Web service composition system. In general, we will discuss three techniques to solve such kind of problem.

1. Firstly, we describe the subsumption inference for DAML+OIL language. Since DAML+OIL corresponds to a fragment of DL, the DL reasoner can be used to detect the subsumption relationships between each two DAML class used in describing the parameter types of the Web services;

2. After the subsumption relationships are known, the next problem is how to present the subsumption by LL so that the results of the semantic reasoner can be used in LL proof. We will prove that the subsumption can be presented as LL implication, thus the Web service composition method introduced in the last Chapter can also deal with subclass polymorphism;

3. Finally, we will design a concrete interaction protocol between the semantic reasoner and the LL theorem prover. The two parts are implemented separately and used for different purposes. The correctness and the efficiency for the interaction will be also discussed. This is elaborated in the next Chapter instead of this Chapter.

The rest of this chapter is organized as follows. Section 5.2 introduce the basic concepts and languages about Semantic Web. Section 5.3 presents the relationship between DAML+OIL language and DL. This Section also introduces how subsumption inference is enabled by DL. Section 5.4 introduces how to present the subsumption relationships by the subtyping rules for LL. At last, the Chapter is summarized in Section 5.5.

5.2 Semantic Web and Languages

Today's Web enables people to access documents and services on the Internet. Because the interface to services is represented in Web pages written in natural language, today's methods require human intelligence to understand the Web pages. The *Semantic Web* is an extension of the current Web in which information is given well-defined meaning, aiming for machine-understandable Web resources, whose information can be shared and processed both by automated tools, such as software agents, and by human users. The vision of the Semantic Web was first introduced by Tim Berners-Lee [20]. An example in [21] illustrated how the Semantic Web might be useful. "Suppose you want to compare the price and choose flower bulbs that grow best in your living area given zip

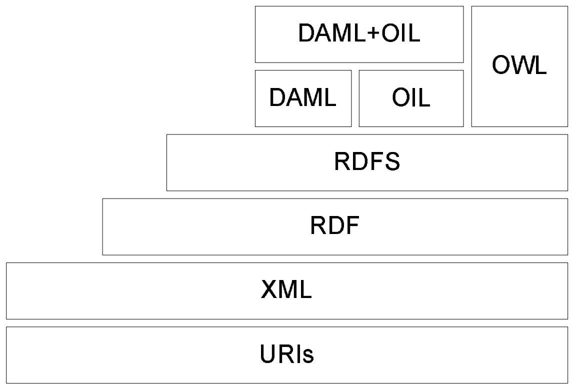


Figure 5.1: The layer of data representation standards for the Semantic Web

code, or you want to search online catalogs from different manufactures for equivalent replacement parts for a Volvo 740. The raw information that may answer these questions, may indeed be on the Web, but it is not in a machine-usable form. You still need a person to discern the meaning of the information and its relevances to your needs”.

The Semantic Web addresses this problem in two ways. First, it will enable communities to expose their data so that a program does not have to strip the formatting, pictures and ads from a Web page to guess at the relevant bits of information. Second, it will allow people to write (generate) files which explain - to a machine - the relationships between different sets of data. For example, one will be able to make a ”semantic link” between a database with a ”zip-code” column and a form with a ”zip” field that they actually mean the same thing. This will allow machines to follow links and facilitate the integration of data from many different sources.

To make sure that different users have a common understanding of the terms, one needs ontologies in which these terms are described, and which establish a joint terminology between the Semantic Web users. The meanings of the vocabularies are defined explicitly by ontologies. The word ”ontology” [46] seems to generate some controversy in the published literatures. It has a long history in philosophy, in which it refers to the subject of existence.

In the context of knowledge sharing, the term ontology is used to mean a specification of a conceptualization. That is, an ontology is a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents. This definition is consistent with the usage of ontology as set-of-concept-definitions, but more general. And it is certainly a different sense of the word than its use in philosophy.

The Semantic Web is designed to be built on layers of enabling standards. Figure 5.1 shows their relationships. In general, the Semantic Web languages include the following standards:

- Uniform Resource Identifiers (URIs) is a fundamental component of current Web, which provides the ability to uniquely identify resources as well as relations among resources. The symbol of a URI includes two parts: an XML namespace and a vocabulary. XML namespace are used to distinguish the same vocabularies of different symbols that defined in different documents.

- eXtensible Markup Language (XML) [26] is a fundamental component for syntactical interoperability on Web. XML is the universal format for structured documents and data on the Web. XML itself is not an ontology language, but XML-Schemas, which define the structure, constraints and the semantics of XML documents, can to some extent, be used to specify ontologies.

- The Resource Description Framework (RDF) [65] family of standards leverages URI and XML to allow documents being described in the form of metadata by means of resources (subjects, available or imaginable entities), properties (predicates, describing the resources), and statements (the object, a value assigned to a property in a resource).

- RDF Schema (RDFS) [27] is an extension of RDF, which defines a simple modeling language on top of RDF. RDFS enables the representation of class, property and constraint while RDF allows the representation of instances and facts, thus making it a qualified lightweight ontology language. While RDF and RDFS are different, they are combined together to form the basic language for knowledge representation denoted as RDF(S).

- Other Web based ontology specification languages, such as DAML+OIL and OWL use the RDF(S) as a starting point and extend it to the more expressive powered ontology specification languages for the description of the Semantic Web. The different languages are developed initially by different communities, but the merging of the languages to a universal language is undertaken by W3C. The knowledge sources presented by Web based ontology are openly available and decentralized, typically using HTTP as an access mechanism.

In the following, we introduce the Semantic Web standards and languages in greater detail.

OIL(Ontology Inference Layer) [40] is an initiative funded by the European Union programme for Information Society Technologies as part of the On-To-Knowledge project. OIL is both a representation and exchange language for ontologies. The language synthesized work from different communities (modeling primitives from frame-based languages; semantics of the primitive defined by Description Logic; and XML syntax) to achieve the aim of providing a general-purpose markup language for the Semantic Web. OIL is also compatible with RDF(S) as it is defined as an extension of RDF(S). The language is defined in a layered approach. The three layers are: Standard OIL (mainstream modeling primitives usually found in ontology language), Instance OIL (includes individual into the ontology) and Heavy OIL (not yet defined, but aims at additional reasoning capabilities). OIL provides a predefined set of axioms (like disjoint class, covering etc.) but does not allow defining arbitrary axioms.

DAML+OIL [54,2] is a product of efforts in merging two language - DAML (DARPA Agent Modeling Language) and OIL. Usually we also simply call it as DAML language. DAML+OIL is a language based on RDF(S) with richer modeling primitives. In general, what DAML+OIL adds to RDF Schema is the additional ways to constrain the allowed values of properties, and what properties a class may have. The differences between OIL and DAML+OIL are subtle, as the same effect can be achieved by using different construct of the two languages (For instance, DAML+OIL has no direct equivalent to OIL's ”covered” axiom, however, the same effect can be achieved using a combination of ”unionOf” and ”subClass”) . In addition, DAML+OIL has better compatibility with RDF(S) (for instance, OIL has explicit ”OIL” instances, while DAML+OIL relies on RDF for instance). DAML+OIL is also a proposed W3C recommendation for semantic markup language for web resources.

OWL (Web Ontology Language). OWL [34] is a semantic markup language for publishing and sharing ontologies on the web. OWL is the latest W3C proposed recommendation for that purpose. The language incorporates learning from the design and application of DAML+OIL. OWL has three increasingly-expressive sublanguages, namely, OWL Lite (Classification hierarchy and simple constraints), OWL DL (adding class axioms, Boolean combinations of class expression and arbitrary cardinality) and OWL Full (Permits also meta-modeling facilities in RDF(S)). Ontology developers should consider which sublanguage best suits their needs. The choice between OWL Lite and OWL DL depends on the extent to which users require the more-expressive constructs provided by OWL DL. The choice between OWL DL and OWL Full mainly depends on to which extend the users require the meta-modeling facilities of RDF Schema. The reason why OWL DL contains the full vocabulary but restricts how it may be used is to provide logical inference engines with certain properties desirable for optimization.

Currently, both DAML+OIL and OWL are W3C recommendations for Semantic Web markup languages. Although it seems that the the trend is OWL will overwhelm DAML+OIL and become a universal language, DAML+OIL and OWL is very similar in syntax and they can replace each other in most cases. In this chapter, we will only discuss the semantic reasoning for DAML+OIL because we built the Web service composition system by DAML+OIL language from the beginning. Because both DAML+OIL and OWL use DLs as the logical foundation, the semantic reasoning techniques for DAML+OIL could apply to OWL as well [52].

5.3 Semantic Reasoning for DAML+OIL

DAML+OIL is designed to describe the structure of a domain. It takes an object oriented approach that describes the structure in terms of classes and properties. An ontology consists of a set of axioms that assert, e.g., subsumption relationships between classes and properties. The subsumption relationships are defined through the property `rdfs:subClassOf` and `rdfs:subPropertyOf`. From a formal point of view, DAML+OIL can be seen to be equivalent to an expressive Description Logic(DL), with a DAML+OIL ontology corresponding to a DL terminology. As in a DL, DAML+OIL classes can be names (URIs) or expressions, and a variety of constructors are provided for building class expressions. The expressive power of the language is determined by the class(and property) constructors supported, and by the kinds of axiom supported.

Description Logics(DLs), a family of logical formalisms for the representation of and reasoning about conceptual knowledge, are of crucial importance to the development of Semantic Web. Their role is to provide formal underpinnings and automated reasoning services for Semantic Web ontology languages such as OIL, DAML+OIL and OWL.

DLs include several fragments, each of which provides different expressive power. In general, DAML+OIL equals to the $\mathcal{SHIQ}(\mathcal{D})$ fragment of DL. In the leftmost three columns of Table 5.1, we summarize the terms in the fragment of DLs and their counterparts in DAML+OIL.

An important feature of DAML+OIL is that, besides "abstract" classes defined by the ontology, one can also use XML Schema Datatype (e.g., so called primitive datatype such as string, decimal or float, as well as more complex derived datatypes such as integer sub-ranges) as a special class. The domain of XML Schema Datatype is represented by Δ_D. This feature is enabled by $(\mathcal{D})$ fragment of DLs. The reasoning issue of datatypes is introduced in [96].

The interpretations of the DLs terms is the basis for the reasoning of DLs. An interpretation $\mathcal{I}$ $=(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a set $\Delta^{\mathcal{I}}$, presenting the set of all instances in the domain, and a function $\cdot^{\mathcal{I}}$, which maps every class to a subset of $\Delta^{\mathcal{I}}$ and every property to a setset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. For example, the property *hasBook* takes an object class *Person* and a subject class *Book*. Here, a class C is interpreted by a set $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ if all instances of class C are also instances of class D. The interpretations of all term in $\mathcal{SHIQ}(\mathcal{D})$ are summarized in the rightmost column of Table 5.1.

We can define the reasoning tasks for DAML+OIL in term of its interpreta-

	DL Syntax	*DAML+OIL Syntax*	*Example*
	C	daml:Class	$C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
	P	daml:Property	$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
	$\top$	daml:Thing	$\Delta^{\mathcal{I}}$
	$\bot$	daml:Nothing	$\emptyset$
	$C_1 \sqcap C_2$	daml:intersectionOf	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
$\mathcal{S}$	$C_1 \sqcup C_2$	daml:disjointUnionOf	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
	$\neg C$	daml:complementOf	$\Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$
	$C_1 \sqsubseteq C_2$	daml:subClassOf	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
	$C_1 \equiv C_2$	daml:sameClassAs	$C_1^{\mathcal{I}} = C_2^{\mathcal{I}}$
	$\forall P.C$	daml:toClass	$\{x_1 \mid \forall x_2.\langle x_1, x_2\rangle \in P^{\mathcal{I}} \rightarrow x_2 \in C^{\mathcal{I}}\}$
	$\exists P.C$	daml:hasClass	$\{x_1 \mid \exists x_2.\langle x_1, x_2\rangle \in P^{\mathcal{I}} \wedge x_2 \in C^{\mathcal{I}}\}$
$\mathcal{H}$	$P_1 \sqsubseteq P_2$	daml:subPropertyOf	$P_1^{\mathcal{I}} \subseteq P_2^{\mathcal{I}}$
	$P_1 \equiv P_2$	daml:samePropertyOf	$P_1^{\mathcal{I}} = P_2^{\mathcal{I}}$
$\mathcal{I}$	P^-	daml:inverseOf	$\{\langle x_1, x_2\rangle \in P^{\mathcal{I}} \mid \langle x_2, x_1\rangle \in P^{\mathcal{I}}\}$
	$\leq nP$	daml:maxCardinality	$\{x_1 \mid \lvert\{x_2 \mid \langle x_1, x_2\rangle \in P^{\mathcal{I}}\}\rvert \leq n\}$
$\mathcal{N}$	$\geq nP$	daml:minCardinality	$\{x_1 \mid \lvert\{x_2 \mid \langle x_1, x_2\rangle \in P^{\mathcal{I}}\}\rvert \geq n\}$
	$= nP$	daml:cardinality	$\{x_1 \mid \lvert\{x_2 \mid \langle x_1, x_2\rangle \in P^{\mathcal{I}}\}\rvert = n\}$
	$\leq n.P.C$	daml:maxCardinalityQ	$\{x_1 \mid \lvert\{x_2 \mid \langle x_1, x_2\rangle \in P^{\mathcal{I}} \wedge x_2 \in C^{\mathcal{I}}\}\rvert \leq n\}$
$\mathcal{Q}$	$\geq n.P.C$	daml:minCardinalityQ	$\{x_1 \mid \lvert\{x_2 \mid \langle x_1, x_2\rangle \in P^{\mathcal{I}} \wedge x_2 \in C^{\mathcal{I}}\}\rvert \geq n\}$
	$= n.P.C$	daml:cardinalityQ	$\{x_1 \mid \lvert\{x_2 \mid \langle x_1, x_2\rangle \in P^{\mathcal{I}} \wedge x_2 \in C^{\mathcal{I}}\}\rvert = n\}$
	D	daml:Datatype	$D^{\mathcal{I}} \subseteq {\Delta_D}^{\mathcal{I}}$
$(\mathcal{D})$	T	daml:datatypeProperty	$T^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times {\Delta_D}^{\mathcal{I}}$
	$\exists T.d$	daml:hasClass + XMLS Type	$\{x_1 \mid \exists x_2.\langle x_1, x_2\rangle \in T^{\mathcal{I}} \wedge x_2 \in D^{\mathcal{I}}\}$
	$\forall T.d$	daml:toClass + XMLS Type	$\{x_1 \mid \forall x_2.\langle x_1, x_2\rangle \in T^{\mathcal{I}} \rightarrow x_2 \in D^{\mathcal{I}}\}$

Table 5.1: The correspondence of DL and DAML+OIL terms.

tions. In summary, the reasoning tasks for the DAML+OIL classes include the following four categories:

Satisfiability: A class C is satisfiable if there exists an interpretation $\mathcal{I}$ so that $C^{\mathcal{I}} \neq \emptyset$. Such an interpretation $\mathcal{I}$ is called a model of C.

Subsumption: A class C is subsumed by a class C' if $C^{\mathcal{I}} \subseteq C'^{\mathcal{I}}$. In this case we write $\mathcal{I} \models \mathcal{C} \sqsubseteq \mathcal{C}'$.

Equivalence: Two class C and C' are equivalent if $C^{\mathcal{I}} = C'^{\mathcal{I}}$. In this case, we wirte $\mathcal{I} \models \mathcal{C} \equiv \mathcal{C}'$. C and C' are equivalent iff C is subsumed by C' and C' is subsumed by C.

Disjointness: Two classes C and C' are disjoint if $C^{\mathcal{I}} \cap C'^{\mathcal{I}} = \emptyset$.

One can reduce the above four tasks to the unsatisfiability problem of classes. For classes C_1 and C_2, we have:

1. C_1 is satisfiable iff $\neg C_1$ is unsatisfiable;

2. C_1 is subsumed by C_2 iff $C_1 \sqcap \neg C_2$ is unsatisfiable;

3. C_1 and C_2 are equivalent iff both $C_1 \sqcap \neg C_2$ and $\neg C_1 \sqcap C_2$ are unsatisfiable;

4. C_1 and C_2 are disjoint iff $C_1 \sqcap C_2$ is unsatisfiable.

The unsatisfiability of class descriptions can be proven by so-called tableau-based algorithms. The basic idea of the algorithm is to decide the satisfiability of the classes with respect to the property hierarchies and transitive properties. This procedure, which is described in more detail in [53], incrementally builds the model by looking at the formula, by decomposing it in a top/down fashion. Since $\mathcal{SHIQ}(\mathcal{D})$ has a tree model property, we can assume that this model has the form of a finite tree. In the tree representing this model, a node x corresponds to an instance, and we label each node with the set of classes that the node is supposed to be an instance of. Similary, edges represent property-successor relationships, and an edge between x and y is labelled with the properties supposed to connect x and y. The procedure exhaustively looks at all possibilities, so that it can eventually prove that no model could be found for unsatisfiable formulas. The unsatisfiable formula is called a clash, i.e., an obvious inconsistency, such as $\{C, \neg C\}$.

