Peep Küngas

Distributed Agent-Based Web Service Selection, Composition and Analysis through Partial Deduction

Doctoral thesis
for the degree of doktor ingeniør

Trondheim, July 2006

Norwegian University of Science and Technology
Faculty of Information Technology,
Mathematics and Electrical Engineering
Department of Computer and Information Science

NTNU
Norwegian University of Science and Technology

Doctoral thesis
for the degree of doktor ingeniør

Faculty of Information Technology,
Mathematics and Electrical Engineering
Department of Computer and Information Science

ISBN 82-471-7783-8 (printed version)
ISBN 82-471-7781-1 (electronic version)
ISSN 1503-8181

Doctoral theses at NTNU, 2006:21

Printed by NTNU-trykk

Abstract

In order to facilitate agile business and support online partnership formation, many modern information systems are designed to support interoperability. This tendency has become mainstream with advancements in distributed systems and is supported by the Internet and industrial standards (or standard proposals) like XML, WSDL, SOAP and BPEL. However, increasing complexity of distributed information systems puts forward requirements to high adaptivity of distributed information systems.

Alternative technologies have been proposed in academia for supporting adaptivity in information systems. These technologies include cooperative problem solving, agent technology, Web service composition, the Semantic Web and P2P networks. Despite the high potential of such technologies, only few efforts have been made to integrate them into commercial applications.

The main aim of this thesis is to investigate and determine to what extent automated Web service composition can be applied in practice. We have implemented a tool, which exploits symbolic negotiation for distributed Web service composition. While applying our approach to automated Web service composition we demonstrate that automated composition methods are useful for analysing Web service domains. More specifically, potential interactions and synergy between different Web services' domains can be discovered by using automated composition.

The main contributions of the thesis are the following. First, it formalises partial deduction for linear logic. Also soundness and completeness of the formalism is proved. Second, it formalises symbolic negotiation with respect to partial deduction and identifies relations between cooperative problem solving and symbolic negotiation. Third, a multi-agent system, utilising the developed formal methods, is designed and implemented. Moreover, the multi-agent system is extended with P2P capabilities. Fourth, a distributed Web service composition tool is described and implemented. Finally, automated Web service composition is evaluated over a set of existing governmental and commercial Web services.

To Õnnela

Contents

List of Figures

List of Tables

Preface

This thesis is submitted to the Norwegian University of Science and Technology 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), Norwegian University of Science and Technology (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, Royal Institute of Technology, Stockholm, Sweden. This work was partially supported by the Norwegian Research Foundation in the framework of Information and Communication Technology (IKT-2010) program through the ADIS project.

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 studies. I am grateful for the opportunity to work and study at IDI, which provided an excellent environment to cross-fertilise research ideas.

I would like to thank all people at IDI, among them my former colleague Jinghai Rao for discussions on automated Web services composition. Sari Hakkarainen, Yun Lin, Jennifer Sampson and Csaba Veres provided useful hints on the Semantic Web and ontologies. Xiaomeng Su was of great help while introducing me information retrieval, document classification and ontology mapping techniques. Several discussions over semantics in P2P networks were held together with Hao Ding. Arne Sølvberg provided general guidance and support during my employment and studies at IDI. They all contributed indirectly to the content of this thesis.

Additionally I would like to thank my former master students Terje Olsen and Aanund Austrheim at IDI, NTNU for implementing the GUI of ADIS tool, which was used throughout experiments. My master students from Royal Institute of Technology, Kista, Sweden, Shenghua Liu and Magnus Wallin contributed also to the ADIS tool. Shenghua implemented JXTA message transportation protocol for JADE and Magnus experimented with automated Web service classification and semantic Web service annotation. Both results have been incorporated into the ADIS tool.

The evaluation part of this thesis was supported by X-Road project team members including Ahto Kalja from Tallinn University of Technology and others. Their as-

sistance greatly simplified collection of governmental Web services' descriptions and allowed thus to compare commercial Web services with governmental ones.

Finally, I would like to thank my family, especially my beloved wife Õnnela for her enduring support and endless love.

Peep Küngas
May 31, 2006

Chapter 1

Introduction

Web services are revolutionising the way industry and public sector operate. As the Web evolves into the Semantic Web, the myriad of available Web services are being described such that their automated processing is possible. Machine-understandable descriptions enable automatic discovery, use, and composition of Web services.

Automated Web service composition is a methodology for constructing composite Web services from already existing ones with minimal developer intervention. The methodology enhances reuse of existing Web services and thereby reduces time plus man hours devoted to manual Web service development. Given requirements for a new Web service, the methodology provides methods to either construct the Web service automatically or to identify that the Web service has to be implemented from scratch. Some methods even facilitate construction of partial solutions, by determining specifications for new Web services whose implementation would render a partial solution complete.

Traditionally automated Web service composition has been applied in centralised configuration. It means that all available atomic Web services are known prior to the composition. Furthermore, the composition process has been usually performed by a single program. We, in contrary, consider automated composition in a distributed environment, where composition is performed by multiple agents simultaneously. Furthermore, we assume that each agent has knowledge only about a fraction of available Web services. Thus, in order to compose a required Web service, agents may have to collaborate.

In this chapter we explain what initiated our interest into the subject and identify some related technologies. We also specify research problems under consideration, describe our contribution and present the organisation of the thesis.

1.1 Motivation

In order to facilitate agile business and to support dynamic partnership formation, modern Information Systems (IS) are designed to facilitate interoperability between themselves. This tendency has become mainstream with advancements in distributed

systems. In particular, the Internet and industrial standard proposals like XML, WSDL and BPEL have contributed to this progress. However, increasing maintenance costs of distributed IS-s have created the need for more adaptive information systems.

Simultaneously to industrial efforts, alternative technologies have been proposed for automating information system integration in academia. The technologies include multi-agent systems, cooperative problem solving mechanisms, automated Web service composition, the Semantic Web and P2P networks. Anyway, despite of the potential of the mentioned technologies, only few efforts have been made to embed them into commercial applications.

Therefore, one aim of this thesis is to help bridge the gap between industrial and academic efforts by evaluating the applicability of automated Web service composition. Although several articles have proposed methods for automated Web service composition, according to author's knowledge, none of the methods have been evaluated in industrial settings so far.

In order to provide the evaluation we first propose a formal framework for encoding and solving automating Web service composition problems. Then we design an architecture for facilitating distributed Web service composition. Finally, we evaluate our implemented automated composition method over a set of commercial and governmental Web services. In this way we demonstrate how formal methods could be applied in practical cases. Furthermore, our evaluation results indicate settings and applications where automated Web service composition could be useful. We combine both theoretical and empirical results in order to answer the proposed research questions.

1.2 Background

As stated previously, we are interested in determining applicability of automated (distributed) Web service composition in industrial settings. Given the inter-disciplinary nature of the problem, efforts and results in different fields impacted the content of this thesis. There are four research areas, which influenced the material in this thesis. First of all we were inspired by research in logics and their applications. We strongly believe that fixed semantics of an input language leads to better results in automation. Additionally, we were motivated by research in agents and multi-agent systems. Given the distributed nature of our problem, agents would provide us with communication primitives and the degree of proactivity, which is required for distributed Web service composition. Finally, our work is supported by the achievements in the Semantic Web initiative and the current progress in automated Web service composition.

1.2.1 Logics

Traditionally, logics have been studied as a branch of philosophy. Since the mid-1800s logics have been commonly studied in mathematics, and, even more recently, in com-

puter science. As a science, logic investigates and classifies the structure of statements and arguments, and devises schemata by which these are codified. The scope of logics is therefore very large, including reasoning about probability and causality, time and resources, etc.

Logics are extensively applied in the fields of artificial intelligence and computer science since these fields provide a rich source of problems in formal logic. In the 1950s and 1960s, researchers predicted that when human knowledge could be expressed using logic with mathematical notation, it would be possible to create a machine that reasons, or Artificial Intelligence (AI). This turned out to be more difficult than expected because of the complexity of human reasoning.

Anyway, nowadays all kinds of applications of logic in computer science are studied under the framework of *computational logic* [162]. Computational logic centers around the famous definition:

$$Algorithm = Logic + Control.$$

According to this view, algorithms consist of a problem description (the logic part) along with a strategy to carry out useful computations on this description (the control part). Computational logic is devoted to the ideal of a programmer who concentrates solely on the description of the problem and spends no time at all on the actual computation mechanism. This unique paradigm of "declarative programming" leads to programs that are fast and simple to develop and easy to understand and maintain. Moreover, the rigorous use of logic revolutionises the whole field of software verification. Instead of having to undergo a test phase, which necessarily cannot give more than just some confirmation that a program will do what it is supposed to do, the development of programs and systems that are guaranteed to be correct is the second ideal of computational logic. Therefore we are going to apply a computational logic for describing how our computational machinery is supposed to function.

1.2.2 Multi-agent systems

The study of Multi-Agent Systems (MAS) focuses on systems where many intelligent agents interact with each other. The agents are considered to be autonomous entities, such as software programs or robots. Their interactions can be either cooperative or self-interested. That is, the agents can share a common goal (e.g. an ant colony), or they can pursue their own interests (as in the free market economy).

Progress in MAS has resulted in communications languages, interaction protocols, and agent architectures that facilitate the development of multi-agent systems. Research in MAS has drawn ideas from many disciplines outside of AI, including biology, sociology, economics, organization and management science, complex systems, and philosophy. Several attempts have been made in order to formalise Cooperative Problem Solving (CPS) (see Chapter 2). Most of them are based on classical or modal logics. In particular, Wooldridge and Jennings [199] provide a formalisation of CPS process where a multi-modal logic is used as a formal specification language.

However, since the multi-modal logic lacks a strategy for generating constructive proofs of satisfiability, the formalisation does not lead to direct execution of specifications. Moreover, since modal logics (like classical logic) lack the mechanism for keeping track of resources, it is not possible for agents reason about resources. Therefore we take advantage of a resource-conscious logic to formalise CPS and agent negotiation.

It has been indicated in [77] that negotiation is the most fundamental and powerful mechanism for managing inter-agent dependencies at run-time. Negotiation may be required both for self-interested and cooperative agents. It allows to reach a mutually acceptable agreement on some matter by a group of agents.

Anyway, in order to implement a MAS, a suitable MAS architecture is required. This is another issue, which we have to resolve. Combination of a MAS architecture together with a computational logic would naturally lead into an adaptive system.

1.2.3 The Semantic Web

The Semantic Web [15] provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. It is a collaborative effort led by W3C with participation from a large number of researchers and industrial partners. It is based on the Resource Description Framework (RDF), which integrates a variety of applications using XML for syntax and URIs for naming. Generally speaking, the Semantic Web is an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation [15].

Therefore, efforts in the Semantic Web are directly beneficial to automated Web service composition. Practical and efficient Web services selection, provision and composition requires more detailed descriptions of Web services compared to those provided by WSDL documents and tModels in UDDI. These additional descriptions should facilitate discovery, integration and composition of Web services in a more efficient way than it is supported now. The Semantic Web project has resulted in initiatives (see [32] for a short overview of approaches to model the Semantic Web services) like WSMO, SWSO, OWL-S and WSDL-S, which are designed to model and describe the meanings and intended usage of Web services more precisely than it is currently supported by industrial standards.

1.2.4 Automated Web service composition

Automated Web service composition is about taking a set of Web service descriptions plus requirements for a new Web service and transforming them into a workflow, which implements a required composite Web service. In order to do that, composition algorithms should be sustained by machine-readable descriptions of Web services. Moreover, these descriptions should unambiguously indicate the semantics of associated Web services.

With the increased interest in the Web services paradigm, composition of Web services has become of primary importance. Several languages for describing Web services and their composition are currently being defined and seek to become standards. The current leading proposals are the Web Service Description Language (WSDL) and the Business Process Execution Language for Web Services (BPEL4WS), an industry-developed flow language. These industrial efforts are extended by academic efforts like WSMO, SWSO, OWL-S and WSDL-S as mentioned already.

By now many methods have been proposed for composing Web services automatically from existing OWL-S and WSML-like Web service descriptions. The methods range from AI planning [132, 172, 200] to automated theorem proving [134, 160, 194] and graph search algorithms. For a more comprehensive review of published methods see Chapter 2.

1.3 Research questions

The overall research question is the following:

> *How can automated Web service composition be applied in practice, and to what extent?*

In order to answer this general question, we have to answer to the following more detailed research questions:

RQ1 How could we automate Web service composition?

RQ2 How to distribute automated Web service composition?

RQ3 How to extend Web services to support automated Web service composition?

RQ4 Which problems in industry can be solved through automated Web service composition?

RQ5 What are the limitations of automated Web service composition?

1.4 Proposed solutions

In order to answer the proposed research questions we first formalise partial deduction (PD) for linear logic (LL) and prove its completeness and soundness. Based on the PD formalisation we formalise cooperative problem solving (CPS) and symbolic negotiation in such a way that multiple PD threads could be run simultaneously.

Then we design and implement a multi-agent system (MAS) for applying the CPS/symbolic negotiation formalisation. Additionally we extend the MAS with P2P capabilities. Finally the MAS is applied for automated distributed Web service composition and experiments are conducted.

The experiments indicate applicability of automated Web service composition in certain cases. They also determine which supporting technologies are required to increase the degree of applicability or to make automated Web service composition applicable at all. Moreover, the results help us to propose some novel applications of automated composition.

1.4.1 Formalisation of partial deduction

Partial Deduction (PD) (or partial evaluation of logic programs, which was first introduced by Komorowski [89]) is known as one of optimisation techniques in logic programming. Given a logic program, PD derives a more specific program while preserving the meaning of the original program. Since the program is more specialised, it is usually more efficient than the original program.

For instance, let A, B, C and D be propositional variables and $A \rightarrow B$, $B \rightarrow C$ and $C \rightarrow D$ computability statements in a logical framework. Then possible partial deductions are $A \rightarrow C$, $B \rightarrow D$ and $A \rightarrow D$. It is easy to notice that the first corresponds to forward chaining (from facts to goals), the second to backward chaining (from goals to facts) and the third could be either forward or backward chaining or even their combination.

Although the original motivation behind PD was deduction of specialised logic programs with respect to a given goal, our motivation for PD is a bit different. Namely, it turns out that PD could be applied to find *partial solutions* of problems written as logical formalisms. In our case, given a formal specification of a problem, if we fail to solve the entire problem, we apply PD to generate partial solutions for it. These partial solutions can be used for analysing why no solutions were found.

As a logical formalism for application of PD we use Linear Logic [56]. LL has been advocated [82] to be a computation-oriented logic. Although PD has been formalised for several frameworks, including fluent calculus [111], normal logic programs [120], etc., it turns out that there is no work considering PD for LL. Our goal is to fill this gap by providing a formal foundation of PD for LL.

As a part of the formalisation we define PD steps as inference figures in LL. While using those inference figures instead of basic LL rules, we can achieve higher efficiency during proof search. Multiple examples are presented to demonstrate application of PD.

Although LL provides a rich formalism for representing resources and agent capabilities, it still lacks a construction for specifying another important aspect of dynamic systems, namely time. Therefore we extend the proposed PD formalisation with the notion of time through usage of temporal LL (TLL). This approach gives us an opportunity to go beyond barely resource-oriented problem solving and to solve a larger class of problems than LL would alone.

1.4.2 Formalisation of symbolic negotiation

Symbolic negotiation is regarded in the field of computer science as a process, where parties try to reach an agreement on the high-level means for achieving their goals by applying symbolic reasoning techniques. Formalisation of symbolic negotiation contributes both to the studies of human behaviour and multi-agent systems. In the former field the rational part of human reasoning can be modelled, while in the latter field self-organising systems with adaptive behaviour can be implemented.

Until now mainly game theoretical negotiation has been applied in multi-agent systems. The latter is based on a numerical utility function, which is applied to choose a negotiation strategy. However, game theoretical negotiation has shortcomings in three areas. First, a negotiation strategy is chosen before negotiation starts and cannot be changed during negotiation. Second, the participating agents cannot hide their internal states from each-other, since the states are used to define the utility function. And third, the negotiation process is based on numerical information, which is hardly interpretable by humans. Thus human participants may not be able to follow the negotiation process by their own and thus cannot evaluate the validity of the results. The last disadvantage is one of the reasons why human users may not trust their software agents.

Symbolic negotiation in contrast is based on logical formalisms and thus overcomes the previously mentioned disadvantages. It means that encapsulation of agent preferences, resources and goals is supported. Additionally, the negotiation process and the result of the process is declarative and thus more easily interpretable than numerical information. And finally, agents are allowed to dynamically adjust their negotiation strategies during negotiation as well.

We believe that distributed theorem proving is a natural metaphor for symbolic negotiation. Therefore we formalise CPS through PD and then extend the CPS framework with symbolic negotiation rules. Initial ideas for applying LL for symbolic multi-agent negotiation have been presented in [65]. We regard symbolic negotiation as interleaved CPS and plan modification. We also formalise coalition formation process and analyse its effect to CPS and symbolic negotiation. Regarding other previously proposed coalition formation methods, which are oriented to task allocation, our method could be described as goal-oriented.

We would like to underline that from a computational point of view, we can regard CPS as AI planning and symbolic negotiation as plan reuse/repair. It has been shown [142] that from problem solving point of view in general neither planning from scratch nor plan repair has an advantage over eachother. Therefore we expect both CPS and symbolic negotiation to be computationally equivalent. Moreover, both CPS and symbolic negotiation lead to the same results as we prove in this thesis.

However, compared to CPS, symbolic negotiation provides a more human-like way of problem solving, which can be more naturally followed by human participants. In addition, symbolic negotiation may encode a sort of search heuristics, which would make CPS computationally less demanding. These heuristics, however, are not dis-

cussed in this thesis.

The cooperative problem solving has been considered to consist of four steps [199]:

- recognition of potential for cooperation
- team formation
- plan formation
- plan execution

An important feature of our approach is that we do not separate team and plan formation into different processes and that negotiation is embedded into the reasoning. Although this approach does not preserve the accepted structure of CPS, we think that it may be more natural for representing computational aspects of CPS, where team and plan formation processes interact with each other.

1.4.3 MAS architecture

In order to implement the formalised symbolic negotiation procedure, we first have to construct a MAS architecture for it. The architecture identifies the main entities, which would participate in symbolic negotiation. Also agent communication protocols are determined.

We take advantage of the AGORA multi-agent environment [127], which was developed with the intention to support cooperative work between agents. The system consists of 2 types of components (nodes)—agents and agoras. Agoras are cooperative nodes which facilitate agent communication, coordination and negotiation. An agora node contains *default agents* and *registered agents*. In our scenario, the default agents are Agora Manager and Negotiator. Agora Manager implements general agora functions, such as service matchmaking and agent/service registration, while Negotiator applies symbolic negotiation. Registered agents represent Web service providers and requesters.

1.4.4 P2P extension of MAS

The increasing popularity of P2P systems (such as Overnet, Kazaa and Gnutella) for file sharing, indicates general interest in resource sharing. However the current P2P systems suffer at least from two drawbacks. First, they are mostly designed for sharing either data or CPU power, but not both in the same system. Moreover, in the case of CPU sharing, the executable computational processes are expected to be known *a priori* for each participant (like in SETI@Home). Second, the current P2P network nodes still lack a degree of proactivity, which would provide higher degree of autonomy, rationality and fairness.

In contrary, MAS-s still seem to lack enough capabilities to reorganise themselves in dynamic environments. In particular, despite of the intelligent behaviour assigned

to agents, MAS architectures are currently mostly designed manually. Therefore combining MAS-s and P2P networks would extend the capabilities of both architectures.

We shall exploit a structured P2P network to self-organise MAS infrastructure for efficient resource discovery. Structured P2P networks allow more efficient resource discovery compared to non-structured topologies, where the network is flooded with messages in order to locate a resource. We take advantage of Chord algorithm, which manages structured P2P networks.

Despite of the promotion of cross-fertilisation of peers and agents, we take more conservative position by deciding that P2P- and agent-related issues should not be mixed. While P2P handles issues related to indexing and efficient location of resources, agents are those who initiate the actions on P2P networks. Thus MAS would be a control layer for P2P networks, whereas P2P network is just another communication medium for MAS.

1.4.5 Automated Web service composition

Due to its expressiveness we choose LL for describing Web services and Web service composition problems, as proposed by Rao et al [160]. Therefore it is natural to apply our PD framework for automating Web service composition. Moreover, by regarding *distributed* Web service composition as symbolic negotiation we are able to exploit the previously proposed MAS architecture as a platform for Web service composition in distributed environments.

While LL allows us to describe Web services in a natural way, having PD as a basis for automated Web service composition has other advantages. Namely, by using PD for automated Web service composition, we can determine the reasons for not finding a solution for a requested composite Web service. Since PD allows *partial* solutions, compared to all-or-nothing capability of pure LL theorem proving, we can collect a set of partial solutions and look deeper into them. Thereby we can determine some new Web services, which have to be implemented in order to get a composite solution. Moreover, analysis of partial solutions may also indicate inconsistencies within existing descriptions of Web services.