An efficient algorithm for $\mathcal{SHIQ}(\mathcal{D})$ is implemented in the FaCT system [50] and has been proven to be PSPACE complete. The FaCT system is programmed by Lisp language and has a CORBA interface to communicate with other systems. FaCT has been used for many DLs reasoning tasks, in particular in OilEd [16] to inference the OIL language. The connection among DLs, $\mathcal{SHIQ}(\mathcal{D})$, FaCT and OIL is illustrated in Figure 5.2, which is revised from the original version in [78]. In summary, FaCT is an inference engine for $\mathcal{SHIQ}(\mathcal{D})$, a fragment of DLs. OIL is implemented as the logical programming language for FaCT. DAML+OIL is the XML syntax for OIL so it can be used as a Web based ontology language.

The Web service composition system in this thesis requires to know the subsumption relationships between each two classes or two properties. This task can be committed by DL reasoners. For example, in the composition system, the Web services are described by DAML+OIL language, so FaCT can be used to detect the hidden subsumption relationships between DAML classes. From the application prespective, in term of DAML+OIL constructors, the hidden subsumption relationships exist in the following three occasions.

The transitive subsumption relationships: If class A is a subclass of class B and class B is a subclass of class C, then class A is a subclass of class C. Such rule also applies to the subproperty relationships.

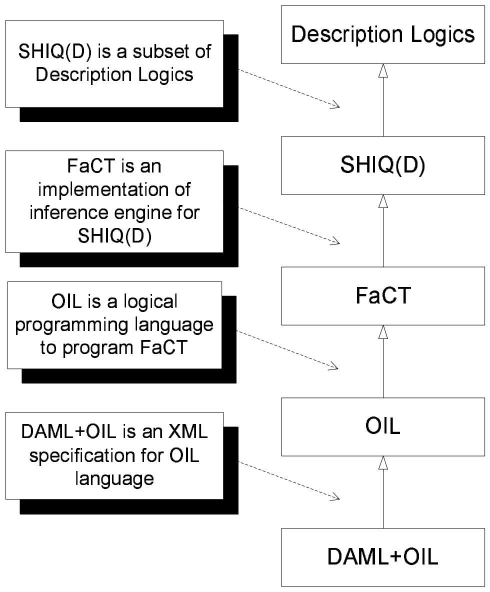

SHIQ(D) is a subset of Description Logics
FaCT is an implementation of inference engine for SHIQ(D)
OIL is a logical programming language to program FaCT
DAML+OIL is an XML specification for OIL language
Description Logics
SHIQ(D)
FaCT
OIL
DAML+OIL

Figure 5.2: The connections among DLs' components, adapted from [78]

Part of the union: If class A is defined as a union of multiple classes of which class B is a member, then class B or any subclass of class B are subclass of class A.

Narrower inclusive datatype: For two classes A and B that are both defined as the restrictions of datatype properties, if the restriction to A is narrower than that to B, A is a subclass of B. For example, the class $Adult$ could be asserted to be equivalent to $Person \sqcap \exists age.over18$, which states the adults are person whose age is over 18 years old. Thus the class $OldPeople$, which is defined as person whose age is over 60 years, is a subclass of $Adult$.

All the above three occasions can be proven by tableau-based algorithms in FaCT system.

After the detection of the subsumption relationship for each two DAML classes or properties using the method included in this Section, the next step is to use those subsumption relationships in the LL theorem prover so that the Web service composition system would deal with the subsumption polymorphism just as what has been done in Object-Oriented programming. The next Section describe the subtyping methods in LL.

5.4 Subtyping Rules for LL

Under the condition that the subsumption relationships between the DAML classes and properties are known, the problem is how to use the subsumption relationships in the LL theorem prover so that the Web Service composition process takes the Semantic Web information into account inside the proof. We have developed a method to integrete the subsumption relationships to the inference rules introduced in the previous Chapter. This method uses the LL propositions to represent the specific DAML classes and properties, and the subsumption relationships are represented in the form of LL implication $\multimap$. In this Section, we will discuss the subtyping rules in term of LL formulae.

We just recall some terms in LL. In the intuitionistic LL, a sequent is divided into two parts by $\vdash$ symbol. We call the propositions at the left side as "resources", and the propositions at the right side are called "goals". A LL sequent can be explained as: a set of goals can be achieved by consuming a set of resources. In the following two cases, the resources or goals in a LL sequent can be replaced:

1. given a goal x of type T, it is safe to replace x by another goal y of type S, as long as it holds that T is a subtype of S;

2. given a resource x of type S, it is safe to replace x by another resource y of type T, as long as it holds that T is a subtype of S.

Here, type T is a subtype of S iff S subsumes T. More precisely, the following rules capture basic facts of the subtype relations. These rules are defined as subtype behavior presented by Markus Lumpe [69]. In the rules, $S <: T$ denotes that proposition S is a subtype of proposition T.

The subtype relations consist of two structural rules stating that it is relexive and transitive:

Subtyping reflexivity:

$$\vdash T <: T$$

Subtyping transitivity:

$$\frac{\vdash T <: S \quad \vdash S <: U}{\vdash T <: U}$$

The subtyping rules state that the replacement of either resource or goal propositions given the subtyping relations:

Resource subtyping:

$$\frac{U \vdash S \quad \vdash T <: U}{T \vdash S}$$

Goal subtyping:

$$\frac{T \vdash U \quad \vdash U <: S}{T \vdash S}$$

In addition, two propositions are of the same type iff they subsume each other:

$$S \equiv T \dashv\vdash (S <: T) \wedge (T <: S)$$

The readers may have noticed that the subtyping operator <: has same inference behaviour as the LL implication $\multimap$. Actually, if we replace the subtyping operator by the LL implication, the above subtyping rules can be proven by LL inference rules.

Subtyping reflexivity can be proven directly from "Identity" rule:

$$\dfrac{\dfrac{}{T \vdash T}\ id}{\vdash T \multimap T}\ R \multimap$$

Subtyping transitivity can be proven by "cut" rule in LL:

$$\dfrac{\dfrac{\dfrac{\vdash T \multimap S}{T \vdash S}\ Shift \quad \dfrac{\vdash S \multimap U}{S \vdash U}\ Shift}{T \vdash U}\ cut}{\vdash T \multimap U}\ R \multimap$$

The resource and goal subtyping rules can be proven by "cut" as well:

$$\dfrac{\dfrac{\vdash T \multimap U}{T \vdash U}\ shift \quad \dfrac{}{U \vdash S}\ ax}{T \vdash S}\ cut$$

$$\dfrac{\dfrac{}{T \vdash U}\ ax \quad \dfrac{\vdash U \multimap S}{U \vdash S}\ shift}{T \vdash S}\ cut$$

Type equivalence is defined as LL theorem:

$$S \equiv T \dashv\vdash (S \multimap T) \otimes (T \multimap S)$$

According to the above proof, the subtyping rules are defined as certain inference figures in LL. In order to emphasize that new inference rules are used for typing purposes, but not for describing the Web service computation like what we introduced in the previous Chapter, we further write $\multimap_<$ to denote the subtype relations.

Considering the process extraction, we associate the proof terms to the inference rules. Thus the "Goal subtyping" and "Resource subtyping" rules are rewritten as follows:

$$\dfrac{\Sigma \vdash P : T(t) \quad \Gamma \vdash T \multimap_< S}{\Sigma, \Gamma \vdash P : S(t)}\ Goal\ subtyping$$

$$\frac{\Sigma, S(s) \vdash G \quad \Gamma \vdash T \multimap_{<} S}{\Sigma, \Gamma, T(s) \vdash G} \textit{ Resource subtyping}$$

The subtyping rules can be applied either to functionality (parameters and states)or non-functional attributes. We use the following two examples to illustrate the basic idea of subtype relations.

First, we show a simple case of how two services with different types of parameters are composed. Let us assume that the output of the skis recommendation service is the length of skis measured by centermeter. While the input of a measurement converting service is a general centermeter unit that can be used to measure everything.

```
<daml:Class rdf:about="&Onto#SkiLengthInCM">
  <rdfs:subClassOf>
    <daml:Class rdf:about="&Onto#LengthInCM"/>
  </rdfs:subClassOf>
</daml:Class>
```

Because the DAML Class `&onto#SkiLengthInCM` is a subclass of the DAML Class `&onto#LengthInCM`, it is safe to transfer data from the more specific output to the more general input. We can either apply the “Goal subtyping” rule to replace the output by its superclass, or apply the “Resource subtyping” rule to replace the input by its subclass.

Another example considers the non-functional qualitative fact about service location. We assume that a service requester specifies that the composite service is located in Europe, which means all atomic services included in the composite service should be in Europe. If an atomic service is located in Norway and since Norway is an European country, the DAML class definition of Norway is a subclass of the DAML class definition of Europe. Because the qualitative fact is presented as a LL goal proposition, we can us the goal subtyping to replace Norway with Europe in the service composition process using LL theorem prover. Intuitively, if the user requires a service that is located within Europe, the service located within Norway meets such requirement.

5.5 Summary

In this Chapter, we have elaborated the method to introduce subsumption reasoning to the Web service composition system. Because the building blocks of Web services, including functionalities and non-functional attributes are described by DAML+OIL based Semantic Web language, we can use semantic reasoner to detect the subtyping relationships between the DAML classes and properties. We further prove that the subtyping relatiioships can be represented by LL implication, thus they can be asserted into the LL theorem prover as axioms. In case all the subtyping relations can be detected by the semantic reasoner, the Web service composition system should deal with the subtypes during the composition process.

A limitation of the method in this Chapter is that it can only handle the subsumption relationships from one concept to another concept. Actually, the "union"($\sqcup$) in DL can be mapped to LL "multiplicative conjunction"($\otimes$), and the "disjunction" ($\sqcap$) can be mapped to LL "additive disjunction"($\oplus$) by following some restrictions. For example, we assume the type of an output for a service is a union of two classes C and D, and the other service has two conjunct inputs that have types as C and D respectively. It should be possible to decompose the output into two parts, thus they can be sent to two separate inputs. Such problem has been partly discussed in [23]. But we will not go deeply because of the complexity of such method.

We should emphasize that the semantic reasoner can only detect the subsumption relationships that can be reasoned from the applied fragment of DLs. In the research area of ontology mapping, information retrieval and statistic methods are used to find the similarity between two concepts, including the subsumptions. A survey of such kind of methods is reported in [59]. However, such methods can not ensure the completeness and correctness of the result, so we do not consider them in this thesis.

Chapter 6

A Multi-agent Architecture

6.1 Introduction

In this Chapter, we will present a system architecture for Web service composition based on a multi-agent platform, AGORA. We use software agents to represent the Web services' providers, requesters and other facilitators due to the consideration of the dynamic nature of Web and Web services. In contrary to the centrally controlled systems, the distributed systems, such as multi-agent systems provide more scalability and flexibility to the Web service applications. In summary, the increasing demand of distribution for Web service environment raises the following new challenges:

- The number of Web services increases rapidly, which makes a huge repository. But little consideration has been reported on scalability for those kinds of system currently. It is common to have a centralized maintenance of Web services repository/directory/catalog which restrict scalability.
- Both the location and the programming interface of the Web services can be changed. For a central controlled system, the service composer needs to know all the changes about the existing services by checking them frequently. That's a big hurdle on performance.
- The search and retrieval of components is done only in one way. It means that a system looks for Web services but not Web services look for a system where they can be applied.

A practical way to solve those problems is applying agent system into automated programming. The agent system is composed of service requesters, service providers and other facilitators, such as theorem prover, semantic reasoner, translator, etc. The provider, acting as the representative of a Web service, advertises the service's specification, e.g. functionalities and non-functional attributes, to the the broker agents. The requester represents the one who needs the composite service. It gives the requirement specification to the broker agents too. A broker agent is a synthesizer who builds a service that meets the requester's requirement by the composition of a set of providers' services. During the composition process, the broker agent interacts with other facilitator agents for the specific functions, such as semantic reasoning and multi-criteria negotiation.

The advantages of this architecture are:

- The system is easily scalable. No central repository is required.
- Decentralized control and asynchronous processing.
- Both providers and requesters can be active in performing mutual search. The providers are autonomous which can inform any change on components to the broker agents.
- Easily to integrate into a Web service framework, for example, Apache and .NET.
- Non-traditional software engineering methods like, service advertising and marketing, can be embedded to the service composition system, since service providers are proactive.

In this Chapter we propose the agent architecture based on AGORA infrastructure [74]. In Section 6.2 we introduce the basic concepts of AGORA system. Next we present a system architecture for Web service composition based on AGORA system. The system architecture includes both the agent model and the interaction model. Finally, we present conclusions and future work.

6.2 The AGORA Multi-agent Platform

6.2.1 General Description

The AGORA system is a multi-agent infrastructure which provides support for implementation of software agents and agent-based marketplaces. The concept of cooperative node is an infrastructure element where agents participating in a cooperative activity can register themselves and get support for communication, coordination and negotiation among the agents. The basics of the AGORA system architecture are depicted in Figure 6.1.

It consists of a set of interconnected components. There are basically two component types: agents and agora nodes. Agora nodes are meeting places or facilitators that provide services and a common context for agents. Agents are divided into registered agents (external) and default agents (internal). The

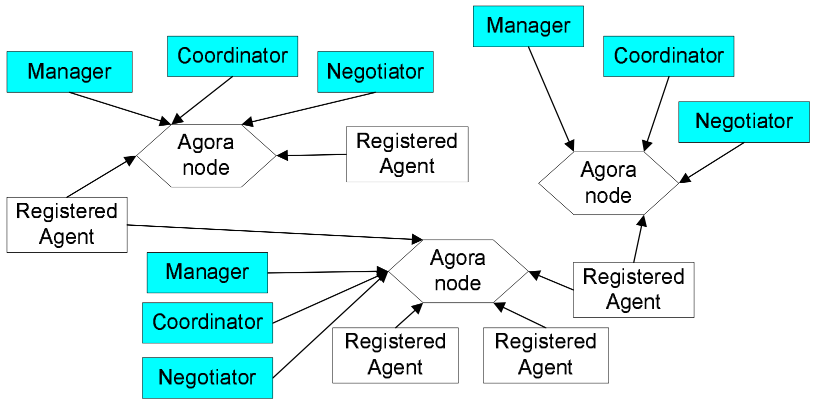


Figure 6.1: Basic AGORA system architecture.

registered agents are participants at the agora nodes who represent the service providers and service requesters. The default agents are attached to agora nodes to perform specific tasks, for example, the coordinator, the negotiator and the manager.

The idea is that the registered agents with specific goals or interests receive services and information through agora nodes. An registered agent can register itself at an agora node by announcing its name and general properties. The properties might consist of goals, tasks of interest or tasks it can perform. The agora node collects information about the registered agents and helps them to fulfill their tasks and goals. Some functionalities of the agora nodes are implemented through their default agents.

An agent can be registered at several agora nodes, and may have different goals and tasks registered in each of them. Agora nodes associate agents with roles according to their registered goals and tasks. The role of an agent may be predefined or established dynamically after registration.

6.2.2 Agent

The structure of a single agent, either a default agent or a registered agent is illustrated in Figure 6.2. An agent uses the Message Proxy and the Log System to interact with the outside world. The Message Proxy allows agents communicate in Agent Communication Language, which is compliant with both FIPA ACL [3] and KQML [41].

The Log System is the interaction channel between the agent and its owner, normally a human user. First, the user is able to monitor the executing status of an agent through the Log System. This information can be presented either through a GUI log window or stored as a log file. The second function of the

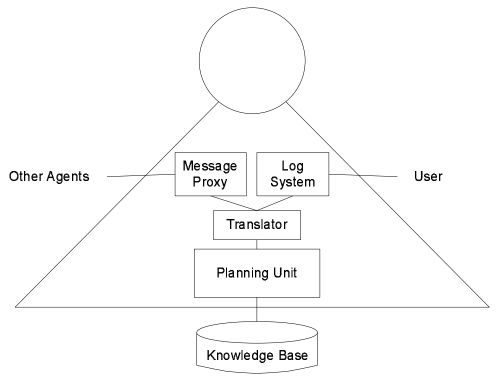


Figure 6.2: An agent architecture.

Log System is that the agent should be able to get instructions from the user at runtime. This is important for e-market applications because, in many cases, the final decision is made by the user and not by the software agent.

We use a Prolog-based presentation for messages, facts and rules in the Knowledge Base, implemented using the XProlog system [120]. In order to integrate the FIPA messages with the Knowledge Base, a Compiler between agent communication language and Prolog clauses is implemented. Currently, the compiler just translates seven FIPA message elements that are important for the decision making process: performative, ontology, sender, receiver, content, reply-with and in-reply-to. The last two are used to uniquely identify a message. This set of elements can be extended as required by the specific application.

The Planning Unit decides the agent's next action by a set of explicitly defined rules. In Agora, the plan is specified in a XML-based scripting language. Each step in the plan has an action to be performed and post-conditions. The action refers to an outgoing FIPA message or a method (function) written in Java or Prolog. Post-conditions are described as a reaction of the agent to a communicative act received from another agent.

6.2.3 Agora node

The Agora node is a cooperative node which facilitates communication, coordination and negotiation between agents. As shown in Figure 6.1, the registered agents connect to the agora nodes and exchange information. Basic information about the registered agents is stored in individual agent information repository. It contains:

Identity: name and address of the agent.

Goals: the problem to be solved or supported by the agent.

Beliefs, desires and intentions (BDI): agents are rational and have the ability to reason about the information they are working on.

Tasks: those an agent can perform and those an agent is interested in.

Friend agents: the name of agent who have connection with this agent.

This information makes it possible for the agora node to provide the services expected by the registered agents. The basic goal is to matchmaker agents to each other, so that they can cooperate and accomplish their goals in a cooperating manner.

Agora nodes and agents have some similar properties, and agora nodes may have information about itself (self-information) like tasks, goals, beliefs and so on. This may include:

- Organizational model of participating agents.
- Protocols for communication (set of performatives).
- A set of default agents (at least one coordination or negotiation agent).
- Information about current activities, both their own and those belonging to other registered agents.
- List of registered individual agents.

An agora node contains several default agents, including manager, negotiator and coordinator. A standard manager implements general agora node functions, such as matchmaking and registration. However, functionality of the manager can be modified by user via overriding the standard implementation. In particular, the manager can perform decision making procedure, fuzzy matchmaking, knowledge base and ontology managing etc. Complexity of the manager agent can be different for different applications, however, basic matchmaking functionality is provided by default in all managers. The default matchmaker can be overridden in order to support more advance functionality, such as:

- Matchmaking using ontology and semantic relations.
- Event handling.
- Decision making.
- Processing queries about registered activities, ontology used and other general information

- Handling agent registration/unregistration protocols
- Pro-active reasoning with available knowledge
- History maintenance and analysis

Negotiator manages the negotiation process. Negotiation is initiated in order to achieve an agreement between two or more parties. The negotiator agent acts as a broker in negotiation, and receives startup messages from the manager when activities require negotiation. The negotiator decides on a specific negotiation protocol, and the agents must be able to follow its instructions. One widely used negotiation protocol is the Contract Net Protocol [112], which has been implemented in AGORA system as a default negotiation protocol. The current AGORA version offers basic infrastructure for protocol specifications. In specific scenarios, decisions must be taken on what kind of attributes that should be negotiated.

Making the involved agents in a system interact in a coherent way is the responsibility of the coordinator. The coordinator is an essential part when applications are realized in open environments. However, this part is not emphasized in the current version of AGORA.

6.3 The Agent Architecture

In this Section, we propose a general architecture of the system for agent-based Web service composition. The main approach is as follows. We use software agents for representing the service providers, service requesters and facilitators. These agents cooperate in processing requests from the service requesters. The problem solving process includes the following five stages. First, the service manager agent collects the service requests and service advertisements from the service requester agents and service provider agents respectively. Second, a matchmaker agent discovers the matching offers for each request. Third, besides the exact matching results, the requests can be also fulfilled by the composition of existing services. We use LL theorem proving to fulfill each request by the composition of existing services. During the proving, the service composer agents interact with the semantic reasoner agents to get the subsumption relationship information. Fourth, if more than one solution is found, multi-criteria negotiation is used to select the "best" solution. Fifth, the unique selected composition service is used for invocation.

We present the proposed architecture in three steps. In the first step, we present an agent model that describes the agent roles who participates in the composition process and their functionalities. The second step is to elaborate an interaction model stating the interaction protocol among the participant agents. In the final step, we discuss the usage of the facilitator agents and presents a candidate solution to integrate these agents into the Web service composition system.

6.3.1 The Agent Model for Service Composition

To create an AGORA-based agent model we should go through the following steps:

- participants of the cooperative activities should be defined (in our problem they are users and service providers)
- nodes where the cooperative activities are performed should be defined (in our problem such nodes are, for example, service providers shops, user agents coordination node and facilitators coordination node)
- participants are mapped into agents
- cooperative nodes are mapped into agora nodes
- functionalities of corresponding agora managers, coordinators and negotiators are defined/implemented

Based on the above mentioned approach an AGORA-based architecture for Web service composition is presented in Figure 6.3.

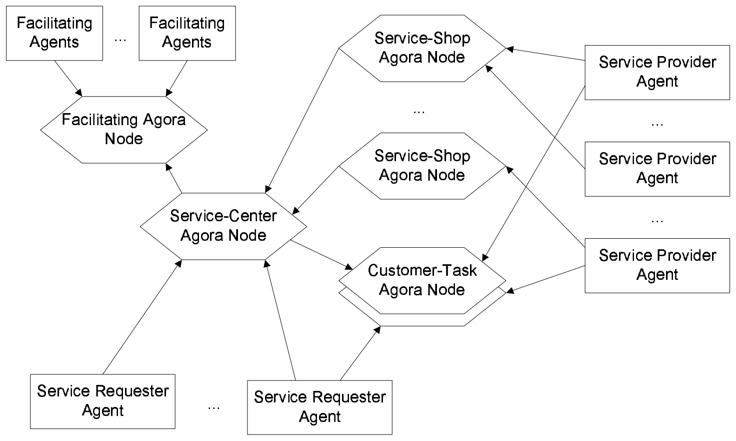


Figure 6.3: An architecture for Web services provision.

A Web service composition system includes the following participants:

Service Provider Agent: The Service Provider Agents propose Web services to users and other agents. Capability of the agents are specified by the Web services the agents provide, described by the DAML-S ServiceProfile.

Service Requester Agent: The Service Requester Agents consume information or services offered by Service Providers Agents. The service requesters are willing to pay (or provide some information) for getting the wanted information or services. We present the request of a service requester as an offer. The offer is also written in DAML-S ServiceProfile.

Service-Shop Agora Node: The Service Provider Agents register their capabilities at specific Service-Shop Agora Nodes according to their service domains. For example, agents providing services for measurement converting may register themselves and their services at Service-Shop Agora nodes devoted to that domain. Some agora nodes may represent also coalitions of service providers.

Service-Center Agora Node: The Service-Center Agora Node represents cooperative node where service provision planning is performed. Service Requester Agents register themselves at this agora node with required service tasks and Service-Shop Agora Nodes register here with their provided services. The manager of the Service-Center Agora Node may apply registration protocol to each requester agent in order to create a requester's profile (the information that the requester is willing to disclose to a service provider) and requester's task description. Another of the manager's functions is to create a composition of available services for satisfying each requester's task. The resulting service provision plan will contain the matching or composition of services registered at the Service-Center Agora Node.

Customer-Task Agora Node: When the plan is ready the coordinator of the Service-Center Agora Node creates a new Customer-Task Agora Node where coordinator of the Customer-Task Agora Node employs the service provision plan as a coordinating activity. The coordinator agent sends information to all registered Service-Shops Agora Nodes about requested services. The service providers registered at the Service-Shop Agora Nodes may contact the Service Requester Agent for marketing their services and the Service Requester Agent may decide to register at corresponding Service-Shop Agora Node for the selection of a particular service provider. Finally, only the selected Service Provider Agent register themselves to the Customer-Task Agora Node. These registered agents may provide the results of the invocation of their services to fulfill the requirement from the Service Requester Agent.

Facilitating Agora Node: During the planning process, the manager of the Service-Center Agora Node may require extensional capabilities to deal with the problems, such as theorem proving, semantic reasoning and language translating, etc. One Facilitating Agora Node is a collection of the agents who provide a specific facilitating service. For example, the agents in semantic reasoning agora node may support the reasoning on different fragment of DLs. The ability to use Facilitating Agora Nodes provides flexibility to the architecture where a new capability that is required can be registered as a facilitating agent at a specific Facilitating Agora Node.

After the participant agents and agora nodes have been defined, the next step is to design the interaction among the participants.

6.3.2 The Interaction Model

Ongoing conversations between agents often fall into typical patterns. In such cases, certain message sequences are expected, and, at any point in the conversation, some predefined messages are expected to be received. These typical patterns of message exchange are called interaction models. The term interaction model is a synonym for the term interaction protocol used in FIPA specification [6]. A designer of agent systems has the responsibility to make the agents sufficiently being aware of the meanings of the messages and the goals, beliefs and other mental attitudes the agent possesses, and that the agent's planning process causes such interaction models to arise spontaneously from the agents' choices. In other words, at any point, the agent should know what is its next step according to the message it receives from other agents. In the AGORA system, we use a reactive model to direct the agent following a given interaction model, so we could develop as many interaction models for the agents in agora nodes by storing them in different plan files. In general, the interaction model is a pragmatic solution for agent conversation, so that an agent can engage in meaningful conversation with other agents, simply by following the interaction model.

The interaction model is formally represented via preconditions and actions. Each precondition is defined by the performatives in the interaction model. We use the FIPA (The Foundation for Intelligent Physical Agents) [3] communicative acts for interoperability with other FIPA compliant agent systems. Each performative has unique meaning in the context of service composition ontology (Table 6.1).

Figure 6.4 presents the proposed interaction model in Agent UML notation as follows.

1. The Service Provider Agents register their capabilities to a Service Shop Agora. The Service Shop Agora then tells the registration information to the Service Center Agora.

2. The Service Requester Agents register their offers to the Service Center Agora.

3. The manager of Service Center Agora starts the matching and composition process. For each offer, three alternative results can be returned according to the available services and offers:

   - If the offer can be satisfied by a single registered capability, both the Service Requester Agent who submits the offer and the Service Provider Agent whose capability is matched are informed.
   - If the offer can be satisfied based on the registered capabilities, both the Service Requester Agent and the Service Provider Agents whose capabilities are candidate solutions are informed.
   - If no solution is found, the process is failed.

4. After the Service Requester Agent and Service Provider Agents are get informed, they will register at the Service Customer Agora and submit their preferences.

5. The Service Customer Agora holds a negotiation and the "best" offer will be selected. The negotiation process is a coalition formation task which has been discussed in many papers. In particular, a virtual enterprise formation method based on AGORA has been introduced in [100]. This method select the partners based on their costs, availabilities and skills providing the candidate partners are known. This method can be also used in composite service selection.

6.3.3 Facilitating Agora Nodes

We use the facilitating agoras to improve the scalability and flexibility of the service composition system. A system architecture is illustrated in Figure 6.5. In general, a logic-based Semantic Web service synthesis system requires the following components:

Translator Agents: translate between an external presentation of Web service and the internal logical presentation. In our system, DAML-S profile is used externally for presentation of semantic web service specification, while LL axioms are used internally for planning and composition. The process model is internally presented by a process calculus. The calculus can be translated either into DAML-S or BPEL4WS.

Matchmaker Agents: allow service requesters to upload their service requests, and the matchmaker agent determines the connectivity of the requirements with the registered services in its repository. There are quite a lot

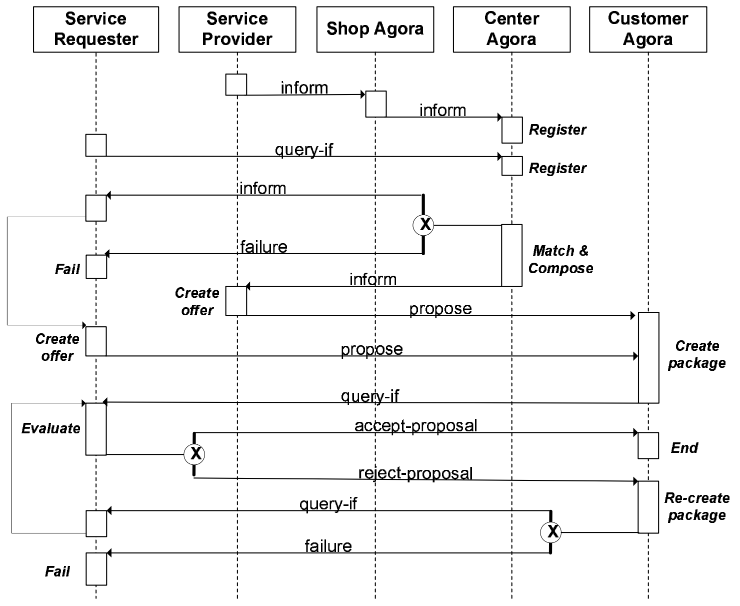


Figure 6.4: An interaction model of service composition participants.

work for Web service matching. A significant result based on DAML-S service specification has been reported in [97]. The matching algorithm used in this work has been further elaborated in [115]. It is natural to assume that the matchmaker agent uses this algorithm in the matchmaking process. Although the service composition is also a special case for service matching, we believe that the matchmaking algorithm is usually more efficient than the composition algorithm, since the matchmaking task is usually simpler. Therefore, the overall performance of the system is improved by separating the matchmaking and the composition tasks.

Theorem Prover Agents: prove whether the input sequent can be proven by the available axioms. In case the sequent represents the user's requirement and axioms represent the existing services, the proving process actually constructs the process model for the composite service. The process model is extracted from the proof.

Semantic Reasoner Agents: detect the subtyping and some other relationships among concepts in input ontology. The formal logics used in Semantic Reasoner Agent could be logics developed for expressing knowledge and reasoning about concepts and concept hierarchies, for example, the Description Logic [45].

Other Adapter Agents: perform as the mediator between other component

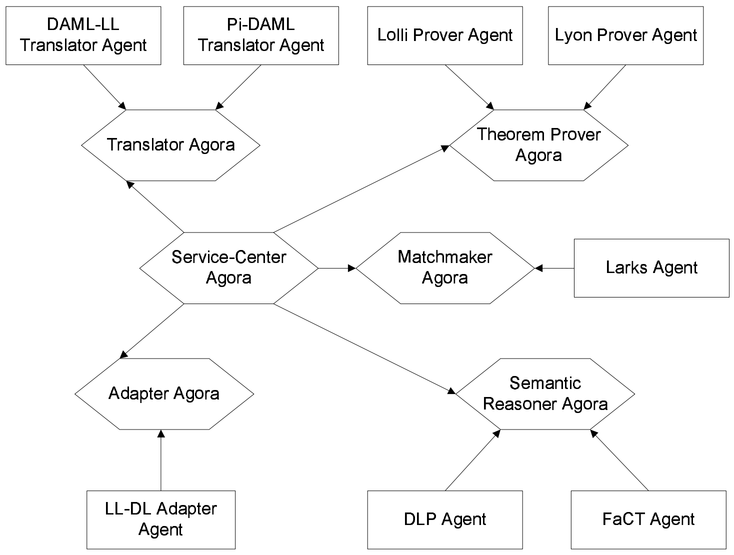


Figure 6.5: The agent architecture for facilitators.

agents who can not interact with each other. For example, although we have proven the subsumption relationship has the same behavior as the implication in LL, they are encoded differently in $\mathcal{SHIQ}(\mathcal{D})$ DL Semantic Reasoner Agent and LL Theorem Prover Agent. Thus an Adapter Agent is required for the translation between LL and the internal presentation used by the Semantic Reasoner.

1. For each service advertisement, the manager of Service Center Agora sends it to the DAML-LL Translator Agent. The return LL axiom is asserted into LL Theorem Prover Agent.

2. On receiving a requirement, the manager of Service Center Agora asks the DAML-LL Translator Agent to transform the requirement into LL sequent. The sequent is sent to the LL Theorem Prover Agent for proving.

3. Before proving, the LL Theorem Prover Agent ask the Adapter Agent to check the hidden subsumption relationships.

4. The Adapter translates the classes and properties to an ontology model in the format that is recognized by the Semantic Reasoner Agent.

5. The Semantic Reasoner Agent analyzes the ontology model and derives all subtype relations of the classes and properties in the ontology model.