It should be noted that in order to apply automated composition as proposed by Rao et al [160], Web services must be semantically annotated. Thus WSDL descriptions of Web services should be accompanied with OWL-S or WSML documents, which allow the semantics of Web services to be described. However, annotation languages are out of the scope of this thesis, since we consider only the formal aspect of automated Web service composition.

1.4.6 Implementation

Since we are interested in experimental analysis of our symbolic negotiation method, we implemented the proposed PD mechanism and the corresponding agent architecture. The MAS is built using the JADE agent development environment and integrates

PD into the proposed symbolic negotiation framework. We also implemented a tool for automated and distributed Web service composition. By combining automated reasoning, multi-agent systems and Web services, the tool inherently provides flexibility, which would not have been achieved by using these technologies individually.

The tool supports semantic annotation and automated composition of Web services. Composite Web services are graphically displayed and can be exported into BPEL4WS documents. The tool also supports teamwork by allowing collaborative composition of Web services by remote users of the same tool.

1.4.7 Applicability of automated Web service composition

Recently the field of Web services has gained focus both in industry and academia. While industry has been mostly interested in standardisation and promotion of the technology, academia has been looking for ways to fit the technology into other frameworks, such as the Semantic Web. Web services' research in the Semantic Web context varies from automated annotation [31,70,71,150,163] and ontologies [9] to automated Web service composition [132,134,172,186,200]. While automated annotation intends to provide methods to extract semantics from existing Web services, research related to ontologies and Web services has focused to the modelling of Web services. The latter has led to initiatives like WSMO, SWSO, OWL-S and WSDL-S.

Efficient selection and integration of inter-organisational and heterogeneous Web services at runtime is an important requirement for Web services provision. For instance, if no single Web service satisfies the required functionality, existing Web services could be combined together to fulfill the request. This aspect of Web services provision is supported by automated Web service composition methodology. Several methods for dynamic composition of Web services have been proposed in recent years.

Despite the increased academic and commercial interest to Web services, there is currently no case study analysing the applicability of automated Web service composition in commercial applications. Neither is there any publication about methodologies for identifying most relevant and potent Web services. Moreover, according to authors' knowledge, there is no publicly available study analysing the structure and potential synergy between commercial and governmental Web services.

We cover this shortcoming by applying automated composition for analysing the applicability of currently available Web services. We define the applicability of a Web service as the property to have the Web service in composite Web services. We propose a method for measuring the applicability of Web services with respect to existing Web services. Additionally, we propose a methodology for identifying most applicable Web services and demonstrate it on a case study. We also analyse interaction and potential synergy between commercial and governmental Web services.

1.5 Contributions

The main contribution of the thesis is a study of applicability of automated Web service composition. More specific contributions, which paved the way to the main contribution include both theoretical and empirical research. These contributions of the thesis are the following.

1. The thesis formalises PD for LL. Moreover, we prove the soundness and completeness of the formalism. The application of PD in explained through a number of examples.

2. The thesis formalises symbolic negotiation using PD. Additionally symbolic negotiation is positioned according to cooperative problem solving and similarities between these two ideologies are identified. It turns out that symbolic negotiation is a special case of cooperative problem solving.

3. The thesis proposes a multi-agent system architecture. The system is implemented using JADE and evaluated in an application for distributed automated Web service composition.

4. A distributed Web service composition tool is described and implemented. The tool demonstrates our vision of automated Web service composition.

5. Applicability of automated Web service composition is evaluated based on currently available commercial and governmental Web services. The analysis determines also the data, which is most common in inputs and outputs of today's Web services. The results are supported by a methodology for systematic analysis of Web services domains.

1.6 Outline

This thesis is divided into 4 parts. These parts are further divided into 12 chapters and 2 appendices. Part I presents the formal foundation of our automated computation framework. Part II describes the proposed multi-agent system, extends it with P2P capabilities and explains how it is going to be applied for distributed automated Web service composition. Part III describes the implemented tool and evaluates applicability of automated Web service composition. Finally, Part IV underlines the main results, concludes the thesis and discusses future work.

The general structure of the thesis is depicted in Figure 1.1. The figure shows relations between different topics covered in this thesis and indicates in which parts they are covered. In order to understand the upper-level topics in the figure at least partial understanding of lower-level topics is required.

Results in this thesis have been partially presented already in the following publications:

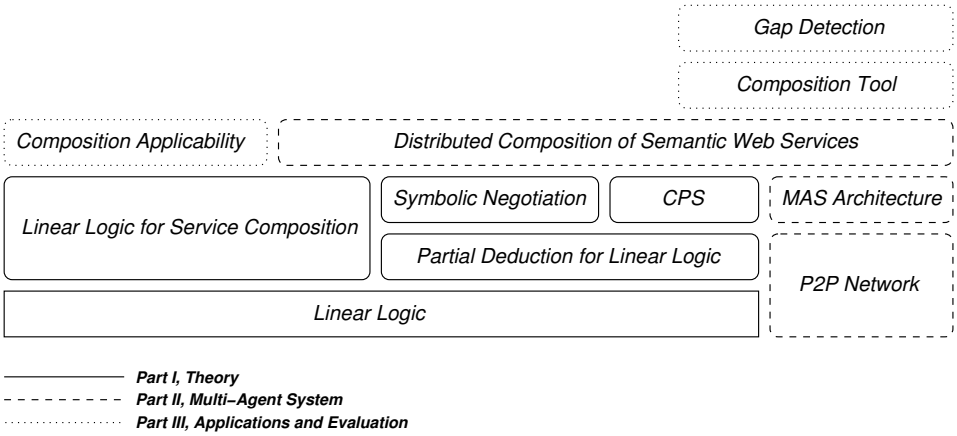


Figure 1.1: Research topic roadmap.

1. PD and symbolic negotiation have been formalised in [96, 97, 102, 103],
2. LL theorem proving for automated Web service composition has been explored in [159–161],
3. symbolic negotiation for distributed automated Web service composition has been clarified in [93, 106],
4. ideas about PD for gap detection have been explored in [92, 99–101],
5. symbolic and non-symbolic negotiation for composition have been combined in [98],
6. a larger picture of agent-based Web service composition has been compiled in [129],
7. P2P indexing for a MAS and distributed Web service composition has been analysed in [95],
8. Web services and data roadmap has been presented in [105],
9. abstraction as a PD strategy has been analysed in [91],
10. JXTA message transportation protocol implementation for JADE has been published in [119],
11. first steps to formalise temporal extension of symbolic negotiation have been taken in [94],
12. the methodology for evaluating automated Web service composition has been presented in [104].

Chapter 2

State of the Art

In this chapter we review the literature related to our technology. The related areas include linear logic, agent systems, P2P networks, the Semantic Web, automated Web service composition, etc. Many of the reviewed articles cross multiple areas.

For a quite general overview of basics, architectures and languages of agents and agent systems the reader is forwarded to [198]. Some agent-based applications are broadly reviewed in [78].

Considering that LL has been applied for Artificial Intelligence (AI) planning, and AI planning is employed in agent systems, we review also connections between AI planning and agent technologies. We put special emphasis to such agent activities as negotiation, team formation and coordination, since they are strongly related to the foundations of our work.

2.1 Related formalisms

Although PD was first introduced by Komorowski [89], Lloyd and Shepherdson [120] were the first ones to formalise PD for normal logic programs. They showed PD's correctness with respect to Clark's program completion semantics. Since then several formalisations of PD for different logic formalisms have been developed. Lehmann and Leuschel [111] developed a PD method capable of solving planning problems in the fluent calculus. A Petri net reachability checking algorithm is used there for proving completeness of the PD method. However, they do not consider how to handle partial plans.

Analogically Leuschel and Lehmann [113] applied PD of logic programs for solving Petri net coverability problems while Petri nets are encoded as logic programs. De Schreye et al [42] presented experiments related to the preceding mechanisms by Lehmann and Leuschel, which support evaluation of certain PD control strategies.

Matskin and Komorowski [128] applied PD to automated software synthesis. One of their motivations was debugging of declarative software specification. The idea of using PD for debugging is quite similar to our application of PD for symbolic agent

negotiation. In both cases PD helps to determine computability statements, which cannot be solved by a system.

Tammet [181] proposes a set of theorem proving strategies for speeding up LL theorem proving. He also presents experimental results, which indicate a good performance of the proposed strategies. Some of his strategies remind the usage of our PD inference figures. Thus some LL theorem proving strategies are already implicitly handled in our PD framework.

There are some similarities between abduction and PD. However, while abduction is about finding a hypothesis to explain given results, then PD achieves the hypothesis as a side-effect. The latter could be explained by stating that in our case the given results are a part of a program and PD is about program transformation, not about finding an hypothesis. By taking into account the preceding, abduction could be implemented through PD. Given the simplification that induction is abduction together with justification, PD relates to induction as well. An overview of inductive logic programming (ILP) s given by Muggleton and de Raedt [139].

Forward and backward chaining for linear logic have been considered by Harland et al [64] in the logic programming context. In this article we define backward and forward chaining in PD context. Indeed, the main difference between our work and the work by Harland et al could be characterised with a different formalism for different purposes.

Some researchers have enriched LL with modalities from modal logics. Kanovich et al [83] introduced time to LL on the first order level. However, the encoding is not flexible enough to handle several problems in agent systems. Hirai [72] proposes a framework, which is based on timed Petri nets and embodies both LL and modal logic S4 for time modalities. Thus both, formulae in S4 and LL are provable in this logic. Its main contribution compared to the work of Kanovich and Ito [81] is that full intuitionistic LL is considered, instead of the fragment of LL without modality !. The latter has the important feature from negotiation point of view—with ! also unbounded access to resource could be offered. Another approach for describing temporal LL through timed Petri nets is given in [182]. However, it lacks completeness theorem for timed Petri nets as stated in [72].

In [18] LL has been enriched with modalities for distribution and mobility. There also locations and movements of resources are considered and it is speculated to have applications in distributed systems and security protocols. This approach has connections to mobile ambients calculus [33] (MAC) and spatial logics.

MAC is an extension of π-calculus. While in π-calculus movements of processes can be described, in MAC it is possible to characterise also contexts (called ambients) of processes and their movements. Ambients can become parts of other ambients, if it is decided that they should move to an (another) ambient. Processes are described there as agents. This formalism is suitable for describing mobile agents (mobile code) in mobile environments (ambients). Security issues may be specified as well.