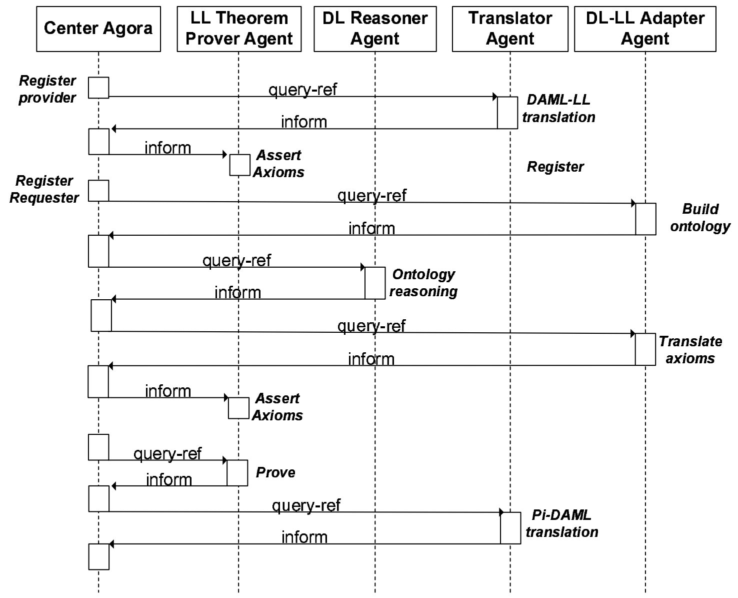


Figure 6.6: An interaction model for facilitators.

6. The Adapter Agent translates the subtype relations to LL axioms. The axioms are sent to the LL Theorem Prover Agent.

7. The LL Theorem Prover Agent checks whether the request can be satisfied by composition of existing atomic services (this is done by performing theorem proving in LL). If the sequent corresponding to the requested composite service has been proven and the proof is generated then a process calculus presentation is extracted from the proof directly.

8. The last step is the construction of flow models. For example, if the process calculus is sent to the calculus-DAML Translator Agent, the process model described in DAML-S ServiceModel is returned to the Service Center Agora.

6.4 Summary

In this Chapter we describe an agent-based design for the system of automated composition of Web services. Agent-specific aspects provide Web service composition system with proactivity, reactivity, social ability and autonomy, while the use of DAML-S, FIPA ACL and application domain specific ontologies provides a standardized medium for Web service deployment. In particular, the use of DAML-S allows to publish semantically enriched specifications of Web services and thus to fit well to the Semantic Web initiative, where both Web services and data are labeled with semantic information.

Another aspect of the agent-based design is that it enables the plugin of the facilitator agents. This design enables the different components for Web service composition system, such as the theorem prover, semantic reasoner and translator to integrate to each other in a loosely coupled manner. We present a general architecture and interaction model for such design.

Although the different components are located distributely, the service composition is centrally controlled by the Service Center Agora Node that could be the bottleneck for the performance. The idea of using Partial Deduction [67,61] to enable cooperative problem solving has been discussed in [62]. The system performance would be improved if multiple service composer agents can work in a coordinated way. However, this method need further development and justification, so it will not be included in this thesis.

Performative	*Sender*	*Receiver*	*Meaning*
inform	Provider	Shop Agora	The Service Provider Agent tells its capability to the Service Shop Agora
inform	Shop Agora	Center Agora	The Service Shop Agora tells the capabilities of all registered agents to the Service Center Agora
query-if	Requester	Center Agora	The Service Requester Agent tells its offer to the Service Center Agora
inform	Center Agora	Requester	The Service Center Agora finds one or more solution for the Service Requester Agent's offer. The Service Requester Agent should contact with the Customer Task Agora in selecting the best solution
failure	Center Agora	Requester	The Service Center Agora does not find any solution for the Service Requester Agent
inform	Center Agora	Provider	The Service Center Agora informs one Service Provider Agents to be selected as a candidate to provide its service. The Service Provider Agent should contact with the Customer Task Agora for negotiation
propose	Requester	Customer Agora	The Service Requester Agent registers at the Service Customer Agora and tells its preferences on composite service
propose	Provider	Customer Agora	The candidate Service Provider Agent registers at the Service Customer Agora and tells the conditions on using its service
query-if	Customer Agora	Requester	The Service Customer Agora creates a package solution according to the Service Requester Agent's preference and asks if the requester will accept it
accept-proposal	Requester	Customer Agora	The Service Requester Agent satisfies with the proposed package solution
reject-proposal	Requester	Customer Agora	The Service Requester Agent does not satisfies with the proposed package solution
fail	Customer Agora	Requester	No solution can satisfy the Service Requester Agent, so the negotiation fails

Table 6.1: Meanings of performatives.

Chapter 7

The Prototype Implementation

7.1 Introduction

A prototype of the approach is implemented as a part of the thesis work. The prototype is the main tool to verify that the proposed approach is an applicable solution. It also paves way for evaluating the approach in both qualitative and quantitative manners. This chapter is focused on functionality specification, as well as some technical details that are specialized for this system.

We illustrate an implemented architecture in Figure 7.1. Although, as we have described in Chapter 6, the different agents connect to different agoras, we omit the details about how agoras are connected. Here, we emphasize four components for implementing the functionalities of the agents who form the composition system. They are Jena for parsing and generating DAML documents, FaCT for semantic Web reasoning, Forum for LL theorem proving and GUI for interacting with end-users. We describe them in detail as follows:

Jena: Jena (a Java framework for building Semantic Web applications) [5] is used to implement the translator between DAML-S documents and the internal presentation of Web services.

FaCT: The FaCT system [50] is programmed by Lisp language and has a CORBA interface to communicate with other systems. FaCT has been used for many DLs reasoning tasks, in particular in OilEd [16] to inference the OIL language.

Forum: Forum, a LL programming language introduced by Dale Miller in 1994 [87], is distinguished by two key features. First, it is complete for all fragments of LL, in the sense that LL operators that are not part of Forum can be mapped to Forum by a provability-preserving translation. Second, a form of goal-directed proof search (as characterized by uniform proofs) is complete. Forum, as an abstract language framework, can be implemented by different languages. In our work, we use UMA Forum

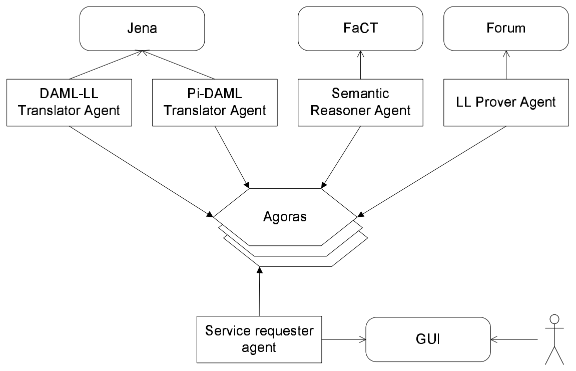


Figure 7.1: An illustration of the implemented agents.

that is developed by Pablo López and Ernesto Pimental in University of Málaga, Spain. UMA Forum is an implementation in Prolog of a subset of Forum. It is being used as part of a research project concerning LL logic programming, object orientedness and concurrency. UMA Forum can be obtained from http://www.lcc.uma.es/~lopez/umaforum/.

GUI: visualizes services (both composed and atomic). The graphical presentation includes visualization of functionalities and non-functional attributes. The GUI has been implemented in Java Swing.

We will not present the components in same detail, because the implementations of most components are quite straightforward. The most difficult part is the LL theorem prover based on Forum. The point is, we should encode the proof extraction method described in Chapter 4. We will firstly introduce the implementation of LL theorem prover in the next section. Then we will discuss other components in sequence.

7.2 LL Theorem Prover

In recent years, a number of logic programming languages based on LL have been proposed: LO [11,12], LinLog [10], Lolli [49], Lygon [48], Forum [87] and RAPS [60].

A foundation of LL theorem prover is the idea of *uniform proof* [88], proposed by Miller et. al. Uniform proof is a simple and powerful notion for designing logic programming languages. Uniform proof search is a cut-free, goal-direct proof search in which a sequent $\Gamma \vdash G$ denotes the state of the computation trying to solve the goal G from the resource Γ. Uniform proof is characterized operationally by the bottom-up construction of proofs in which right-introduction rules are applied first and left-introduction rules are applied only when the right-hand side is atomic. This means that the operators in the goal G are executed independently from the program Γ, and the program is only considered when its goal is atomic. A logical system is an abstract logic programming language if restricting it to uniform proofs retains completeness. Lolli is a LL programming language based on the theory of uniform proof.

Although uniform proof can guarantee the soundness and completeness, its highly restriction makes a lot of valid sequent unprovable. For example, it can not deal with the program that has multiple conjunctive goals. In this way, uniform proof has problem to present the concurrent features emerging from LL.

Forum, providing the presentation of concurrency features, can be regarded as an extension of those systems based on uniform proof. Forum, a fragment of LL introduced by Dale Miller in 1994 [87], is distinguished by two key features. First, it is complete for all of LL, in the sense that LL operators that are not part of Forum can be mapped to Forum by a provability-preserving translation. Second, a form of goal-directed proof search (as characterized by uniform proofs) is complete.

From this point, Forum is most likely a theorem prover instead of a logic programming language.

Since we use LL sequents to represent Web services, we should study the method for specifying LL sequents and their inference rules using Forum presentation. As we have mentioned before, a requirement to a Web service is expressed by a LL sequent in the form of $\Gamma; \Delta \vdash G$, where Γ denotes a set of extralogical axioms representing available Web services, Δ is a multiplicative conjunction of non-functional constraints; and G is a multiplicative conjunction of a required service and non-functional results. This sequent can be translated into a Forum sequent as:

$$\cdot; \cdot \vdash ?\lfloor \Gamma \rfloor \wp \lfloor \Delta \rfloor \wp \lceil G \rceil.$$

The $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ predicates are used to identify which formulas appear on which side of the sequent, and the ? modal is a par of the ! modal, which is used to mark the formulas can be used for unbounded times.

The inference rules in LL sequent calculus can be encoded as clauses in Forum. Consider the LL inference rules we have introduced in Table 4.2, we can specify the inference rules in Forum using the clauses in Table 7.1. Let Ψ represent the set of clauses, The sequent

$$\Psi; \cdot \vdash ?\lfloor \Gamma \rfloor \wp \lfloor \Delta \rfloor \wp \lceil G \rceil$$

has a Forum proof iff the sequent $\Gamma; \Delta \vdash G$ has a LL proof using the inference rules in Table 4.2. This issue has been proven in [89]. Here, the right introduction rule for multiplicative conjunction ($R\otimes$) enables us to prove the

(id)	$\lfloor B \rfloor \wp \lceil B \rceil$	(Cut)	$\bot \multimapinv \lfloor B \rfloor \multimapinv \lceil B \rceil$
$(1L)$	$\lfloor 1 \rfloor \multimapinv \bot$	$(1R)$	$\lceil 1 \rceil \Leftarrow \top$
$(L\otimes)$	$\lfloor A \otimes B \rfloor \multimapinv \lceil A \rceil \wp \lfloor B \rfloor$	$(R\otimes)$	$\lceil A \otimes B \rceil \multimapinv \lfloor A \rfloor \multimapinv \lceil B \rceil$
$(L \multimap)$	$\lfloor A \multimap B \rfloor \multimapinv \lceil A \rceil \multimapinv \lfloor B \rfloor$	$(R \multimap)$	$\lceil A \multimap B \rceil \multimapinv \lfloor A \rfloor \wp \lceil B \rceil$
$(L\oplus)$	$\lfloor A \oplus B \rfloor \multimapinv \lfloor A \rfloor \& \lfloor B \rfloor$	$(R \oplus (a))$	$\lceil A \oplus B \rceil \multimapinv \lceil A \rceil$
$(R \oplus (b))$	$\lceil A \oplus B \rceil \multimapinv \lceil B \rceil$	$(R\&)$	$\lceil A \& B \rceil \multimapinv \lceil A \rceil \& \lceil B \rceil$
$(L\&(a))$	$\lfloor A \& B \rfloor \multimapinv \lfloor A \rfloor$	$(L\&(b))$	$\lfloor A \& B \rfloor \multimapinv \lfloor B \rfloor$
$(W!, L!)$	$\lfloor !B \rfloor \multimapinv ((B \multimap \bot) \Rightarrow \bot)$	$(C!)$	$\lceil !B \rceil \multimapinv \lceil B \rceil$

Table 7.1: Forum specification of the LL sequent calculus.

sequents other than the uniform conclusion ones. In all cases, proofs in Forum match closely proofs in the LL sequents with all operators.

We implement the clauses in Table 7.1 by the program of a specific Forum system, UMA Forum. UMA Forum is an implementation of a subset of Dale Miller's Forum Specification Language developed at the University of Malaga (UMA). UMA Forum uses a new stack-based resource management system based on the Lolli resource management system. The key idea of the resource management system is that LL sequent contexts (i.e. logic programs and goals) can be manipulated as stacks, just as in hereditary Harrop formula based languages like lambda Prolog. In particular, the UMA Forum resource management system represents sequent contexts as stacks instead of multisets, replacing costly union and intersection multiset operations by simple, inexpensive push and pop stack operations. This stack-based approach to resource management simplifies the implementation, providing an efficient linear logic proof search strategy at a very low cost. The UMA Forum system, written in Prolog, can be easily applied to the implementation of LL theorem provers and programming languages, either single or multiple-conclusion. The searching mechanism of UMA Forum is based on a lazy splitting system [68]. Lazy splitting can considerably improve the performance of the implementation of the theorem prover.

UMA Forum clauses are freely generated from the LL asynchronous connectives. The ASCII rendering of these connectives is shown in Table 7.2.

We implement the LL theorem prover as well as the process extraction using UMA Forum. The code is presented as follows. An atom in the program is in form of $predicate(A, B, C)$, where $predicate$ can be replaced by either res or $goal$. A is the sequent to be proven. It equals to the combination of propositions and connectives in the LL presentation of Web service. B denotes the

Name	*LL operators*	*UMA Forum connectives*
additive true	$\top$	`top`
multiplicative false	$\bot$	`bot`
multiplicative disjunction	$\wp$	#
additive conjunction	&	&
linear implication	$\multimap$	-*, *-
intuitionistic implication	$\vdash$	=>, <=
why not exponential	?	?

Table 7.2: The connectives in UMA Forum.

correspond port variables of the LL sequent represented by A. It represents the information in the brackets in the LL presentation of Web service. C provides the information of the progress of the proof. It equals to the information stored in the proof terms.

```
% linear implication
res((A -* B), impl(Name, A1, B1), 0) *- goal(A, A1, A2) *- res(B, B1, seq(A2, Name)).
goal((A -* B), 0, B2) *- res(A, 0, 0) # goal(B, 0, B2).

% multiplicative conjunction
res((A x B), (A1 x B1), par(Pa, Pb)) *- res(A, A1, Pa) # res(B, B1, Pb).
goal((A x B), (A1 x B1), par(Pa, Pb)) *- goal(A, A1, Pa) *- goal(B, B1, Pb).

% multiplicative disjunction
res((A & B), (A1 & 0), Pa) *- res(A, A1, Pa).
res((A & B), (0 & B1), Pb) *- res(B, B1, Pb).
goal((A & B), (A1 & B1), par(Pa, Pb)) *- goal(A, A1, Pa) # goal(B, B1, Pb).

% additive disjunction
res((A @ B), (A1 @ B1), cho(Pa, Pb) *- res(A, A1, Pa) & res(B, B1, Pb).
goal((A @ B), A1, Pa) *- goal(A, A1, Pa).
goal((A @ B), B1, Pb) *- goal(B, B1, Pb).

% of course modality
res((!B), B1, Pb) *- ((res(B, !B1, Pb) -* bot) => bot).
goal((!B), B1, Pb) <= goal(B, !B1, Pb).

%cut
bot *- res(B, A1, A3) *- goal(B, A2, seq(A3, cut(A1, A2))) *- A1 \== 0 *- A2 \== 0.

%Id
res(B, 0, 0) # goal(B, A2, rec(A2)).
res(B, A1, A2) # goal(B, 0, seq(A2, snd(A1))).
```

The result of the generated process model is shown in a formula as the value of variable C. The formula has a straightforward mapping with the process calculus introduced in Chapter 4. The following presents the mapping between the formula and the process calculus:

$$
\begin{aligned}
0 &\equiv \pi(0) & !a &\equiv \pi(!a) \\
rec(a) &\equiv \pi(a) & snd(a) &\equiv \pi(\overline{a}) \\
cut(a,b) &\equiv \pi(\overline{a}b) & seq(a,b) &\equiv \pi(a.b) \\
par(a,b) &\equiv \pi(a|b) & cho(a,b) &\equiv \pi(a+b)
\end{aligned}
$$

7.3 Jena

Jena is a Java framework for building Semantic Web applications. As an open source software, it is grown out of work within the Hewlett-Packard Labs Semantic Web Programme. Jena provides a programmatic environment for RDF, RDFS DAML+OIL and OWL. By using Jena, one can parse, create and search the concepts in Semantic models based on RDF technique. The most recent version, Jena2 was released in August 2003. The Web page for Jena2 is located at http://jena.sourceforge.net.

Jena2 uses RDF Graph as its core data structure. An RDF Graph is simply a set of triples (S, P, O), where P names a binary predicate over (S, O). Here, P can either refer to a user-defined property or a reserved binary relationship, such as $subClassOf$ or $propertyOf$. S and O refers to the classes, properties, instances or XSD datatypes.

One of the main contribution of Jena is the rich Model API for manipulating RDF Graphs. Through the API, Jena provides various tools, including I/O modules for: RDF/XML, N3 and N-triple; and the query language RDQL. Using the API the user can choose to store RDF Graphs in memory or in persistent stores. In addition, Jena provides an integrated module to manipulate both DAML+OIL and OWL in an uniform data model.

The Jena architecture is illustrated in Figure 7.2. The heart of the Jena architecture is the RDF graphs, a set of triples of nodes. This is shown in the Graphs layer. The Graph layer is based on the RDF Abstract Syntax. The most important functionality of the Graph layer is a triple storage. The triples of nodes are stored both in memory and backed by persistent storage. The Model layer is the interface between the application programmer and Jena internal storage. This gives a rich set of methods for operating on both the graph itself (the Model interface) and the nodes within the graph (the Resource interface and its subclasses). Further, the DAML API is updated and enhanced in Jena2 to form Ontology Models that can be realized as a DAML API or an OWL API. The Model layers lie on top of the Graph layer via an intermediate layer: the EnhGraph layer. This provides an extension point for providing views of graphs, and views of nodes within a graph. The views include the inheritance hierarchy and polymorphism.

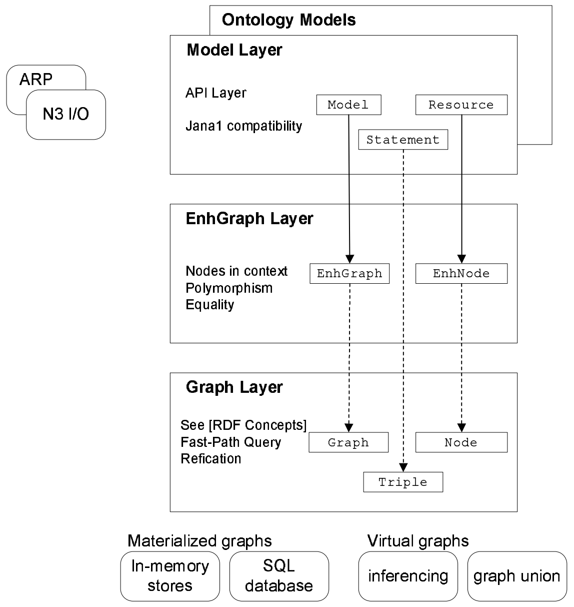


Figure 7.2: The Jena architecture, from [28]

When translating the DAML-S Web service specification, the DAML-S documents for the services are stored as Jena data models. Each instance in the model corresponds to the description of a service; The ServiceProfile of a DAML-S service is defined as a DAML class as the value of the “profile” property. The functionalities and non-functional attributes are properties of the “profile” class.

Jena2 provides a range of inference capabilities for reasoning of Semantic Web languages, such as OWL and DAML+OIL. The main part of Jena build-in reasoner is a transitive reasoner that provides transitive closure of the RDFS `subClassOf` and `subPropertyOf` relationships contained in the source graphs. This reasoning mechanism is relatively simple. In Chapter 5, we enumerate three different cases for subsumption relationships: “transitive subsumption relationships”, “part of the union” and ”Narrower inclusive datatype”. The build-in reason can only deal with the first case. Fortunately, the Jena2 architecture permits plug-in connections to engines being developed by the wider community, such as Racer [47], FaCT [50] and the Java Theorem Prover [42]. So, we use FaCT as a stronger semantic reasoning engine. The connection between Jena inference framework and FaCT is a CORBA interface.

7.4 FaCT

FaCT is a prototype knowledge representation system for DL. It uses an optimized tableaux subsumption algorithm to provide complete inference for a relatively expressive concept description language, $\mathcal{SHIQ}$. We have given sufficient introduction about the $\mathcal{SHIQ}$ fragment of the DLs in Chapter 5 and we have known the expressive power of this fragment is comparable with that of the DAML+OIL. FaCT is written in Common Lisp, and has been run successfully with several commercial and free lisps, including Allegro, Liquid (formerly Lucid), Lispworks and GNU Lisp. As the source code is available (under the GNU general public license), FaCT can be run on any system where a suitable Lisp system is available. Binaries (executable code) of FaCT are also available (in addition to the source code) for Linux and Windows systems, allowing FaCT to be used without a locally available Lisp system. A generic FaCT server has been built using the Object Management Group(OMG)'s Common Object Request Broker Architecture(CORBA) [94]. The CORBA interface was chosen because it is not tied to any particular language or platform. In particular, CORBA can be used with both Lisp and Java running on both Unix and Microsoft platform.

The communication between the FaCT client and server is to pass classes and properties as single data items using eXtended Markup Language(XML). The interface conforms to a standard "tell and ask" format: facts are asserted to the knowledge base (KB) and queries answered by the reasoner engine automatically. The available "tell" and "ask" operations that can be invoked from Java client has been listed in [17]. In general, the "tell" operations assert axioms that represent the relationships between concepts, for example, the relationship between two classes or two properties. It also asserts the conclusion on whether a property is transitive or functional. The "ask" operations will answer the question asked by the client. One of the most common used "ask" operation in the prototype is the `subsumes` operation that returns true if two classes or properties have subsumption relationship. Another operation is the `equivalent` operation that returns true if two classes or properties are equivalent.

The DAML+OIL data model of Jena can be used by FaCT directly through the third-party adapter. Given the DAML+OIL model, we use FaCT to detect the hidden subsumption relationships between each DAML classes or properties. The subsumption relationships are used in the LL theorem prover as axioms. The working process is shown as follows:

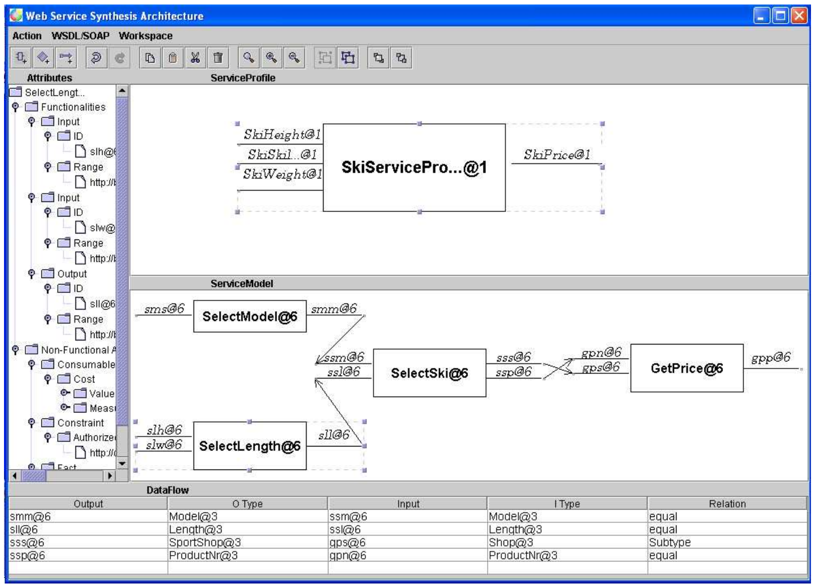
- Before the LL theorem prover starts, the Jena data model of the DAML-S documents is asserted into the KB as a set of axioms through the "tell" operation.

- We have discussed before that all propositional variables in the LL sequents refer to the DAML+OIL classes and properties. The next step is to detect the subsumption relationships between each resource propositional variable and goal propositional variable in the LL sequents. This is done by the "ask" operation provided by FaCT.

- If the answer is "true", a new axiom that indicates the subsumption relationship between the goal propositional variable and the resource propositional variable is asserted into the LL theorem prover.

- The LL theorem prover starts after all pairs are asked.

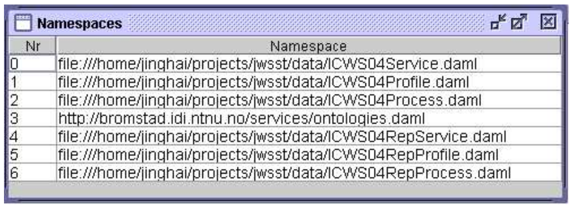
7.5 GUI



Output	O Type	Input	I Type	Relation
smm@6	Model@3	ssm@6	Model@3	equal
sll@6	Length@3	ssl@6	Length@3	equal
sss@6	SportShop@3	gps@6	Shop@3	Subtype
ssp@6	ProductNr@3	gpn@6	ProductNr@3	equal

Figure 7.3: A screen shot of main window.

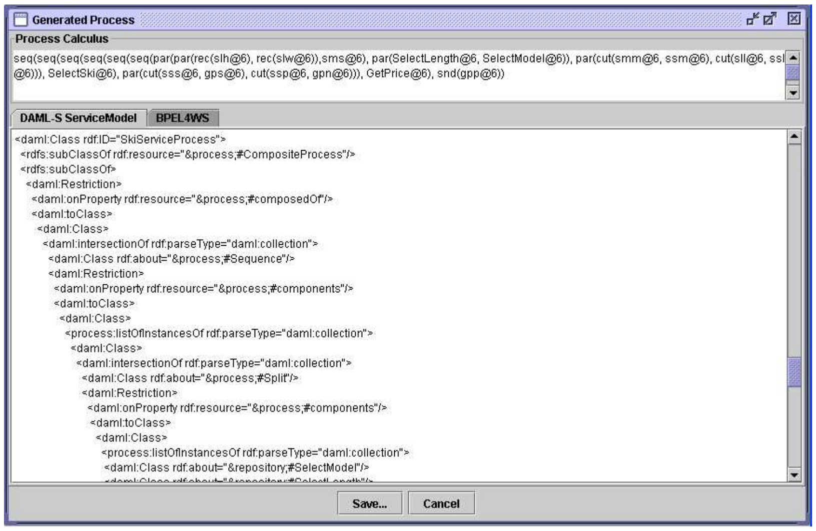
Basic GUI features of the prototype are depicted in Fig. 7.3. The interface of the required service is presented in the **ServiceProfile** panel (upper right) and the dataflow of the component atomic services is presented in the **Service-Model** panel (lower right). The screenshot in Fig. 7.3 shows that the composed service is combined from five atomic services. For each Web service, the detailed information of functionalities and non-functional attributes is displayed in the left hand side panel. The bottom panel demonstrates semantic relationships between parameters.



Nr	Namespace
0	file:///home/jinghai/projects/jwsst/data/ICWS04Service.daml
1	file:///home/jinghai/projects/jwsst/data/ICWS04Profile.daml
2	file:///home/jinghai/projects/jwsst/data/ICWS04Process.daml
3	http://bromstad.idi.ntnu.no/services/ontologies.daml
4	file:///home/jinghai/projects/jwsst/data/ICWS04RepService.daml
5	file:///home/jinghai/projects/jwsst/data/ICWS04RepProfile.daml
6	file:///home/jinghai/projects/jwsst/data/ICWS04RepProcess.daml

Figure 7.4: Namespaces mapping window.

To use a short name to present the classes and properties, we use number to represent the namespaces that allow us to disambiguate between things that may use the same name. The mapping between the numbers and the actual URLs for the namespaces can be obtained from the Namespace mapping windows shown in Figure 7.4. For example, the actual URI for the service `SelectSki@6` is file:///home/jinghai/projects/jwsst/data/ICWS04RepService.daml#SelectSki.



```
seq(seq(seq(seq(seq(seq(par(par(rec(slh@6), rec(slw@6)),sms@6), par(SelectLength@6, SelectModel@6)), par(cut(smm@6, ssm@6), cut(sll@6, ssl
@6))), SelectSki@6), par(cut(sss@6, gps@6), cut(ssp@6, gpn@6))), GetPrice@6), snd(gpp@6))
```

```
<daml:Class rdf:ID="SkiServiceProcess">
 <rdfs:subClassOf rdf:resource="&process;#CompositeProcess"/>
 <rdfs:subClassOf>
  <daml:Restriction>
   <daml:onProperty rdf:resource="&process;#composedOf"/>
   <daml:toClass>
    <daml:Class>
     <daml:intersectionOf rdf:parseType="daml:collection">
      <daml:Class rdf:about="&process;#Sequence"/>
      <daml:Restriction>
       <daml:onProperty rdf:resource="&process;#components"/>
       <daml:toClass>
        <daml:Class>
         <process:listOfInstancesOf rdf:parseType="daml:collection">
          <daml:Class>
           <daml:intersectionOf rdf:parseType="daml:collection">
            <daml:Class rdf:about="&process;#Split"/>
            <daml:Restriction>
             <daml:onProperty rdf:resource="&process;#components"/>
             <daml:toClass>
              <daml:Class>
               <process:listOfInstancesOf rdf:parseType="daml:collection">
                <daml:Class rdf:about="&repository;#SelectModel"/>
```

Figure 7.5: A screen shot of generated process model.

The generated composite service is presented both graphically and in code. The generated process model corresponding to the composite service illustrated in the **ServiceModel** panel is presented to the user through a process document window (figure 7.5). The result process calculus is presented at the upper text area. The corresponding DAML-S ServiceModel document and BPEL4WS document are presented as well. The user can store the documents as files to local disk.

7.6 Summary

In this Chapter, we have elaborated the four different components forming the Web service composition system, namely, the LL theorem prover based on UMA Forum, the DAML-S translator based on Jena, the semantic reasoner based on FaCT, and the GUI. The implementation is of prototype quality and we have tried to integrated those components through the AGORA multi-agent system. The different components are represented by different software agents.

Besides the LL theorem prover and the DAML-S translator, the implementation of other components are straightforward, because the supporting platforms of those components have been already well developed by there developers. In this Chapter, we discuss the LL theorem prover in greater detail. In particular, we give the code to present the generation of the process calculus formula from the LL proof in UMA Forum. The correctness of the generated result is guaranteed by the correctness of the Forum programs. The quality of the result and the performance evaluation are the subjects of the coming evaluation chapter.

Chapter 8

Evaluation

This chapter evaluates the thesis mainly by evaluating the answers to the research questions, the contributions and the comparition with other similar systems. Most of the result in this thesis has been published in international conferences and journals. We also give a discussion of the performance issue of the implemented system.

8.1 Answers to the Research Questions

The main research question, which is presented in Chapter 1, is:

> *How can we enable the intelligent agents to automatically retrieve and compose Web services to achieve the goals specified by their users?*

The main research question has been answered, in general, by the design and implementation of an agent-based system for automated Web Service retrieval and composition. The system includes three parts. First, we proposed an automated Web service composition approach using LL-based program synthesis. The service composition problem is represented as a process of LL theorem proving, and the process model of the composite service can be extracted from the complete proof directly. Second, the user's requirement and the available services are described by Semantic Web markup languages, so that the composition system can deal with the semantic relations between the concepts that are used to define the types of Web service parameters. Finally, all the components for service composition are implemented as software agents. Such design, on one hand, enables the distributed manner of Web services; on the other hand, makes the system scalable.

Our answers to the underlying and more specific research questions are as follows:

RQ1: What is Web services and how are they composed?

- We model the existing Web services from the black box view of modular software. The Web services are viewed in terms of observable interfaces with the internals of the system hidden from view. The interfaces of a Web service include the input/output parameters, pre-/post conditions and the non-functional attributes.
- The process model of the generated composite service is represented by a process calculus that specifies the data flow and control flow among the component services. The data flow is represented by a multiset of pairs including an output parameter and an input parameter, which reveals the message of data transferred from the output of one service to the input of another service. The control flow presents the execution orders of component services, including sequence, parallel and choice.
- In our experimental system, we use software agents in AGORA-based multi-agent system to represent the service providers, service requesters and the facilitating components for service publishing, matching, composing and invocation. The interaction of the agents are presented as interaction protocol of multi-agent systems. This is not the only solution to tackle the problem of Web service supporting platform. A lot of systems, such as .NET, WebSphere and Apache can be used as Web service platforms as well.