2.2 Linear logic and planning

Several authors [29, 38, 60, 76, 84, 126] have emphasised the contribution LL could have for AI planning. LL enriches problem domains with resource consciousness and allows to represent nondeterminism emerging from interaction with an environment. There exist [72, 81] even work extending LL with certain features of temporal logic. Therefore LL provides a comprehensive formalism for expressing most of the rigorous aspects of planning problems—resources, time, nondeterminism, etc.

Usage of LL, similarly to other logics, provides sound and complete AI planning mechanism. Soundness means that plans found by applying LL inference rules are correct with respect to initial problem specifications. Completeness guarantees that if there is a solution, it will be found. Moreover, by fixing the fragment of LL for planning, we determine computational complexity of planning as well, since complexities of different LL fragments have been already studied [82, 84, 117].

One merit of LL deductive planning is said [60] to consist in its ability to solve the technical frame problem [131] without the need to state frame axioms explicitly and provides thus a natural way for representing causal relations between actions and resources.

While considering AI planning within LL, one of intriguing issues is how to represent planning domains and problems. We take advantage of a resource-oriented representation of STRIPS-like operators as adopted by [17] for Transition Logic and [60, 76, 84, 126] for LL framework, where planning operators are encoded as extra-logical axioms and planning problem as a sequent which correctness has to be proved.

The first of LL planners was presented in [126], where a demonstration of robot planning system has been given. Influenced by [126], LL theorem proving has been used by Jacopin [76] as an AI planning kernel for STRIPS-like problem presentations. In [38] a formalism has been proposed for deductively generating recursive plans in LL. This advancement is a step further to more general plans, which are capable of solving instead of a single problem a class of problems. Another approach to deductive planning in a resource-conscious logic (Transition Logic) is given in [17]. The list of other deductive planners includes [16, 19, 59, 63].

Since it has been proven [46] that the Horn fragment of LL [82] can be presented with Petri nets, we list in the following some Petri net planners as well. There are some domain-independent planners, which use Petri net representation and corresponding analysing methods in a certain stage of planning. TokenPlan [50] for instance utilises colored Petri nets. First a PDDL description is translated to a colored Petri net, so that every place in the net represents a predicate from problem description and every transition corresponds to an planning operator. Then the flow of tokens inside this net is analysed and based on that information another graph is generated. This graph is then analysed in a way Graphplan does.

In Petriplan [173] a marriage of Graphplan [20] and Petri net submarking reachability problem solving is proposed. The Graphplan is used there for generating a plan graph for an initial AI planning problem and thus for reducing the search space. Then

the plan graph is converted into an acyclic Petri net representation and thus Petri net reachability or coverability checking tools become available for plan construction.

In manufacturing automation Petri nets have been used for assembly, disassembly and task sequence planning. Anyway Petri nets used there have specific properties and do not support domain-independent planning. An overview of Petri net usage in manufacturing automation is given in [138]. In [141] general Predicate/Transition nets were used for AI planning. However, while using Place/Transition nets, like in PNPlanner, we are able to recognise and reuse subplans. A survey of Petri nets is [140], where an overview of basic concepts and extensions and subclasses of Petri nets may be found. Propositional planning with Petri nets or within our LL fragment may be viewed as multiset rewriting. Rewriting has been proposed for AI planning in [192].

State-of-the-art domain-independent planners include Graphplan [20], SATPLAN [86], Blackbox [87] and their derivates. SATPLAN and Blackbox use propositional SAT encodings for representing planning problems and therefore fast algorithms for satisfiability testing can be applied to solve AI planning problems. Further advancements include HSP, HSP-R [23] and FF [73]. In [45] it has been shown that compiling the Graphplan planning graph into CSP yields additional advancements.

FF, HSP and HSP-R are based on "planning as heuristic search" paradigm, where heuristic information is derived from the specification of the planning instance and used for guiding the search through the search space. Both time and consumable/renewable resources have been embedded [68] to heuristic planning to combine expressiveness and performance.

2.3 Web services and intelligent agents

Gibbins et al [55] are probably the first ones who demonstrated, through an implementation, the usage of DAML-S Web service descriptions within agents. They separate the intentional meaning of messages (whether it is a request or an assertion, for instance) from their application domain specific content within DAML-S. That conforms with multi-agent systems (MASs), where intent of messages is given through an agent communication language (ACL) and domain specific content is given by means of domain-specific ontologies. That approach reduces the brittleness of a system. Their agents embody DAML-S descriptions of services and agent communication is managed through FIPA ACL.

Another step towards incorporating Web services into agents is proposed by Ardissono et al [8]. Their main contribution is support for more rigorous service execution protocols compared to currently prevalent 2-step protocols. These current protocols mainly consider sending input data and then collecting results from services. However, if we consider MASs, we have to support also other aspects than simple remote execution. One example is agent negotiation protocols, which have more refined structure than the simple remote procedure call supported by WSDL, for instance. In the paper also several shortcomings of WSDL, like inability to describe interaction protocols and

sequences of executions in complex services, are identified. To overcome these difficulties it is proposed that the service provider guides the consumer by specifying at each step of the interaction the set of eligible turn customers may perform. This could be done, in limited extent, through WSFL-like interaction flow specification.

We go beyond that approach by allowing a server agent to compose a sequence of action for a service consumer such that a required sequence of service command executions is constructed automatically at server side and the consumer can provide all details once. If that is not possible or does not suit for the consumer, the solution at server side could be partial as well, or the server may engage other servers as well, given consumer requirements. That would lead us to automatic service composition, which can be easily automated, if enough semantic information about user requirements and accessible Web services is available.

Dickinson and Wooldridge [44] are strongly motivated in writing agent applications for the Semantic Web. They adopt BDI approach as an underlying mechanism for agents. Message passing and agent directory are provided by JADE agent platform. Messages are encoded in FIPA ACL and agents' internal states are presented in AgentSpeak(L) [157]. However, authors do not consider non-agent standards for knowledge representation for accessing non-agentised services.

A comparison of 2 currently prevalent agent communication languages, KQML and FIPA ACL, is presented in [108]. Although they both have different origins, they both are almost identical with respect to their basic concepts. The main difference lies in their underlying semantic frameworks.

All the mentioned approaches are supplement to each-other and cover different aspects of agent-based Web services. We are trying bring these efforts together under a framework, which takes advantages of both agents and the Semantic Web languages languages. Agent specific aspects provide Web services with proactivity, reactivity, social ability and autonomy, while the usage of the Semantic Web services languages, FIPA ACL and application domain specific ontologies provide a standardised medium for Web service deployment. Usage of DAML-S allows publishing semantically enriched specifications of Web services and thus fits well to the Semantic Web vision. Agents together with semantic description of Web services allow hiding complex interactions of Web service invocation. A nice introduction to usage of ontologies and Web services in multi-agent systems is given in [69].

2.4 Automated Web service composition

Web services' research in the Semantic Web context varies from automated annotation and ontologies [9] to automated Web service composition. While automated annotation intends to provide methods to extract semantics from existing Web services, research related to ontologies and Web services has focused to the modelling of Web services. The latter has led to such initiatives as WSMO, SWSO, OWL-S and WSDL-S. Some existing Web services standards and related technologies are reviewed in [187].

Several methods for dynamic composition of Web services have been proposed in recent years. Most of them fall into one of the following two categories: methods based on pre-defined workflow model and methods based on AI planning.

For the methods in the first category, the user should specify the workflow of the required composite service, including both nodes and the control flow and the data flow between the nodes. The nodes are regarded as abstract services that contain search recipes. The concrete services are selected and bound at runtime according to the search recipes. This approach is widely adopted by members of the Information Systems community (in particular, see [34] and [169]).

The second category includes methods related to AI planning and automated theorem proving. They are based on the assumption that each Web service is an action which alters the state of the world as a result of its execution. Since Web services (actions) are software components, the input and the output parameters of Web services act as preconditions and effects in the planning context. After a user has specified inputs and outputs required by the composite service, a process (plan) is generated automatically by AI planners without the knowledge of predefined workflows.

A strong interest to automated Web service composition from AI planning community could be explained roughly by similarity between DAML-S and PDDL [133] representations. PDDL is widely recognised as a standardised 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 automated Web service composition is required, DAML-S descriptions can be translated to PDDL without any user intervention. Then different planners could be exploited for further Web service composition.

In [134] a modification of Golog [115] programming language is used for automatic construction of Web services. Golog is built on top of situation calculus and has been enriched with some extra-logical constructions like **if**, **while**, etc. Golog also provides an efficient way to handle equivalence relations. Therefore, it is argued that Golog provides a natural formalism for automatically composing services on the Semantic Web.

However, one disadvantage of Golog origins from its logical foundations. Since Golog is based on situation calculus, frame axioms have to be specified, or some higher-order constructions have to used as has been proposed in [115]. Frame axioms specify fluents, which remain unaffected, if an action is applied in situation calculus. Anyway, construction of frame axioms is a comprehensive task for Web services developers.

Although non-monotonicity of situation calculus allows Golog to model state changes in a world, theorem proving with situation calculus is still based on truth values of literals. Therefore it is not possible in Golog to present and reason about multiple copies of literals in world states. Due to the latter problem some services may not be describable within Golog, or extra efforts and logic programming tricks are required.

Waldinger [194] proposes initial ideas for another deductive approach. The ap-

proach is based on automated deduction and program synthesis and has its roots in the work presented in [123]. First available services and user requirements are described with a first-order language, related to classical logic, and then constructive proofs are generated with SNARK [175] theorem prover.

Although some authors [109] have considered service synthesis with propositional logics, first-order logics provide richer semantics. Additionally, the usage of first-order logics allows to shrink the size of ontologies used in Web service composition. The latter in turn affects directly the performance of discovery and exploitation of available Web services.

Lämmermann [109] takes advantage of disjunctions in classical logic to describe exceptions, which could be thrown during service invocations. We take advantage of disjunctions for more general purpose—to represent exclusive outputs of services, which could be exceptions as well as other results provided by services.

Hashemian and Mavaddat [67] combine breadth-first graph search and interface automata [41] for automating Web service composition. While graph search is used for finding a path with minimum length from identified input nodes to identified output nodes, interface automata is applied for composing paths into a composite Web services. Graph search operates over a directed graph, where edges represent available Web services and nodes represent inputs/outputs of particular Web services.

Mao et al [124] apply Dijkstra's shortest path search algorithm for first composing a service and then distributing the constructed workflow over a physical network. The path search operates over graphs, which represent sets of available services and physical network nodes. Although Mao et al [124] do not consider semantics and currently prevalent Web services standards, the paper is one of the earliest contributions focusing directly to Web service composition in a general sense.