RQ2: How can we automatically compose the Web services via logic-based program synthesis?

- In the logical language level, we specify the existing Web services as the logical axioms. In Chapter 3, we have presented a guideline and formal semantics for the translation from the DAML-S ServiceProfile to the LL axioms.
- The foundation of our automated Web service composition approach is based on deductive program synthesis. Deductive program synthesis is stemmed from the observation that proofs are equivalent to programs because each step of a proof can be interpreted as a step of a computation. This transforms the problems of software composition or program synthesis into a theorem proving task. This observation is also known in the literature as the "Curry-Howard isomorphism", or the "proofs as programs paradigm". Comparing to the traditional program synthesis methods that are based on functional programming and sequential process, our service composition approach considers much more about the relationship between LL and the concurrent features in Web services. In our research, we take advantage of the previous results relating LL with concurrent process modeling, such as Petri Nets and π-calculus. Therefore, the

process model of the generated composite service can be extracted directly from the complete proof, enabled by attaching the process information specified by the π-calculus based process calculus.

- The concrete method used to extract the process model from the LL proof is presented in Chapter 4. In general, the process information is attached to the logical formulae as proof terms. We study the inference rules of MAILL fragment of LL from the point of view of giving each inference rule a concrete computational interpretation in the context of Web service composition. We have developed a set of inference rules with proof terms in Table 4.2. The inference rules enable us to construct the process from the steps of the LL proof. We use a process calculus to present the process model of the composite service formally, and the final result is presented to the users in XML based Web service process languages, such as DAML-S ServiceModel or BPEL4WS. The formal semantics of the translation from the process calculus to the Web service process languages is given in Chapter 4.

RQ3: Is it reasonable to present the Web service composition problem in the context of multi-agent framework?

- The main purpose to use multi-agent system as the platform for our Web service composition approach is to develop a scalable and flexible environment concerning the dynamic features of Web services. In the the agent-based approach, both service providers, service requesters and other facilitators, such as theorem prover, semantic reasoner, translator, etc., are represented by software agents. The service providers and requesters can be active in performing mutual searching. The design of facilitator agents enables easily to add new functions or components to the legacy system.
- We use the agent interaction protocol to present the communication among the participants in Web service composition system. In particular, this design leverages the heterogeneity between the LL theorem prover and DL-based semantic reasoner. The two parts are implemented separately and used for different purposes. The agents who represent the different parts can interact and exchange messages by unifying the message format. We have developed a concrete interaction model in Chapter 6.

RQ4: How can we use Semantic Web markup for facilitating the Web service composition task?

- We use the Semantic Web technology to enable the type inference. The service composition approach allows reasoning with types from the service specifications. We use the types to define the valid dataflow for the composite services. For example, when a value of type A is expected in an input port, a value of any subtype of type A can also be accepted. The type specification is built using the Semantic Web markup language, DAML+OIL. We have developed a semantic reasoner based on Description Logic (DL) to detect the subtyping relations between the classes and properties used to describe the Web services. Since DL is developed to ensure logical consistency of the model developed by DAML+OIL language and to answer logic queries including satisfiability, equivalence and subsumption, such task is straightforward.

- Under the condition that the subtyping relationships between the DAML+OIL classes and properties are known, we can use the subtyping relationships in the LL theorem prover in a way that the Web service composition system takes the Semantic Web information into consideration as well. This is supported by the help of two techniques. First, we use LL propositions to represent the classes and properties in DAML+OIL language. Second, we have proven that the subtyping relationship can be represented by the linear implication in LL. We developed a set of subtyping rules, which can be incorporated with the regular LL inference rules for theorem proving.

8.2 Contributions

A more detailed summary of the contributions of this thesis are given below:

C1: An generic framework is developed for the purpose of presenting an abstract process of the automated Semantic Web service composition. The framework is presented in Chapter 2. This framework is in high-level abstraction, without considering a particular language, platform or method used for the service composition tasks. The aim of the framework is to give the basis to discuss similarities and differences of the available automated service composition methods. This framework can be considered as a contribution to the automated Web service composition community, regardless of which composition method is used. Its contribution to the state-of-the art is that it is very likely to be the first published framework that generalizes the process of automated Semantic Web service composition tasks.

C2: A specific system based on the generic platform in **C1** has been developed. The system uses LL-based program synthesis to construct the composite Web services in an automated manner. The general process of such system is as follows. First, the system uses a Semantic Web service language (DAML-S) for external presentation of existing Web services, while, internally, the services are presented by extralogical axioms and proofs in LL. The translation from DAML-S to LL axioms is elaborated in Chapter 3. Second, we use a process calculus to present the process model of the composite service formally. The process calculus is attached to the LL inference rules in the style of type theory. Thus the process model for a composite service can be generated directly from the complete proof. Further, the process calculus can be translated into either DAML-S ServiceModel or BPEL4WS. The process extraction and translation is presented in Chapter 4. The contributions of this part include: 1) this is the first effort that demonstrate both the guideline and the formal semantics for the translation between Web service languages and the logical languages; 2) it applies the "Proof as Process" view from Abramsky [8] into a practical application of the concurrent system.

C3: Application of the subtyping inference rules that are used for semantic reasoning is discussed. The main contribution is that we present the semantic relations as LL inference figures, thus the semantic reasoner and LL theorem prover can operate together. It is natural to notice that the formal language used to describe the services and the language used to present semantic information are very different. However, no previous work attempting to tackle such heterogeneity problem has been reported so far. In our setting where LL is used to present the services and DL-based languages, such as DAML+OIL is used to provide semantic information, the heterogeneity has been tackled by a unified presentation that is introduced in Chapter 5.

C4: We have developed an agent architecture as the platform for Web service provision and composition. The main feature of the agent-specific architecture is that it provides Web service composition task with proactivity, reactivity, social ability and autonomy, while the usage of DAML-S, FIPA ACL and application domain specific ontologies provides a standardized medium for Web service advertisement, interaction, and deployment. Another reason that we consider agent architecture is that it enables the plugin of the facilitator agents. This design enables the different components for Web service composition system, such as the theorem prover, semantic reasoner and translator to integrate to each other in a loosely coupled manner. The agent architecture can be regarded as a contribution in developing the Web service composition system because the different components and their interaction for such system is never elaborated in previous research efforts. In Chapter 6, we demonstrate the agent model and the interaction model for such agent architecture. We further elaborate the detail implementation of each facilitator agents in

Chapter 7.

8.3 Comparison with Other Methods

A lot of research efforts have been reported in the area of automated composition based on the semantic description of Web services. Comparing to the other systems reviewed in Chapter 2, the approach in this thesis has a number of properties:

- Our method, at the first time, unify the presentation of both service functionalities and non-functional attributes together in the composition process. Other AI planning based methods mostly consider only the functionalities, namely the input/output parameters and pre-/post condition of the service. Although in [111], non-functional attributes are treated as filters and they are specified by the user as constraints, it is an extra step that is outside the planning process. In our approach, both functionalities and non-functional attributes are treated as LL propositions. The inputs, preconditions and non-functional constraints are represented as LL resources, while the outputs, postconditions and non-functional results are represented as LL goals. For the LL theorem prover, the functionalities and non-functional attributes are treated without any difference.

- Most Web service composition approaches based on AI planning make an closed world assumption, meaning that if a literal does not exist in the current world, its truth value is considered to be *false*. In logic programming this approach is called *negation as failure*. The main trouble with the closed world assumption, from Web service construction's perspectives, is that merely with truth literals we cannot express that new information has been acquired. For instance, one might want to describe that after sending a message to another agent, an identity number to the message will be generated. Thus during later communication the ID could be used. The ID number is a new literal that can not be considered in close world assumption. While using resource-conscious logics, like LL or transition logic, this problem is treated implicitly and there is no need to distinguish informative and truth values. The reason is that the literals are regarded as resources to be consumed or generated instead of truth values. Therefore, we can view generated literals as references to informative objects. If a new literal is inserted into the world model, new piece of information will be available. LL provides an elegant framework for modeling incomplete knowledge – although before plan execution only partial knowledge is available, and during execution more details would be revealed.

- Another advantage of using LL instead of classical logic is that LL provides a richer set of connectives. In particular, the additive conjunction and disjunction of LL distinguish the internal choice and the external choice. The internal choice is enabled by additive conjunction &. An applicable interpretation for this operator is a dialog box that asks users to choose "yes" or "no". From the computational viewpoint, the dialog box service produces two outputs and either of them chosen by the users would lead the process to follow different branches. A typical external choice situation is that a service may produce one of several alternative outputs every time when it is executed. In particular, this is the case with exception handling. In that case, the output produced by the service depends on the execution environment instead of the user. None of the existing Web service specification languages supports internal choice, so this issue is not applicable in current stage. However, we regard this as an important branch of future development of Web service languages.

- The multi-agent platform for the Web service composition system helps to leverage the heterogeneity among the different components used for the service composition. Although a lot of approaches are designed for the Semantic Web services composition using AI planning technique, there is no concrete method that has been proposed for the interaction between the AI planner and the Semantic Web reasoner. In our approach, we try to tackle such problem using two techniques. First, we introduce a set of subtyping rules for the LL theorem prover, so that the subtyping relationships in Semantic Web description can be represented as LL implication. The second technique is to model the LL theorem prover and semantic reasoner as software agents. Thus the interaction between those two components can be specified as agent interaction protocols. Furthermore, by using the concept facilitator agent, different theorem provers and semantic reasoner can be attached or detached from the system in a loosely coupled manner. This enhance the scalability and flexibility of the system. We have present the concrete method in this thesis.

8.4 Publications

This thesis is partly based on papers presented at international conferences and journals. This Section summarized the results presented in a collection of selected papers. Each paper is summarized briefly together with its relation to the contributions and answering research questions in the thesis.

Paper A - Mihhail Matskin and Jinghai Rao. "Value-Added Web Services Composition using Automatic Program Synthesis". In *Proceedings of Web services, e-Business, and the Semantic Web Workshop, WES'2002*, Toronto, Canada, May, 2002. LNCS 2512, Springer-Verlag. [75]

This paper presents the idea to apply the software synthesis and composition methods to value-added Web services composition. The program synthesis method is based on Structural Synthesis of Programs(SSP). The paper can be regarded as an early exploration in the approaches of Web service composition using logic-based program synthesis. The content of this paper does not appear in this thesis, however, this paper shows the pace of the development of our research work.

Paper B - Jinghai Rao, Peep Küngas and Mihhail Matskin. "Application of Linear Logic to Web Service Composition". In *Proceedings of the First International Conference on Web Services, ICWS'2003*, Las Vegas, USA, June, 2003. CSREA Press. [102]

This paper presents our first attempt to use Linear Logic as the formal description language for the modeling of Web services. This paper focuses on the presentation of Web services, in particular, elaborating the translation mechanism from the DAML-S ServiceProfile to LL axioms. The content of this paper corresponds directly to Chapter 3 in the thesis.

Paper C - Sobah Abbas Petersen, Jinghai Rao and Mihhail Matskin. "Virtual Enterprise Formation with Agents – an Approach to Implementation". In *Proceedings of the 2003 IEEE/WIC International Conference on Intelligent Agent Technology, IAT'2003*, Halifax, Canada, October, 2003, IEEE Computer Society Press. [100]

The papers describes an implementation of virtual enterprises using AGORA multi-agent system. The method used in this paper is quite close to what presented in Chapter 6. In a virtual enterprise setting, the software agents represent the interested parties to form the enterprise. Therefore the whole system are presented in terms of multi-agent architecture, including agent model and interaction model. We also use the same manner to present the Web service composition system.

Paper D - Xiaomeng Su, Mihhail Matskin and Jinghai Rao. "Implementing Explanation Ontology for Agent System". In *Proceedings of the 2003 IEEE/WIC International Conference on Web Intelligence, WI'2003*, Halifax, Canada, October, 2003. IEEE Computer Society Press. [114]

The overall issue addressed in this paper is to improve semantic interoperability among and across agent systems. The interoperation is expressed in terms of an explanation ontology shared by the agents who participate in the communication. The explanation ontology is defined in a way being general enough to support a variety of explanation mechanisms. The paper describes the explanation ontology and provides a working through example illustrating how the proposed generic ontology can be used to develop specific explanation mechanism. The example has been implemented using AGORA multi-agent system. The paper demonstrates an early result of semantic integration in the context of multi-agent system. This is the basis of Chapter 5 and Chapter 6.

Paper E - Jinghai Rao and Xiaomeng Su. "Toward the Composition of Semantic Web Services". In *Proceedings of the Second International Workshop on Grid and Cooperative Computing, GCC'2003*, Shanghai, China, December, 2003. LNCS 3033, Springer-Verlag. [105]

The main contribution of this paper is to propose a general framework for the task of automated Semantic Web service composition. The functional settings of the framework are discussed and techniques for DAML-S presentation, Linear Logic presentation, and semantic integration are presented. A prototype implementation of the approach is proposed to fulfill the task of representing, composing and handling of the services. This paper contributes to parts of Chapter 2 and Chapter 7 in the thesis.

Paper F - Jinghai Rao, Peep Küngas and Mihhail Matskin. "Logic-Based Web Service Composition: from Service Description to Process Model". In *Proceedings of the 2004 IEEE International Conference on Web services, ICWS'2004*, San Diego, USA, July, 2004. IEEE Computer Society Press. **This paper received a best paper runners up award at the conference.** [104]

This paper focuses on the extraction of the process model from the LL proof. We developed a process calculus to present the composite service formally. The process calculus is attached to the LL inference rules in the style of type theory. Thus the process model for a composite service can be generated directly from the proof. This paper is the basis of Chapter 4 in the thesis.

Paper G - Jinghai Rao and Xiaomeng Su. "A Survey of Automated Web Service Composition Methods". In *Proceeding of the First International Workshop on Semantic Web Services and Web Process Composition, SWSWPC'2004*, San Diego, USA, July, 2004. To be published by Springer-Verlag's LNCS series. [106]

This paper has aimed to give an overview of recent progress in automatic Web services composition. The paper firstly proposes a five-step model for Web services composition process. The composition model consists of service presentation, translation, process generation, evaluation and execution. In these five steps, the paper concentrates on the methods of composite Web services process generation. The paper gives the introduction and comparison with selected methods to support this step. These methods are enabled either by workflow research or AI planning. This paper provides content to Chapter 2.

Paper H - Peep Küngas and Jinghai Rao. "Symbolic Agent Negotiation for Semantic Web Service Exploitation". In *Proceedings of the Fifth International Conference on Web-Age Information Management, WAIM'2004*, Dalian, China, July, 2004. LNCS 3129, Springer-Verlag. [62]

This paper presents an architecture and a methodology for agent-based Web service discovery and composition. The paper assumes that Web services are described with declarative specifications like DAML-S. Then symbolic reasoning methods can be applied while searching for or composing new services in an automated manner. The paper proposes that symbolic agent negotiation enables dynamic Web service discovery and composition to be applied in a more distributed environment. Symbolic negotiation, as demonstrated in this paper, is a mixture of distributed planning and information exchange. Therefore, by using symbolic negotiation for automated service composition, additional information collection and integration are supported during service composition process. The agent architecture presented in this paper has directly contribution to Chapter 6. The symbolic negotiation part does not appear in the thesis. We consider it as future research direction.

Paper I - Jinghai Rao, Peep Küngas and Mihhail Matskin. "Composition of Semantic Web Services using Linear Logic Theorem Proving. To appear in *Information Systems Journal - Special Issue on the Semantic Web and Web Services*, Elsevier Science Publisher. [103]

This paper appeared in journal is a concentration of Chapter 2, 3, 4, 5 and part of 7 in the thesis. This is my first publication to concern all steps of the composition of Semantic Web services using LL theorem proving. The approach uses a Semantic Web service language (DAML-S) for external presentation of Web services, while, internally, the services are presented by extralogical axioms and proofs in LL. LL, as a resource conscious logic, enables us to capture the concurrent features of Web services formally (including parameters, states and non-functional attributes). We use a process calculus to present the process model of the composite service. The process calculus is attached to the LL inference rules in the style of type theory. Thus the process model for a composite service can be generated directly from the complete proof. We introduce a set of subtyping rules that defines a valid dataflow for composite services. The subtyping rules that are used for semantic reasoning are presented with LL inference figures. We propose a system architecture where the DAML-S translator, LL theorem prover and semantic reasoner can operate together.

Besides the publications, I have made several public presentations during the years I pursue the study towards the doctoral degree. Below is a list of selected presentations.

- "Using Program Synthesis to Facilitate Web Services Composition", presented at *Computer Science Graduate Student Conference, CSGSC'2002*, Trondheim, Norway, May, 2002.

- "Value-Added Web Service Composition using Automatic Program Synthesis", presented at *Web services, e-Business, and the Semantic Web Workshop, WES'2002*, Toronto, Canada, May, 2002.

- "Adapting Structural Synthesis of Programs for Web Services Composition", presented at *the Seminar organized by IS-group, IF-group and KS-group*, Trondheim, Norway, October, 2002.

- "Application of Linear Logic to Web Service Composition", presented at *the First International Conference on Web Services, ICWS'2003*, Las Vegas, USA, June, 2003.

- "Toward the Composition of Semantic Web Service", presented at *the Second International Workshop on Grid and Cooperative Computing, GCC'2003*, Shanghai, China, December, 2003.

- "Logic-Based Web Service Composition: from Service Description to Process Model", presented at *the 2004 IEEE International Conference on Web services, ICWS'2004*, San Diego, USA, July, 2004.

- "A Survey of Automated Web Service Composition Methods", presented at *the First International Workshop on Semantic Web Services and Web Process Composition, SWSWPC'2004*, San Diego, USA, July, 2004.

- "Semantic Web Service Composition via Logic-based Program Synthesis", presented at *the DIS lunch meeting*, Trondheim, Norway, October, 2004.

8.5 Performance Evaluation

The performance is not the main concern of the result of this thesis. The reason is that our tasks focus on the presentation of Web services by logical languages and the design of the general framework instead of the development of new theorem prover and semantic reasoner. We take advantage of the third-party developed tools for the most time-consuming work, namely, using UMA Forum as LL theorem prover and using FaCT as semantic reasoner. The overall performance of the Web service composition task highly depends on the performance of these tools together with the preferred quality of the results. For example: if the user doesn't require the theorem prover to report the complete result, the computational time can be decreased significantly.

We ran an experiments to test if the performance of our system is eligible in a runtime response manner. The experiments is run on a Sun SPARC-server with four 1281 MHz sparcv9 processor, 16 GB of RAM, and under a Solaris operating system. We use SICStus Prolog 3.11.1 to as the platform of UMA Forum. The FaCT semantic reasoner is supported by GNU Common Lisp 2.2.

The test set is generated according to the setting presented in Table 8.1. The number of services is up to 1000, which means that we have maximum 1000 axioms in theorem proving. Each axiom that represents a service has no more than 50 literals. The literals here represent service parameters, conditions and non-functional attributes. We also limit the total number of different literals to 5000. We assume that there are 500 pairs of existing subtyping relations. The axioms are randomly generated from the above constraints.

Variable	*Range*
Number of services	10 – 1000
Number of literals	50 – 5000
Literals for each service	10 – 50
Number of subtyping relations	5 – 500

Table 8.1: Experimental settings.

We evaluate the time for generating the process model of the composite service(Table 8.2). We consider three execution times. The first one is the time for semantic reasoner. The second one is time for LL theorem prover to general complete result. The third one is time by sacrifice the completeness of the result. We add two restriction to the experiment for the third time. First, the theorem prover stops whenever ten results have been found. Second, the introduction rule of the "of course" modality for each literal is applied no more than twice continuously. The reason for the latter restriction is because the using of "of course" modality significantly increase the complexity of LL theorem proving [66].

Number of Service	*Time for semantic reasoning (ms)*	*Time for theorem proving (ms)*	*Time for theorem proving (incomplete) (ms)*
10	1	1850	1160
100	13	5.1×10^4	5180
500	73	3.1×10^6	1.8×10^4
1000	146	no result	4.4×10^4

Table 8.2: Process generation time.

The result shows the performance is in a reasonable scope by sacrifice the completeness of the result. For 1000 services, the generation time is no more than one minute, which is acceptable for runtime composition. However, the performance still needs to be improved for a larger number of services.

8.6 Limitations

The limitation is related to the chosen logical framework, which is the propositional fragment of LL. We assume that the composition is correct in a parameter pair if the output is of the same type or subtype of the input. However, in service composition, it is useful to know the properties expected of an input to the Web services or guaranteed of an output, where these properties may be based on the values, rather than the data types. Moreover, considering the service execution, "uncertain" result may still be produced although the type correctness is guaranteed. To understand why this happens, we give an example shown in [101]:

> Example: Consider the following services: personname-to-address and address-to-phone. Further, assume that multiple people can live at the same address and that different people living at the same address may use different phones. For the composite service name-to-phone, the data flow chains the name-to-address and address-to-phone services. And the output type of name-to-address and the input type of address-to-phone are the same. Now, if John and Jack live at the same address but have different phone numbers, the composite service would return both their phone numbers when asked for John's phone. The reason this happens is because address neither uniquely determine the phone number nor the name.

It is clear that the propositional logic can only guarantee the type correctness instead of the correctness of the instances of the type during the runtime. First-order LL should allow better presentation of Web service properties. In particular, first-order LL may provide the express power to present the instances of the types. Moreover, first-order LL may provider the ability of calculation of more complex math type, other than in propositional LL, we can only present positive integer in consumable quantitative attributes.

However, first-order LL may suffer from poor efficiency (also completeness is not ensured). Before moving to the first-order LL we would like to gain experience in practical usage of propositional LL and better understand its niche in the whole Web service composition process. We consider the using of first-order LL to present Web Services as our future research direction.

We ignore some features for Web service system such as security control and transaction management. For example, the service repository should only be exposed to the authorized users. Considering the service repository is operated by multiple distributed users, the transaction control is definitely necessary to keep the consistency of the data. Here, we apply a simple transaction control mechanism. Any time when our user is accessing the repository, either reading or updating, the repository is locked and refuses access from other users. This is not the best solution. However, to elaborate the transaction control is so complex that it is beyond the scope of this paper.

8.7 Lessons Learned

The core part of the work presented in this thesis has been published in international conferences and journals. We have received valuable feedback through the reviewing process. We also learned a lot from the presentation of the papers in the conference, including both questions from the audience and the discussion with researchers who have the same research interest. One of the difficulties for this thesis is the multi-disciplinary nature that covers at least the area of logic-based program synthesis, Web Services and multi-agent systems. It was often difficult to get the right focus for the papers because, depending on the reviewers and their background, the contents of the paper were interpreted differently. However, this helped enrich the feedback that was received for this work.

Chapter 9

Conclusion

9.1 Summary of Results and Contributions

In this thesis we describe an approach to automatic Semantic Web service composition. Our approach has been directed to meet the main challenges in service composition. First, it is autonomous so that the users do not required to analyze the huge amount of available services manually. Second, it has good scalability and flexibility so that the composition is better performed in a dynamic environment. Third, it solves the heterogeneous problem because the Semantic Web information is used for matching and composing Web services. The applicable scenario for this approach is: given the specification of available Web services and user's requirement, an automated agent or program can generate a composition of available services which matches the requirement of the user. The result generation process is fully autonomous without the intervention from the user. The process generation should rely on the specification of Web services, including the functionalities and non-functional attributes.

We have introduced a solution that uses LL based program synthesis to solve the problem. In the solution, the Web services and the user's requirement are both specified by DAML-S ServiceProfile. We use a translator to translate them into LL formulae. The description of existing Web services is encoded as LL axioms, and the requirement to the composite service are specified in form of a LL sequent to be proven. We use an automated LL theorem prover to determine whether the sequent can be proven by the axioms. If the answer is positive, a process model of the composite service can be extracted from the generated proof. The process model is presented formally by a process calculus that is inspired by π-calculus. The calculus is attached to the proof as terms in a type system. The process calculus can be further translated into document in either DAML-S ServiceModel or BPEL4WS languages to presented to the user. The user can use the document to invoke the composite service.

To develop such a solution, we have developed the techniques from the following sources.

First, we argue that the composition approach must also support a specification language that enables users to specify services in an easy way. In this thesis, we use the Semantic Web service markup language DAML-S as the specification language for both the profiles and the processes of Web services. The composition result can additionally be presented by BPEL4WS in order to support the invocation of services. We have developed the translation mechanism considering the translation between the Web service language and the logical languages. In Chapter 3, we elaborate the approach to translate the DAML-S ServiceProfile to the LL axioms. The translation between the process calculus and the Web service process language, such as DAML-S ServiceModel and BPEL4WS is demonstrated in Chapter 4.

Second, we have developed a concrete approach to extract the process of the process of the composed service from the proof. First, we study the the relationship between LL and the concurrent process modeling, such as π-calculus. This relationship is taken up by Abramsky's "proofs as processes" view and further supported by the work of Bellin and Scott. Second, we have developed a set of inference rules of MAILL fragment of LL considering the proof terms from the point of view of giving a concrete computational interpretation in service composition. These inference rules have been implemented based on a LL programming language, UMA Forum.

Third, we use the Semantic Web technology to enable the type inference. Our composition approach allowing reasoning with types from the service specifications is presented in Chapter 5. We use the types to define the valid dataflow inside the composite services. For example, if the output type of one service is subtype of the input type of another service, it is safe to transfer data from the output to the input. The type system is built above the Semantic Web markup language, DAML+OIL. We take advantage of Jena as the DAML+OIL interpreter and the FaCT system as the semantic reasoner with the foundation of SHIQ(D) fragment of DLs.

Fourth, in order to improve the interoperability flexibility and the scalability of the composition system, we use an agent-based approach to support the service composition platform. The agent system is based on an agent architecture, AGORA. In AGORA, we have three kind of agents, service providers, service requesters and the facilitator agents. We elaborate the multi-agent architecture, including the agent model and the interaction model in Chapter 6. The use of facilitator agents, in particular, the theorem prover agent and the semantic reasoner agent, enables the interoperability between the LL prover and the DL-based semantic reasoner through agent communication.

We argue that LL theorem proving, combined with semantic reasoning offers a flexible approach to the success to the composition of Web services. LL, as a logic for specifying concurrent programming, provides higher expressive powers to model Web services than classical logic. Further, the agent-based design enables the different components for Web service composition system to integrated to each other in a loosely coupled manner.

The contributions of this thesis is summarized as follows. First, an generic framework is developed for the purpose of presenting an abstract process of the automated Semantic Web service composition. Second, a specific system based on the generic platform in has been developed. Third, applications of the subtyping inference rules that are used for semantic reasoning is discussed. Fourth, an agent architecture is developed as the platform for Web service provision and composition. The detail of these contributions are elaborated in Chapter 8.

9.2 Directions for Future Work

We see two directions for continuing this work. One direction is to use partial deduction to enable a more flexible composition method in a distributed environment. Another direction is to apply software adaptation to enhance the reusability of the previous composition result.

9.2.1 Partial Deduction

We consider to use partial deduction [67] to extend the current composition method to provide more flexibility and efficiency. First, our experience with the Web service composition shows that users are not always able to completely specify the requirement of the composite service. We consider to apply the principle of partial deduction to provide more flexibility to the user. In case there is no result that can match the user's requirement exactly, it is possible to present the user some partial result and lead the user to update the requirement in a heuristic manner. Second, using partial deduction enables cooperative theorem proving over multiple distributed service repositories. The main advantage of distributed service composition over centralized approaches is that the composition systems are able to engage new Web services more effectively, however, the challenging problem is to scale the proposed composition method up to work on a large number of network nodes. Some initiate results about using partial deduction and symbolic negotiation for Web service composition based on a multi-agent architecture have been reported in [62].

9.2.2 Reusability of Composition Result

Different users may have similar requests to the composite services. It is likely to improve the system efficiency if the solution of the new request can be adapted from the previously generated solutions. Collections of previous solutions can be stored in the service repository to provide generic services and software architectures, called design patterns [43]. It is also possible to generate implementation skeletons that can be refined into full implementations. However, this may cause a repository scalability problem, where the size of the repository must grow combinatorially as additional features are supported. Another problem is the dynamic feature of the service repository. If some services in the previous solution are on longer available, the solution needs adaptation and the services can be replaced by other available services. A software component adaptation method based on traditional input/output signatures proposed by Penix [99] may provide help in this problem.

Bibliography

[1] Axis – an Java implementation of the SOAP protocol. Online: http://ws.apache.org/axis/.

[2] The DARPA Agent Markup Language homepage. Online: http://www.daml.org.

[3] FIPA Agent Communication Language. http://www.fipa.org/.

[4] IBM Web services tutorial. Online : http://www-106.ibm.com/developerworks/webservices/.

[5] Jena - Semantic Web framework for Java. Online: http://jena.sourceforge.net.

[6] FIPA Interaction Protocol Library Specification, 2000. http://www.fipa.org/specs/fipa00025/XC00025E.html.

[7] S. Abramsky. Computational interpretations of Linear Logic. *Theoretical Computer Science*, (111):3–57, 1993.

[8] S. Abramsky. Proofs as processes. *Theoretical Computer Science*, 135(1):5–9, 1994.

[9] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services – Concepts, Architectures and Applications*. Springer-Verlag, 2004.

[10] J.-M. Andreoli. Logic programming with focusing proofs in Linear Logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

[11] J.-M. Andreoli and R. Pareschi. Communication as fair distribution of knowledge. In *Proceedings of OOPSLA 91*, pages 212–229, 1991.

[12] J.-M. Andreoli and R. Pareschi. Linear Objects: Logical processes with built-in inheritance. *New Generation Computing*, 9(3–4):445–473, 1991.

[13] T. Andrews et al. Business Process Execution Language for Web Services (BPEL4WS) 1.1. Online: http://www-106.ibm.com/developerworks/webservices/library/ws-bpel, May 2003.

[14] A. Ankolekar et al. DAML-S: Semantic markup for Web serivces. In *Proceedings of the International Semantic Web Workshop*, 2001.

[15] A. Ankolekar, F. Huch, and K. Sycara. Concurrent execution semantics for DAML-S with subtypes. In *Proceedings of the First International Semantic Web Conference(ISWC)*, Seattle, USA, 2002.

[16] S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OilEd: a reasonable ontology editor for the Semantic Web. In *Proceedings of the Joint German/Austrian Conference on Artificial Intelligence (KI 2001)*, number 2174 in LNAI, pages 396–408. Springer-Verlag, 2001.

[17] S. Bechhofer, I. Horrocks, P. F. Patel-Schneider, and S. Tessaris. A proposal for a Description Logic interface. In *Proceeding of the 1999 International Workshop on Description Logics (DL'99)*. CEUR Pubblication, 1999.

[18] G. Bellin and P. J. Scott. On the π-calculus and Linear Logic. *Theoretical Computer Science*, 135(1):11–65, 1994.

[19] T. Bellwood et al. Universal Description, Discovery and Integration specification (UDDI) 3.0. Online: http://uddi.org/pubs/uddi-v3.00-published-20020719.htm.

[20] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.

[21] T. Berners-Lee and E. Miller. The Semantic Web lifts off. *ERCIM News*, (51):9–10, October 2002.

[22] A. J. Bonner and M. Kifer. The state of change: a survey. In *International Seminar on Logic Databases and the Meaning of Change, Transactions and Change in Logic Databases*, pages 1–36. Springer-Verlag, 1998.

[23] M. Bory. A logical basis for modular software and systems engineering. In *Proceedings of the SOFSEM'98: Theory and Practice in Informatics*, number 1521 in LNCS, pages 19–35. Springer-Verlag, 1998.

[24] D. Box et al. Simple Object Access Protocol (SOAP) 1.1. Online: http://www.w3.org/TR/SOAP/, 2001.

[25] bpmi.org. The Business Process Modeling Language (BPML). Online: http://www.bpmi.org/bpml.esp.

[26] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0 (second edition), W3C recommendation 6 october 2000. http://www.w3.org/TR/2000/REC-xml-20001006.

[27] D. Brickley and R. V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. Online: http://www.w3.org/TR/rdf-schema/.

[28] J. J. Carroll, I. Dickinson, C. Dollin, D. Deynolds, A. Seaborne, and K. Wilkinson. Jena: Implementing the Semantic Web recommendations. In *Proceedings of the Thirteenth International World Wide Web Conference (WWW 2004)*, New York, USA, May 2004. ACM Press.

[29] F. Casati, S. Ilnicki, and L. Jin. Adaptive and dynamic service composition in EFlow. In *Proceedings of 12th International Conference on Advanced Information Systems Engineering(CAiSE)*, Stockholm, Sweden, June 2000. Springer Verlag.

[30] F. Casati, M. Sayal, and M.-C. Shan. Developing e-services for composing e-services. In *Proceedings of 13th International Conference on Advanced Information Systems Engineering(CAiSE)*, Interlaken, Switzerland, June 2001. Springer Verlag.

[31] R. Chinnici et al. Web Services Description Language (WSDL) 1.2. Online: http://www.w3.org/TR/wsdl/.