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 false. In logic programming this approach is called *negation as failure*. The main trouble with the closed world assumption, from Web services perspectives, is that purely with truth literals we cannot express that new information has been acquired. For instance, one might describe that after sending a message to another agent, an identity number to the message will be generated. Thus during later communication the identity number could be used.

McDermott [132] considers this problem in AI planning in composing Web services. He introduces a new type of knowledge, called *value of an action*, which persists and which is not treated as a truth literal. However, 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. Since LL is not based on truth values, we can view generated literals as references to informative objects. Thus, if a new literal is inserted into the world model, new piece of information will be available. Therefore, LL provides an elegant framework for modeling incomplete knowledge—although before plan execution only partial knowledge is available, during execution more details would be revealed.

Sirin et al [174] propose a semiautomatic Web service composition scheme for interactively composing new Semantic Web services. Each time a user selects a new Web service, all Web services, that can be attached to inputs and outputs of the selected service, are presented to the user. In this way a lot of manual search is avoided. Anyway, the process could be fully automated by applying our methodology and if user requirements to the resulting service are known *a priori*.

Paolucci et al [145] evaluate a broker for constructing OWL-S Web services. They also identify some drawbacks of the current OWL-S specification and propose a workaround for the problem. Advantages of applying the broker architecture, like anonymisation, trusted intermediary and communication facilitation, are emphasised there.

Sycara et al [178] describe a methodology for constructing composite Web services written in DAML-S. Also DAML-S semantics is reviewed there (a formalisation). Additionally the implemented DAML-S virtual machine is evaluated there. Still, the two preceding articles consider service composition as matching suitable atomic components. Similarly, the preceding articles view service advertisements as service templates, which describe the inputs/outputs and preconditions/effects of particular services. The concept of advertisements could be significantly extended with domain-specific knowledge.

SWORD [154] is a developer toolkit for building composite Web services. 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 inputs and the outputs of Web services. As a result, reasoning is based on the entity and attribute information provided by an ER model.

Thakkar et al. [184] consider dynamic composition of web services using mediator architecture. The mediator takes care of user queries, generates wrappers around information services and constructs a service integration plan.

Triana [121] is another graphical toolkit for Web service composition and execution. Atomic Web services are either discovered through a UDDI repository or are exported from WSDL files. New Web services are composed manually. The main advantage of the framework, over other similar tools, is its ability to facilitate distribution of composed workflows and their execution across a P2P or a Grid network. A framework for automating Web service composition in Triana and possibly in other systems has been proposed by Majithia et al [122]. A simple backward-chaining algorithm is used for composing workflows, while matchmaking instantiates these workflows.

Hull and Su [75] present a short overview of tools and models for Web service composition. The models include OWL-S, the *Roman* model [14] and the *Mealy machine* [30]. While OWL-S includes a rich model of atomic services and how they interact with an abstraction of the "real world", the *Roman* model and the *Mealy machine* use a finite state automata framework for representing process flows.

Gómez-Pérez et al [57] describe another interesting tool for Semantic Web service composition. The resulting service can be exported to an OWL-S specification. In [200] SHOP2 planner is applied for automatic composition of DAML-S services. Other planners for automatic Web service construction include [152, 153, 172, 186].

The list is constantly growing.

Semantic matchmaking could be seen as a form of Web service composition. Paolucci and Sycara [146] propose that UDDI tModels could be exploited to encapsulate DAML-S descriptions within UDDI repositories. That would provide us semantic matchmaking, while still preserving usage of UDDI repositories.

2.5 Automated Web service annotation

It has to be mentioned that the full power of automated composition cannot be harnessed before Web services are annotated semantically. Hence, given the huge number of Web services available, there is a need for automated annotation methods. Several aspects of automated Web service annotation have been considered by research groups. Patil et al [150] present METEOR-S Web service annotation framework. The framework implements new constructs for embedding semantic annotations into existing industry standards. Four kinds of semantics is considered: data, functional, execution and QoS semantics. This contrasts with our approach where we consider just data semantics and embed functional semantics into data semantics. For mapping Web service data types to each-other, initially corresponding XML Schemas are transformed into *SchemaGraphs*. Then linguistic and structural similarity is computed to evaluate the best mapping between existing ontologies and elements in SchemaGraps. Similarity is measured statistically.

Sabou et al [164] present a case study of using DAML-S ontology for annotating semantically Web services. They identify difficulties of writing DAML-S services. Since programmers are assumed to have knowledge about WSDL, SOAP and DAML, the language is too comprehensive and knowledge-demanding for human users. This result inherently applies also to successors of DAML-S (OWL-S) and other XML-based languages. Thus the degree of automation should be increased to reduce the complexity of annotation.

Sabou [163] proposes a semi-automatic method for extracting semantics from software API documentations. The intuition is that, if particular API implements a Web service, then the semantics of API corresponds to the semantics of the Web service. Heß et al [70, 71] employ the Naive Bayes and SVM machine learning algorithms to classify WSDL documents according to predefined semantic taxonomies. They allow classification of Web services, their domains and data types. Burstein [31] is concerned with construction of ontology mappings between terms in different Semantic Web services. It is argued that since Web service providers do not use a shared ontology for describing semantically their Web services, automated ontology mapping is required.

2.6 Automated software synthesis

Automatic software construction reduces the amount of time and other resources spent for software coding. While identifying possibly reusable fragments of software, logical

approaches may be used for software composition. Thereby software coding is reduced to software specification. Since the resulting software is generated automatically using semantically correct rules, the resulting software is error-free with respect to its formal specification.

Simultaneously with software synthesis there is another field, which is concerned with software correctness—software verification. In contrast to software synthesis, software verification is concerned with looking for inconsistency in already generated programs. Since programming and programs may be viewed as theorem proving and theorem proof, respectively, automatic software construction seems more natural than verification, because the code generated is already correct by means of the initial specification.

Through declarative description of programs, introspection property is provided for programs. The declarative programming way also makes a written (specification) code simpler to modify and understand, thereby reducing the risk of introducing software bugs. It is emphasised [21] that while directly manipulating with concepts of an application, programs become more understandable. One role of declarative programming is thus to bridge the gap between mental concepts and program symbols.

Structural Synthesis of Programs [188] (SSP) is concerned with generating new software from predefined methods. Every method is annotated with a logical axiom $A \vdash B$, where A and B are sets of literals for denoting input and output variables of a method, respectively. Given a sequent defining initialised variables (literals in the antecedent) and variables to be computed (literals in the succedent), theorem proving within propositional intuitionistic logic is employed for software synthesis. This method has been applied in NUT [189] programming environment.

For efficient proof search [130] forward search is performed to generate an initial proof. This proof possibly includes redundant steps, because any applicable axiom may be included to the proof. The redundant inference steps are removed by backward pass, where a set W of needed variables is used for checking appropriateness of every axiom. Initially W includes only variables of the goal. When an axiom is classified to be useful and kept in the proof, its input variables are included into W and its output variables are excluded from W.

Partial deduction has been discussed for SSP in [128]. The main idea is to specify the goal only partially or not at all. This contrasts with widely adapted goal-directed proof search and computation in logic programming. The main reason for partial deduction is to ease up specifying goals—users only specify most import results of computation they are interested in. Or they do not specify anything at all, but later inspect whether anything interesting was produced. With partial deduction a theorem prover generates all possible results for a given domain.

Manna and Waldinger [123] propose a first-order theorem proving methodology for generating programs. Theorem proving is based on resolution rules and substitution. One advantages of this framework is that mathematical induction to resolution framework is introduced—induction is handled as a deduction rule. For more efficient proof search, (sub)formulae are polarised to indicate when certain inference rules may

be applied to them.

Software specifications are given with first-order input-output relations. There is a need to define assertions, goals and how the output of a function to be synthesised depends on inputs. While assertions (preconditions) are in forms A and $A\&B$, goals are constrained to forms A, $A \vee B$ and $A \rightarrow B$, where A and B are first-order formulae.

As a result of theorem proving a code for implementing a function is generated. By theorem proving termination of recursive calls is ensured. Additionally construction of subfunctions is considered.

Darlington [39] proposes a first-order recursion equation language for software synthesis based on transformation. Initial functions are described in the recursive way. For instance, one may write $fact(0) \Leftarrow 1$ and $fact(n+1) \Leftarrow (n+1) * fact(n)$ to define the traditional factorial function.

The underlying transformational system enables to convert normal recursive forms of a program to iterative forms. The system is intended for making programs more efficient by means of program transformations. Program transformation is performed through interaction with a user. The user is required to provide instantiations of left hand side base cases and recursive branches of a program during interaction. The system then proposes new transformed program clauses and the user either accepts or rejects them.

Also subfunctions can be generated with the help of a user. For instance, if a proof could be constructed, given that a new function can be defined, then the user is asked to provide the implementation of that function and proof search continues. Thereby the system can decompose programs or functions into smaller pieces. Thus programming is redefined as interaction between a human and a machine. The methodology allows also generating inverse functions. For example, if a function is defined as $f(x) = 2 * x$, then its inverse function is $f^{-1}(y) = y/2$.

However, it seems that, if software description gets large, users are overwhelmed with answering to rigorous questions. Anyway, the approach may be one of the first steps toward a new possible programming paradigm—interactive programming.

Another approach, based also on transformation rules, is proposed by Broy and Pepper [28]. There every transformation rule is determined by preconditions (P), which should hold before a transformation is applied. Also a matched clause (A) and its replacement (B) has to be given. Thus transformation $A \Rightarrow B$ can be applied, if P holds. Thereby previously applied transformations can bias the following ones. The resulting program is correct by construction.

Likewise the approach of Darlington, the purpose of this approach is program optimisation (and partial implementation), since given rules represent either optimisation techniques or particular implementations. The transformational system can be seen as macro-processing in imperative languages like C.

Summers [177] considers automatic LISP program construction from examples. Examples are represented with input-output pairs of lists. Recursive LISP programs are synthesised from these examples consisting of primitive LISP functions *car*, *cdr*, *cons* and *atom*. First all possible subexpressions of examples' inputs are enumerated and

Table 2.1: Comparison of automated software synthesis approaches.

Method	Language	Purpose	Techniques	Problem representation	Automatic
SSP [130]	intuitionistic propositional logic	structural software compositions	theorem proving	axioms define procedures	yes
Summers [177]	LISP (car, cdr, cons, atom)	induce a program	generalisation	input-output pairs	yes
Darlington [39]	FO recursion equation language	optimisation	transformation	recursive definition of functions	inter– active
Manna and Waldinger [123]	FO language	low – level program construction	deduction math. induction	assertions, goal, function output	yes
Broy and Pepper [28]	FO language	optimisation partial implementation	transformation	original code transformation rules	yes
ILP	Horn clauses	induction of predicates and clause definitions	generalisation specialisation	positive and negative examples of predicates	yes

combined to form outputs. Then inputs are transformed to a normal form representing only the *structure* of inputs. Generated structures in turn allow to order input examples by their structural complexity and inspect how an hypothetical program behaves on input data. Finally generalisation is applied and programs are induced.