[32] L. Chung. Representation and utilization of non-functional requirements for information system design. In R. Anderson, J. J.A. Bubenkio, and A. Solvberg, editors, *The Third Int. Conf. on Advanced Information Systems Enginerring*, pages 5–30, Trondheim, Norway, 1991. Springer-Verlag.

[33] A. G. Cohn and S. M. Hazarika. Qualitative spatial representation and reasoning: An overview. *Fundamenta Informaticae*, 46(1-2):1–29, 2001.

[34] M. Dean et al. OWL Web Ontology Language 1.0 reference. Online: http://www.w3.org/TR/owl-ref/, July 2002.

[35] G. Denker, L. Kagal, T. Finin, M. Paolucci, and K. Sycara. Security for DAML Web services: Annotation and matchmaking. In *Proc. 2nd International Semantic Web Conference (ISWC2003)*, Sanibel Island, Florida, USA, October 2003.

[36] M. Doerr, N. Guarino, M. López, E. Schulten, M. Stefanova, and A. Tate. State of the art in content standards., November 2001.

[37] J. M. Dunn. *Handbook of Philosophical Logic*, volume III, chapter Relevance logic and entailment, pages 117–224. D. reidel Publishing Company, 1986.

[38] ebXML Technical Architecture Team. ebXML technical architecture specification v1.0.4. online: http://www.ebxml.org, February 2001.

[39] U. Engberg and G. Winskel. Petri Nets as models of Linear Logic. In *Proceedings of the 15th Colloquium on Trees in Algebra and Programming, CTAP 1990*, volume 431 of *LNCS*, pages 147–161, Copenhagen, Denmark, 1990. Springer-Verlag.

[40] D. Fensel, I. Horrocks, F. van Harmelen, S. Decker, M. Erdmann, and M. Klein. OIL in a nutshell. In R. Dieng et al., editors, *Proceedings of the European Knowledge Acquisition Conference (EKAW-2000)*, number 1937 in LNAI. Springer-Verlag, October 2000.

[41] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an agent communication language. In *Proceedings of the third International Conference on Information and Knowledge Management (CIKM'94)*, 1994.

[42] G. Frank. A general interface for interaction of special purpose reasoners within a modular reasoning system. In *Question Answering Systems, Papers from the 1999 AAAI Fall Symposium*, pages 57–62, 1999.

[43] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.

[44] J.-Y. Girard. Linear Logic. *Theoretical Computer Science*, 50:1–102, 1987.

[45] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description Logic programs: Combining logic programs with Description Logic. In *Proceedings of the 12th International Conference on the World Wide Web (WWW 2003*, Budapest, Hungary, 2003.

[46] N. Guarino. Formal ontology in information systems. In *Proceedings of Formal Ontology in Information Systems, FOIS'98*, pages 3–15. IOS Press, 1998.

[47] V. Haarslev and R. Möller. Description of the RACER system and its applications. In *Proceedings of the International Workshop on Description Logic (DL-2001)*, Stanford, USA, August 2001.

[48] J. Harland, D. Pym, and M. Winikoff. Programming in lygon: An overview. In *Proceedings of the Fifth International Conference on Algebraic Methodology and Software Technology*, pages 391–405, July 1996.

[49] J. S. Hodas and D. Miller. Logic programming in a fragment of intuitionistic Linear Logic. *Information and Computation*, 110(2):327–365, 1994.

[50] I. Horrocks. The FaCT system. In *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98*, number 1397 in LNAI, pages 307–312. Springer-Verlag, 1998.

[51] I. Horrocks. DAML+OIL: a Description Logic for the Semantic Web. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 25(1):4–9, March 2002.

[52] I. Horrocks and P. F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *Proceedings of the 2003 International Semantic Web Conference (ISWC 2003)*, number 2870 in LNCS, pages 17–29. Springer-Verlag, 2003.

[53] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for very expressive description logics. *Logic Journal of the IGPL*, 8(3):239–263, 2000.

[54] I. Horrocks, F. van Harmelen, and P. F. Patel-Schneider. Reference description of the DAML+OIL (march 2001) ontology markup language. http://www.daml.org/2001/03/reference.html, 2001.

[55] W. Howard. The formulae-as-types notion of construction. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490, London, 1980. Academic Press.

[56] R. Hull, M. Benedikt, V. Christophides, and J. Su. E-service: A look behind the curtain. In *Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, San Diego, USA, June 2003.

[57] IBM. BPWS4J – the IBM Business Process Execution Language for Web Services Java Run Time. Online: http://www.alphaworks.ibm.com/tech/bpws4j.

[58] IBM. WebSphere software platform. Online: http://www-306.ibm.com/software/info1/websphere/index.jsp.

[59] Y. Kalfoglou and M. Schorlemmer. Ontology mapping: The state of the art. *Knowledge Engineering Review*, 18(2), 2003.

[60] P. Küngas. RAPS – Resource-Aware Planning System. Online: http://www.idi.ntnu.no/ peep/RAPS/index.html.

[61] P. Küngas and M. Matskin. Linear Logic, partial deduction and cooperative problem solving. In *Proceedings of the First International Workshop on Declarative Agent Languages and Technologies (in conjunction with AAMAS 2003), DALT'2003, Melbourne, Australia, July 15, 2003*. Springer-Verlag, 2004.

[62] P. Küngas, J. Rao, and M. Matskin. Symbolic agent negotiation for Semantic Web service exploitation. In *Proceedings of the 5th International Conference on Web-Age Information Management, WAIM'2004*, number 3129 in LNCS, Dalian, China, July 2004. Springer-Verlag.

[63] J. Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65(3):154–170, 1958.

[64] S. Lämmermann. *Runtime Service Composition via Logic-Based Program Synthesis*. PhD thesis, Department of Microelectronics and Information Technology, Royal Institute of Technology, June 2002.

[65] O. Lassila and R. R. Swick. Resource Description Framework (RDF) model and syntax specification, W3C recommendation. online: http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/, February 1999.

[66] P. Lincoln. Deciding provability of Linear Logic formulas. In *London Mathematical Society Lecture Note Series*, volume 222. Cambridge University Press, 1995.

[67] J. W. Lloyd and J. C. Shepherdson. Partial evaluation in logic programming. *Jornal of Logic Programming*, 11:217–242, 1991.

[68] P. López and E. Pimentel. A lazy splitting system for forum. In *Proceedings of the Joint Conference on Declarative Programming — APPIA-GULP-PRODE'97*, pages 247–258, Grado, Italy, 1997.

[69] M. Lumpe. *A π-Calculus Based Approach for Software Composition*. PhD thesis, Institute of Computer and Applied Mathematics, University of Bern, 1999.

[70] Z. Manna and R. Waldinger. Fundamentals of deductive program synthesis. *IEEE Transactions on Software Engineering*, 18(8):674–704, 1992.

[71] Z. Manna and R. J. Waldinger. A deductive approach to program synthesis. *ACM Transactions on Programming Languages and Systems*, 2(1):90–121, 1980.

[72] D. Martin et al. DAML-S(and OWL-S) 0.9 draft release. Online: http://www.daml.org/services/daml-s/0.9/, May 2003.

[73] M. Matskin. Collaborative advertising over internet with agents. In *Proceedings of the Twelfth International Workshop on Database and Expert Systems Applications (DEXA 2001)*, Munich, Germany, September 2001.

[74] M. Matskin, O. J. Kirkeluten, S. B. Krossnes, and Ø. Sæle. *Infrastructure for Agents, Muddlti-Agents, and Scalable Multi-Agent Systems*, volume 1887 of *LNCS*, chapter Agora: An Infrastructure for Cooperative Work Support in Multi-Agent Systems, pages 28–40. Springer Verlag, 2001.

[75] M. Matskin and J. Rao. Value-added Web services composition using automatic program synthesis. In *Web Services, E-Business, and the Semantic Web, CAiSE 2002 International Workshop, WES2002*, number 2512 in LNCS, Toronto, Canada, 2002. Springer Verlag.

[76] M. Matskin and E. Tyugu. Structural synthesis of programs and its extensions. *Computing and Informatics Journal*, 20(1), 2001.

[77] D. McDermott. Estimated-regression planning for interactions with Web services. In *Proceedings of the 6th International Conference on AI Planning and Scheduling*, Toulouse, France, 2002. AAAI Press.

[78] R. E. McGrath, A. Ranganathan, R. H. Campbell, and M. D. Mickunas. Use of ontologies in pervasive computing environments. Technical report, Department of Computer Science, University of Illinois at Urbana-Champaign, April 2003.

[79] S. McIlraith and D. Mandell. Comparison of DAML-S and BPEL4WS. Online: http://www.ksl.stanford.edu/projects/DAML/Webservices/DAMLS-BPEL.html, Knowledge Systems Lab, Stanford University, September 2002.

[80] S. McIlraith and T. C. Son. Adapting Golog for composition of Semantic Web services. In *Proceedings of the 8th International Conference on Knowledge Representation and Reasoning(KR2002)*, Toulouse, France, April 2002.

[81] S. McIlraith, T. C. Son, and H. Zeng. Semantic Web services. *IEEE Intelligent Systems*, 16(2):46–53, March/April 2001.

[82] B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid. Composing Web services on the Semantic Web. *The VLDB Journal*, 12(4), November 2003.

[83] L. Meredith and S. Bjorg. Contracts and types. *Communications of the ACM*, 46(10), October 2003.

[84] Microsoft Corporation. ASP.NET Web. Online: http://www.asp.net.

[85] Microsoft Corporation. The Component Object Model specification, October 1995. Draft Version 0.9.

[86] D. Miller. The π-calculus as a theory in Linear Logic: Preliminary results. In E. Lamma and P. Mello, editors, *Proceedings of the 1992 Workshop on Extensions to Logic Programming*, number 660 in LNCS, pages 242–265. Springer-Verlag, 1993.

[87] D. Miller. Forum: A multiple-conclusion specification logic. *Theoretical Computer Science*, 165(1):201–232, 1996.

[88] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied logic*, 51:125–157, 1996.

[89] D. Miller and E. Pimentel. Linear Llogic as a framework for specifying sequent calculus. In *Proceedings of the Annual European Summer Meeting of the Association for Symbolic Logic*, Utrecht, Netherlands, August 1999.

[90] R. Milner. *Logic and Algebra of Specification*, chapter The Polyadic Pi-calculus: A tutorial. Springer-Verlag, 1993.

[91] J. C. Mitchell and K. Apt. *Concepts in Programming Languages*. Cambridge University Press, 2001.

[92] S. Narayanan and S. McIlraith. Simulation, verification and automated composition of Web service. In *Proceedings of the 11th International World Wide Web Conference*, Honolulu, Hawaii, USA, May 2002. ACM. presentation available at http://www2002.org/presentations/narayanan.pdf.

[93] H. S. Nwana. Software agents: An overview. *The Knowledge Engineering Review*, 11(3):1–40, 1996.

[94] Object Management Group. The Common Object Request Broker Architecture (CORBA) core specification, December 2002. Version 3.0.

[95] J. O'Sullivan, D. Edmond, and A. T. Hofstede. What's in a service? Towards accurate description of non-functional service properties. *Distributed and Parallel Databases*, 12:117–133, 2002.

[96] J. Pan and I. Horrocks. Web ontology reasoning with datatype groups. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, number 2870 in LNCS, pages 47–63. Springer, 2003.

[97] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Semantic matching of web services capabilities. In *First International Semantic Web Conference*, Sardinia, Italy, June 2002.

[98] A. Parasuraman, V. Zeithaml, and L. Berry. SERVQUAL: A multiple-item scale for measuring consumer perceptions of service quality. *Journal of Retailing*, 64(1):12–40, 1988.

[99] J. J. Penix. *Automated Component Retrieval and Adaptation Using Formal Specifications*. PhD thesis, Division of Research and Advanced Studies of the University of Cincinnati, 1998.

[100] S. A. Petersen, J. Rao, and M. Matskin. Virtual enterprise formation with agents - an approach to implementation. In *Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology (IAT'2003)*, Halifax, Canada, October 2003. IEEE Publications.

[101] S. R. Ponnekanti and A. Fox. SWORD: A developer toolkit for Web service composition. In *Proceedings of the 11th World Wide Web Conference*, Honolulu, HI, USA, 2002.

[102] J. Rao, P. Küngas, and M. Matskin. Application of Linear Logic to Web service composition. In *Proceedings of the 1st International Conference on Web Services*, Las Vegas, USA, June 2003. CSREA Press.

[103] J. Rao, P. Küngas, and M. Matskin. Composition of Semantic Web services using Linear Logic theorem proving. *Information Systems Journal – Special Issue on the Semantic Web and Web Services*, 2004. to appear.

[104] J. Rao, P. Küngas, and M. Matskin. Logic-based Web services composition: from service description to process model. In *Proceedings of the 2004 International Conference on Web Services*, San Diego, USA, July 2004. IEEE.

[105] J. Rao and X. Su. Toward the composition of Semantic Web services. In *Proceedings of the Second International Workshop on Grid and Cooperative Computing, GCC'2003*, volume 3033 of *LNCS*, Shanghai, China, December 2003. Springer-Verlag.

[106] J. Rao and X. Su. A survey of automated Web service composition methods. In *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, SWSWPC'2004*, LNCS, San Diego, USA, July 2004. Springer-Verlag. to appear.

[107] H. Schuster, D. Georgakopoulos, A. Cichocki, and D. Baker. Modeling and composing service-based and reference process-based multi-enterprise processes. In *Proceeding of 12th International Conference on Advanced Information Systems Engineering (CAiSE)*, Stockholm, Sweden, June 2000. Springer Verlag.

[108] M. Sheshagiri, M. desJardins, and T. Finin. A planner for composing service described in DAML-S. In *AAMAS Workshop on Web Services and Agent-Based Engineering*, Melbourne, Australia, 2003.

[109] S. Shrivastava, L. Bellissard, D. Fliot, et al. A workflow and agent based platform for service provisioning. In *Proceedings of the 4th IEEE/OMG International Enterprise Distributed Object Computing Conference(EDOC 2000)*, Makuhari, Japan, September 2000. IEEE Computer Society Press.

[110] M. Siddalingaiah. Overview of ebXML. Online: http://dcb.sun.com/practices/webservices/, August 2001.

[111] E. Sirin, J. Hendler, and B. Parsia. Semi-automatic composition of Web services using semantic descriptions. In *Proceedings of Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS2003*, 2002.

[112] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions of Computer Science*, 29(12):1104–1113, 1980.

[113] T. Sollazzo, S. Handschuh, S. Staab, and M. Frank. Semantic Web service architecture – evolving Web service standards toward the Semantic Web. In *Proceedings of the 15th International FLAIRS Conference*, Florida, USA, 2002.

[114] X. Su, M. Matskin, and J. Rao. Implementing explanation ontology for agent system. In *Proceedings of the 2003 IEEE/WIC International Conference on Web Intelligence, WI'2003*. IEEE Computer Society Press, October 2003.

[115] K. Sycara, S. Widoff, M. Klusch, and J. Lu. Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. *Autonomous Agents and Multi-Agent Systems*, 5(3):173–203, September 2002.

[116] S. Thatte. XLANG: Web services for business process design. Online: http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm.

[117] A. Tsalgatidou and T. Pilioura. An overview of standards and related technology in Web services. *Distributed and Parallel Databases*, 12:135–162, 2002.

[118] M. Turner, D. Budgen, and P. Brereton. Turning software into a service. *IEEE Computer*, 36(10), October 2003.

[119] M. T. Tut and D. Edmond. The use of patterns in service composition. In *The First Workshop of Web Services, e-Business and the Semantic Web*, Toronto, Canada, 2002.

[120] J. Vaucher. Xprolog for java. `http://www.iro.umontreal.ca/~vaucher/XProlog/`.

[121] W3C. Web services architecture requirements. online: http://www.w3.org/TR/wsa-reqs, October 2002.

[122] R. Waldinger. Web agents cooperating deductively. In *Proceedings of FAABS 2000, Greenbelt, MD, USA, April 5–7, 2000*, volume 1871 of *LNCS*, pages 250–262. Springer-Verlag, 2001.

[123] L. Wischik. New directions in implementing the π-calculus. In *CaberNet Radicals Workshop*, 2002.

[124] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[125] D. Wu, E. Sirin, J. Hendler, D. Nau, and B. Parsia. Automatic Web services composition using SHOP2. In *Proceedings of the Workshop on Planning for Web Services*, Trento, Italy, June 2003.

[126] J. Yang and M. P. Papazoglou. Web component: A substract for Web service reuse and composition. In *Proceedings of the 14th International Conference for Advanced Information Systems Engineering (CAiSE)*, volume LNCS 2348, Toronto, Canada, May 2002.

Index