The approach is a step towards inductive logic programming [110] (ILP). ILP is a research area at the intersection of machine learning and logic programming. We can classify ILP robustly into empirical and interactive. While empirical ILP is concerned with learning a single target relation from a large collection of possibly noisy examples, interactive ILP systems attempt to learn multiple relations, which may be interdependent, from a small set of correct examples. Learning in the latter subdiscipline is incremental.

Basic ILP techniques involve generalisation and specialisation [110, p. 39]. Given a possibly empty set of negative and a nonempty set of positive examples, through generalisation more general logic programming language expressions are derived as long the more general clauses do not cover any negative example. Specialisation works in the reverse order—given a set of general clauses covering also negative examples, the general clauses are specialised until they do not cover any of the negative examples.

Linear logic (LL) has been used for prototyping multi-agent systems [24]. Because of fixed semantics of LL, formal method tools can be used for checking whether the system functions just as intended. Although the prototype LL program is executable, it is still too abstract to produce a final agent-based software. Thus another language is used for building final software.

The reviewed approaches to automatic software synthesis or optimisation are summarised in Table 2.1.

2.7 P2P-based Web service composition

Verma et al [193] consider a P2P infrastructure for publishing and discovering semantically enriched descriptions of Web services. Anyway, they still use UDDI mechanism for publishing Web services, whereas UDDI structures are used for storing se-

mantic information about inputs and outputs of Web services similarly to Paolucci et al [144]. Our approach allows to bypass usage of centralised services for service discovery though we do not neglect their possible usage, if they could provide semantic content as well. In their paper also current state-of-the-art of service advertisement, discovery, invocation and orchestration are reviewed. Some of these standards have definitely place in our architecture as well.

Arpinar et al [9] apply similarly to us automated Web service composition over a P2P network. In their P2P architecture peers are organised into communities such that each community involves peers, which represent the same domain. DAML-S is used for describing Web services and queries (composite Web service interfaces). The major difference between our ideology and the one presented by Arpinar et al is that they try to determine links between Web services at publishing time, while we do it at composition time. While our approach is more flexible and suits better to highly dynamic networks with a moderate amount of queries, their approach is definitely more suitable for more stable networks with massive amounts of queries. Anyway, their method does not consider the non-monotonicity of Web services, which is handled by our methodology.

Paolucci et al [147] implement a P2P service discovery mechanism through the usage of Gnutella P2P network. Discovery process is based on reasoning over DAML-S descriptions of Web services. Java Expert System Shell (JESS) is applied as a DAML-S inference mechanism and is engaged to determine whether a service satisfies a query. The approach is suitable in cases where atomic Semantic Web services are known *a priori* and semantically equivalent or similar services have to be discovered. Anyway, during automated composition it is not known, which atomic services would be included in the a resulting composite service. Thus their approach is not particularly suitable for automated composition as we consider it here.

Similarly to us have Ermolayev et al [47] proposed cooperative agent-based Semantic Web service composition. They exploit OntoServ.NET environment as a P2P network. However, in this paper service composition is handled as an instantiation of an existing workflow, while our approach both constructs a workflow and instantiates it as well.

Benatallah et al [13] consider P2P provision of Web services. However, their service composition process is static and the main contribution is a study of distributed, decentralised service execution. Some essential issues in decentralised Web service provision have been addressed by Papazoglou et al [148].

2.8 Semantics and P2P networks

Broekstra et al [27] consider semantic-based P2P systems. Their main focus is placed on knowledge representation and management in query processing in P2P networks. Due to heterogeneous nature of knowledge in P2P networks certain conventions have to be introduced to semantic reasoning process.

Crespo and Garcia-Molina [37] consider the construction of semantic overlay networks for P2P systems. Their contribution is a method for automatic clustering of P2P networks to semantic overlay networks according to agent properties given by the semantics of their content. A peer may belong to several overlay networks if it encapsulates data with different semantics. Such a clustering allows query routing according to its content. Since the message is sent directly to affected parties the number of messages for resource location is significantly decreased.

Schmidt and Parashar [168] propose a P2P indexing mechanism and associated P2P storage, which supports large-scale real-time search capabilities. The novelty of the system is that it supports location of data sources identified by a query consisting of partial keywords and wildcards. The system is largely based on the Chord indexing schema with an extension that queries can be more complex. Hilbert space filling curves (HSFC) are applied for mapping an n-dimensional keyword space to 1-dimensional hash value space. HSFCs are *locality preserving* are characterised by *digital causality* property. While the former means that points, which are close in 1-dim space, are close in n-dim space as well, the latter implies that indices in the same index space sub-cube have the same prefix for keywords. Related to the previous source, Andrzejak and Xu [5] use the same function (Hilbert SFC), but in a reverse manner (unlike Schmidt and Parashar [168]), to map resources to peers.

Tang et al [183] consider semantics in P2P systems. They adopt Latent Semantic Indexing (LSI) for information retrieval in Content-Addressable Networks (CAN). The semantics of a document is described with a set of keywords. Instead of LSI some other information retrieval algorithm [155] could be applied as well for computing a unique value to a particular set of keys. Another way to discover semantics in distributed systems is proposed by Guha [62], who concerned with semantic negotiation for determining the meaning of concepts according to shared keys, which describe the concepts.

Bawa et al [11] propose a P2P network topology, where the network is clustered into segments by topics. In this case short distance links connect peers sharing the same topic, while long distance links connect peers at different segments. For each topic a centroid is constructed, which represents a centerpoint for a topic. In our network a centroid is represented with a literal. Thus each literal in the system represents a centroid. Peers may be connected to several centroids simultaneously. Thus although we apply Chord [176] ideology and protocols for managing our P2P network, we have a subnetwork for each literal. Each peer may have several identification codes—one for each subnetwork.

Adjiman et al [2] implemented a P2P network for distributed theorem proving over propositional classical logic clauses. The underlying network is based on *small world* topology [74], where each peer has a list of other peers, who share the same literals (parts of a theory) as the agent does. Small world paradigm as an extension to structured P2P networks has been proven [74] to provide efficient access to popular objects if the network content is highly clustered. In our case we are interested in finding a literal carrier from the network and this makes DHT-s more suitable for us. We still

apply some techniques from small world ideology.

Peer-Serv [195] is a framework for exploiting Web services in a P2P environment. Peer-Serv exploits a Web service broker federation, while each peer is assigned to a particular broker. P2P mechanism is used for Web service execution, publishing and querying. The proposed architecture could be seen as a federation of UDDI registries, where queries are handled through a P2P medium. Another mechanism for Web service discovery through structured P2P networks is proposed by Kaffille et al [79].

Another mechanism for resource discovery in structured P2P networks is discussed by Antonopoulos et al [6, 7]. Their P2P network structure is based on Chord and organises indexes in multiple Chord rings. A more thorough analysis of major P2P approaches has been published by Milojicic et al [137].

2.9 Multi-agent systems

Since the literature for agent systems is abundant we review here only these formalisms, which mostly relate to our work and have inspired it in the beginning.

2.9.1 Negotiation

As it has been indicated in [77] negotiation is the most fundamental and powerful mechanism for managing inter-agent dependencies at run-time. Negotiation may be required both for self-interested and cooperative agents. It allows to reach a mutually acceptable agreement on some matter by a group of agents.

Rahwan et al [156] review existing argumentation-based approaches to negotiation. They review issues like challenges, negotiation protocols plus mechanisms for argument construction, selection and evaluation.

Davis and Smith [40] seem to be the first to point out the usage of negotiation as a metaphor for distributed problem solving. They see negotiation as a basis for a protocol for organising problem solving activity. Negotiation is applied in matching problem solvers and tasks whereas Contract Net [40] protocol is applied for negotiation. Anyway, it was PUP6 [112] system to suggest first that interaction between software system entities could be viewed as a *discussion* between interested parties.

Kraus et al [90] give a logical description for negotiation via argumentation for BDI agents. They classify arguments as threats and promises, which are identified as most common arguments in human negotiations. In our case only promises are considered, since in order to figure out possible threats to goals of particular agents, agents' beliefs, goals and capabilities should be known in advance to the persuader. We assume, that our agents do not explicitly communicate about their internal state. Thus, our agents can provide higher degree of privacy in agent applications compared to particular BDI agents.

Parsons et al [149] defined negotiation as interleaved formal reasoning and arguing. Arguments and counterarguments are derived using theorem proving while taking

into consideration agents' own goals. While Parsons et al [149] perform reasoning in classical logic, it is possible to infer missing clauses needed for achieving a goal. The situation gets more complicated, when several instances of formulae are available and, moreover, the actions performed by agents or resources they spend can be interdependent. Thereby, inference in LL is not so straightforward, since some clauses are "consumed" while inferring other clauses. Due to the aforementioned reasons we apply PD to determine missing parts of a proof. Then the missing part is announced to other possibly interested agents.

Fisher [51] introduced the idea of distributed theorem proving in classical logic as agent negotiation. In his approach all agents share the common view to the world and if a new clause is inferred, all agents would sense it. Inferred clauses are distributed among agents via broadcasting. Then, considering the received information, agents infer new clauses and broadcast them further again. Although agents have a common knowledge about inferred clauses, they may hold different sets of inference rules. Distribution of a collection of rules between agents means that different agents may have different capabilities and make different inferences. The latter implies that different agents contribute to different phases of proof search. Our approach differs from that work mainly in 2 aspects (in addition to usage of another logic): (1) our agents do not share a common view of a world and (2) inference results are not broadcasted.

Sadri et al [165] propose an abductive logic programming approach to automated negotiation, which is built on Amgoud et al [3] work on argumentation. The work of Sadri et al is more specialised and detailed than the work by Amgoud et al. That allows deeper analysis of the reasoning mechanism and the knowledge required to build negotiation dialogues.

2.9.2 Coalition formation and teamwork

One of the first formalisations of cooperative problem solving is given by Wooldridge and Jennings [199] (other approaches presented so far are also reviewed there). One of the earliest *implemented* general models of teamwork is described in [180], which is based on joint intentions theory and on shared plans theory.

An advancement to Bratman's theory of intention [25] is presented in [158], where relations between beliefs, goals and intentions are given and for the first time belief, goal and intention revision process is captured. The main idea is that an agent can have only those goals, which she believes to be achievable and only those intentions, which lead to goals.

This approach differs from that of Cohen and Levesque [35] in that it treats intentions as a basic attitude and shifts the emphasis of future commitment from the definition of intention to the process of intention revision. Semantically, this approach differs in that it distinguishes between the choice available to the agent in choosing her actions and her beliefs about which worlds are possible. In addition interrelationship between beliefs, goals and intentions is specified there.

The joint intentions theory [36] determines the means how agents should act to

fulfill joint goals, when they should exchange messages, synchronise between themselves, leave the team, etc. It also determines when the joint goal is considered to be achieved or when and how to break up commitment to it, if it should, for instance, turn out that one agent is not able anymore to perform its task(s).

Decision making about whether a goal has been achieved, is not achievable or there is no need to achieve it anymore, is based on consensus—every agent can initiate a discussion through which consensus is (presumably) achieved. Then everybody acts as stated by the consensus. However, it is not stated how joint goals are formed through negotiation or other processes.

2.9.3 Agent coordination

Our approach could be viewed as distributed planning similarly to the work in [52]. Case-based planning has been applied for coordinating agent teams in [54]. The planner generates a so called shared mental model of the team plan. Then all agents adapt their plans to the team plan. This work is influenced by the joint intentions [36, 114] and shared plans [61] theory.

In [179] agent coordination is performed through task agents by planning. First problem solving goals are raised, then solutions satisfying these goals are computed and finally these plans are decomposed and coordinated with appropriate task or other agents for plan execution, monitoring and result collection. Other agents are information and interface agents, for information collection and interfacing with a human user, respectively.

While task agents have a model of the task domain in advance, information agents are allowed additionally to seek for additional information during problem solving (anyway problem solving is not so sophisticated there as is in task agents).

According to [22] our theorem proving methodology is characterised with *parallelism at the search level*. The approach relates by theorem proving methodology mostly to the successors of Team-Work [53]. Fuchs [53] describes an approach, where distribution of facts and sub-problems is organised on request—that is the basic mechanism behind our methodology as well. However, Fuchs considers first-order logic with equality, which is somehow different from LL.

De Weerdt et al [43] applied a resource logic for multi-agent plan merging. The general idea is to optimise already existing plans of agents through cooperation. Given that each agent has a plan, which solves its task, the plans still may produce some extra resources, which are not required by agents themselves. Therefore, while exchanging the extra resources, agents may discard some steps in their plans and joint efficiency is achieved. In our case we consider online plan construction rather than merging complete individual plans.

Kakas et al [80] present a logical framework, which integrates planning, negotiation and control of operation. Computational logic is used for describing these processes. Although similar processes have been integrated for instance in the CPS model by Wooldridge, the biggest advantage of this framework arises from executability of

logical specifications. Governatori et al [58] apply defeasible logic for automated negotiation.

Van der Krogt et al [191] proposed a resource-based framework for planning and re-planning. Their formalism seems to encapsulate some similar aspects as our framework although in the AI planning domain.

Finally there are several works considering task allocation via agent coalition formation, for instance [167, 170, 171, 185]. Anyway, while these methods tend to focus on task allocation to particular agents and coalitions, we assume that each agent has a task already assigned to it. Thus our agents have to figure out whether and when to form coalitions by themselves. Moreover, while other methods are mostly about task decomposition, our method could be seen as composite task construction through coalition formation.

2.10 Linear logic in agent systems

LL contributions to agent technologies are based mainly on its resource consciousness and both internal and external nondeterministic choices, which cannot be expressed in classical logic. In classical and modal logic, for instance, all derived formulae last literally speaking forever, meaning that if an agent is spending money, it never runs out of it despite of the amount traded for goods or services. LL in contrary allows to determine when and how resources are consumed.

During the recent decade significant improvements in linear logic [56] (LL) and agent technologies have been made. Both disciplines have witnessed a shift from theory to applications. Although in agent technologies other logics like situation calculus, classical and modal logic (ML) have been applied for both, theoretical and practical purposes, LL has been out of interest focus so far. However, since LL provides several advantages, like resource consciousness and differentiation between disjunctions, over other logics explored in agent technologies by now, first attempts have been made to find usage for LL in agent systems.

In [24] LL has been used for prototyping multi-agent systems at conceptual level. Because of the fixed semantics of LL, it is possible to verify whether a system functions as intended at conceptual level. Although the prototype LL program is executable, it is still too high level to produce a final agent-based software. Thus another logic programming language is embedded to compose the final software.

Harland and Winikoff [66] address the question of how to integrate both proactive and reactive properties of agents into LL programming framework. They use forward chaining to model the reactive behaviour of an agent and backward chaining to model the proactive behaviour. This type of computation is called as mixed mode computation, since both forward and backward chaining are allowed. The theoretical background to mixed mode computation is given in [64].

Harland and Winikoff [65] also presented the first ideas of applying LL theorem proving for agent negotiation. The main advantages of LL over classical logic are its

resource-consciousness and existence of two kinds of nondeterminism. Both internal and external nondeterminism in negotiation rules can be represented. In the case of internal nondeterminism a choice is made by resource provider, whereas in the case of external nondeterminism a choice is made by resource consumer. For instance, formula $Dollar^5 \multimap Beer \oplus Soda$ (at the offer receiver side) means that an agent can provide either some *Beer* or *Soda* in return for 5 dollars, but the choice is made by the provider agent. The consumer agent has to be ready to obtain either a beer or a soda. The formula $Dollar \multimap Tobacco \& Lighter$ (again at the offer receiver side) in contrary means that the consumer may select which resource, either *Tobacco* or *Lighter*, s/he gets for a *Dollar*.

The main idea is to describe agents' abilities and expectations with LL formulae. For instance, formula $Apple \otimes Apple \multimap Paint \oplus Pencil$ means that an agent can provide you either with some *Paint* or a *Pencil* in return for two apples, but the choice is made by the agent. Thus the consumer agent has to be ready for both cases—it may obtain either paint or a pencil as well. The formula $Dollar \multimap Tobacco \& Lighter$ in contrary means that the consumer may select by itself which resource, *Tobacco* or *Lighter*, it gets for a *Dollar*. When initial resources owned by agents and expected negotiation result has been specified we can use LL theorem proving for negotiation. Generated proofs embed solutions for achieving defined goals.

In [196] a framework for embedding both declarative and procedural aspects of agents' goals is proposed. Declarative description of a goal defines the state of the world which is sought and procedural description gives a set of actions which are executed in order to achieve the goal. While declarative information allows detecting and solving goal conflicts, procedural information is needed to specify how the agent should achieve the goal. Such a goal presentation brings us closer to goal-oriented logic programming, because the declarative and the procedural part of a goal represent the head and the body of a logic program clause respectively. Such kind of representation reminds also ideas from reactive and partial order planning.

2.11 Summary

Although the work reviewed in this chapter covers a plethora of aspects related to automating several tasks in information systems, there is still a plenty of room for further research. For instance, in the field of automated Web service composition there is still no consensus on how to describe composition problems. Moreover, it is not even clear how expressive languages or formalisms are required for describing the problems. The similar issue applies to the Semantic Web in general—it is not clear which properties of objects constitute their semantics and therefore should be expressed. Since the requirements for a language are not known, it is also unclear whether and which existing (logical) formalisms are sufficient to describe the semantics of data objects and Web services.

There is also need for understanding the characteristics of search space of Web

service composition tasks with respect to AI planning tasks. It seems that while general AI planning problems generate a deep search tree with small branching factor, Web service composition generates a shallow search tree with huge branching factor. This in turn means that specialised heuristics are required for solving Web service composition problems as specialised AI planning problems.

Anyway, in order to apply automated Web service composition, Web services have to be annotated semantically first. However, due to the constantly increasing number of published WSDL documents, automated annotation methods are required. Although there are some efforts to automate Web service annotation, there is still no study, where automated annotation has been applied to a significant number of Web services. Neither seems to be there any initiative, which would allow to compare accuracy and other properties of proposed automated annotation methods. Hence a systematic evaluation of existing annotation methods would help to facilitate automated Web service composition.

Web service discovery through P2P networks requires extending existing P2P network topologies such that full services' structure could be mapped into particular P2P networks. Then efficient discovery methods, considering both structural and QoS properties of required Web services, can be designed. Moreover, it is desired that the new topology would allow to locate both data and services in the unified way.

Finally, there are many opportunities for contributions in the field of multi-agent systems. For example, game-theoretic negotiation has been extensively studied within the agent community. However, symbolic reasoning as negotiation is a methodology, which just recently has received focus. Although some approaches have been proposed for symbolic negotiation, the suitability of many more logics has yet to be evaluated. Moreover, limitations and advantages of symbolic negotiation have not been completely revealed.

Part I

Formal Foundations

Chapter 3

Partial Deduction

In this chapter we formalise Partial Deduction (PD) for intuitionistic fragment of Linear Logic [56] (ILL). ILL gives a resource-oriented basis for encoding non-monotonic computational problems. One instance of these computational problems is automated Web service composition. The proposed PD formalism is used to formalise CPS and symbolic negotiation in Chapter 4.

We encode PD steps as ILL inference figures, which allow us to simplify proof search in our fragment of LL. While using those inference figures instead of basic LL rules, we can achieve higher efficiency during proof search compared to generic theorem proving. Moreover, PD gives us a freedom to control automated theorem proving according to determined strategies.

3.1 Linear logic

LL is a refinement of classical logic introduced by J.-Y. Girard to provide means for keeping track of "resources". In LL two assumptions of a propositional constant A are distinguished from a single assumption of A. This does not apply in classical logic, since there the truth value of a fact does not depend on the number of copies of the fact. Indeed, LL is not about truth, it is about computation.

In the following we are considering intuitionistic fragment of LL (ILL) consisting of multiplicative conjunction ($\otimes$), additive disjunction ($\oplus$), additive conjunction ($\&$), linear implication ($\multimap$) and "of course" operator ($!$). In terms of resource acquisition the logical expression $A \otimes B \vdash C \otimes D$ means that resources C and D are obtainable only if both A and B are obtainable. After the sequent has been applied, A and B are consumed and C and D are produced.

The expression $A \vdash B \oplus C$ in contrast means that, if we have resource A, we can obtain either B or C, but we do not know which one of those. The expression $A \& B \vdash C$ on the other hand means that while having resources A and B we can choose, which one of them to trade for C. Therefore it is said that $\oplus$ and $\&$ represent respectively *external* and *internal* choice. While implication $A \multimap B$ as a computability statement clause in ILL could be applied only once, $!(A \multimap B)$ may be used an unbounded number of

times. When $A \multimap B$ is applied, then literal A becomes deleted from and B inserted to the current set of literals. If there is no literal A available, then the clause cannot be applied. To increase the expressiveness of formulae, we use the following abbreviation $a^n = \underbrace{a \otimes \ldots \otimes a}_{n}$, for $n > 0$.

In order to illustrate the above-mentioned features let us consider the following LL sequent from [116]:

$$(D \otimes D \otimes D \otimes D \otimes D) \vdash (H \otimes C \otimes (O \& S) \otimes !F \otimes (P \oplus I)),$$

which encodes a fixed price menu in a fast-food restaurant: for 5 dollars (D) you can get an hamburger (H), a coke (C), either onion soup O or salad S depending, which one *you* select, all the french fries (F) you can eat plus a pie (P) or an ice cream (I) depending on availability (restaurant owner selects for you). The formula $!F$ here means that we can use or generate a resource F as much as we want—the amount of the resource is unbounded.

Lincoln [117] summarises complexity results for several fragments of LL. Propositional multiplicative additive LL (MALL) is indicated to be PSPACE-complete, whilst first-order MALL is at most NEXPTIME-hard. If we would discard additives $\oplus$ and & from MALL, we would get multiplicative LL (MLL). Both, propositional and first-order MLL, are NP-complete. According to Lincoln these complexity results do not change, if respective intuitionistic fragments of LL are considered. These results hint that for practical computations either MLL or propositional MALL (or their intuitionistic variants MILL and MAILL (IMALL), respectively) might be applied.

3.2 Basics of partial deduction

In this section we present definitions of the basic concepts of partial deduction for ILL. The names of introduced concepts are largely influenced by the computation-oriented nature of our applications, where we intend to apply the framework.

3.2.1 Partial deduction and LL

Partial deduction (PD) (or partial evaluation of logic programs first introduced in [89]) is known as one optimisation technique in logic programming. Given a logic program, partial deduction derives a more specific program while preserving the meaning of the original program. Since the program is more specialised, it is usually more efficient than the original program, if executed. For instance, let A, B, C and D be propositional variables and $A \multimap B$, $B \multimap C$ and $C \multimap D$ computability statements in LL. Then possible partial deductions are $A \multimap C$, $B \multimap D$ and $A \multimap D$. It is easy to notice that the first corresponds to forward chaining (from initial states to goals), the second to backward chaining (from goals to initial states) and the third could be either forward or backward chaining.

Although the original motivation behind PD was to deduce specialised logic programs with respect to a given goal, our motivation for PD is a bit different. We are applying PD for determining subtasks, which cannot be performed by a single agent, but still are possibly closer to a solution than an initial task. This means that given a state S and a goal G of an agent we compute a new state S' and a new goal G'. This information is forwarded to another agent for further inference. From PD point of view this means that the program $\Gamma \vdash S' \multimap G'$ would be derived from $\Gamma \vdash S \multimap G$. Then the derived program is sent to other entities, who modify it further.

Lloyd and Shepherdson [120] considered completeness and soundness issues of PD for classical logic programs. Recently a PD formalisation for fluent calculus was proposed by Lehmann and Leuschel [111]. Anyway, it turns out that there is no work considering PD for LL. This gap would be filled in this paper by providing a formal foundation of PD for LL as a framework for CPS. We also consider completeness and soundness of PD for LL in this paper.

3.2.2 Basic definitions

Definition 1 *A program stack is a multiplicative conjunction*

$$\bigotimes_{i=1}^{n} A_i,$$

where $A_i, i = 1 \ldots n$ is a ILL formula.

Definition 2 *Mapping from a multiplicative conjunction to a set of conjuncts is defined as follows:*

$$\left[\bigotimes_{i}^{n} A_i\right] = \{A_1, \ldots, A_n\}$$

Definition 3 *Consumption of formula A_i from a program stack $\mathcal{S}$ is a mapping*

$$A_1 \otimes \ldots \otimes A_{i-1} \otimes A_i \otimes A_{i+1} \otimes \ldots \otimes A_n \mapsto_{\mathcal{S},A_i} A_1 \otimes \ldots \otimes A_{i-1} \otimes A_{i+1} \otimes \ldots \otimes A_n,$$

where $A_j, j = 1 \ldots n$ could be any valid formula in ILL.

Definition 4 *Generation of formula A_i to a program stack $\mathcal{S}$ is a mapping*

$$A_1 \otimes \ldots \otimes A_{i-1} \otimes A_{i+1} \otimes \ldots \otimes A_n \mapsto_{\mathcal{S},A_i} A_1 \otimes \ldots \otimes A_{i-1} \otimes A_i \otimes A_{i+1} \otimes \ldots \otimes A_n,$$

where $A_j, j = 1 \ldots n$ and A_i could be any valid formulae in ILL.

Definition 5 *A Computation Specification Clause (CSC) is a ILL sequent*

$$\vdash I \multimap_f O,$$

where I and O are multiplicative conjunctions of any valid ILL formulae and f is a function, which implements the computation step. I and O are respectively consumed and generated from the current program stack S, when a particular CSC is applied.

It has to be mentioned that a CSC can be applied only, if $[I] \subseteq [S]$. Although in ILL CSCs are represented as linear implication formulae, we represent them as extra-logical axioms in our problem domain. This means that an extra-logical axiom $\vdash I \multimap_f O$ is basically equal to ILL formula $!(I \multimap_f O)$.

Definition 6 *A Computation Specification (CS) is a finite set of CSCs.*

Definition 7 *A Computation Specification Application (CSA) is defined as*

$$\Gamma; S \vdash G,$$

where Γ *is a CS,* S *is the initial program stack and G the goal program stack.*

Definition 8 *Resultant is a CSC*

$$\vdash I \multimap_{\lambda a_1,\ldots,a_n.f} O, n \geq 0,$$

where f *is a term representing a function, which generates O from I by applying potentially composite functions over* $a_1, \ldots, a_n$.

CSA determines which CSCs could be applied by PD steps to derive resultant $\vdash S \multimap_{\lambda a_1,\ldots,a_n.f} G, n \geq 0$. It should be noted that resultants are derived by applying PD steps to the CSAs, which are represented in form $A \vdash B$. The CSC form is achieved from particular programs stacks by implicitly applying the following inference figure:

$$\dfrac{\vdash A \multimap B \qquad \dfrac{\dfrac{}{A \vdash A}\,Id \qquad \dfrac{}{B \vdash B}\,Id}{A, A \multimap B \vdash B}\,L\multimap}{A \vdash B}\,Cut$$

While resultants encode computations, program stacks represent computations' pre- and postconditions.

3.2.3 PD steps

This section defines all PD steps, which are used in our formalism.

Basic propositional steps

Definition 9 *Forward chaining PD step* $\mathcal{R}_f(L_i)$ *is defined as a rule*

$$\frac{B \otimes C \vdash G}{A \otimes C \vdash G}\,\mathcal{R}_f(L_i)$$

where L_i *is a labelling of CSC* $\vdash A \multimap_{L_i} B$. *A, B, C and G are ILL formulae.*

Definition 10 *Backward chaining PD step $\mathcal{R}_b(L_i)$ is defined as a rule*

$$\frac{S \vdash A \otimes C}{S \vdash B \otimes C}\ \mathcal{R}_b(L_i)$$

where L_i is a labelling of CSC $\vdash A \multimap_{L_i} B$. A, B, C and S are ILL formulae.

PD steps $\mathcal{R}_f(L_i)$ and $\mathcal{R}_b(L_i)$, respectively, apply CSC L_i to move the initial program stack towards the goal stack or vice versa. In the $\mathcal{R}_b(L_i)$ inference figure formulae $B \otimes C$ and $A \otimes C$ denote respectively an original goal stack G and a modified goal stack G'. Thus the inference figure encodes that, if there is an CSC $\vdash A \multimap_{L_i} B$, then we can change goal stack $B \otimes C$ to $A \otimes C$. Similarly, in the inference figure $\mathcal{R}_f(L_i)$ formulae $B \otimes C$ and $A \otimes C$ denote, respectively, an original initial stack S and its modification S'. And the inference figure encodes that, if there is a CSC $\vdash A \multimap_{L_i} B$, then we can change initial program stack $A \otimes C$ to $B \otimes C$.

PD steps for managing unbounded resources

In order to manage access to unbounded resources, we need PD steps $\mathcal{R}_{C_l}$, $\mathcal{R}_{L_l}$, $\mathcal{R}_{W_l}$ and $\mathcal{R}_{!_l}(n)$. The inference figures reflect directly LL rules $!C$, $!L$ and $!W$.

Definition 11 *PD step $\mathcal{R}_{C_l}$ is defined as a rule*

$$\frac{!A \otimes !A \otimes B \vdash C}{!A \otimes B \vdash C}\ \mathcal{R}_{C_l}$$

where A, B and C are ILL formulae.

Definition 12 *PD step $\mathcal{R}_{L_l}$ is defined as a rule*

$$\frac{A \otimes B \vdash C}{!A \otimes B \vdash C}\ \mathcal{R}_{L_l}$$

where A, B and C are ILL formulae.

Definition 13 *PD step $\mathcal{R}_{W_l}$ is defined as a rule*

$$\frac{B \vdash C}{!A \otimes B \vdash C}\ \mathcal{R}_{W_l}$$

where A, B and C are ILL formulae.

Definition 14 *PD step $\mathcal{R}_{!_l}(n), n > 0$ is defined as a rule*

$$\frac{!A \otimes A^n \otimes B \vdash C}{!A \otimes B \vdash C}\ \mathcal{R}_{!_l}(n)$$

where A, B and C are ILL formulae. $A^n = \underbrace{A \otimes \ldots \otimes A}_{n}$, for $n > 0$.

First-order PD steps

Considering the first-order ILL we have to replace PD steps $\mathcal{R}_f(L_i)$ and $\mathcal{R}_b(L_i)$ with their respective first-order variants $\mathcal{R}_f(L_i(\underline{x}))$ and $\mathcal{R}_b(L_i(\underline{x}))$. Other PD steps can remain the same. We also require that the initial and the goal program stack are ground.

Definition 15 *First-order forward chaining PD step $\mathcal{R}_f(L_i(\underline{x}))$ is defined as a rule*

$$\frac{B \otimes C \vdash G}{A \otimes C \vdash G}\ \mathcal{R}_f(L_i(\underline{x}))$$

Definition 16 *First-order backward chaining PD step $\mathcal{R}_b(L_i(\underline{x}))$ is defined as a rule*

$$\frac{S \vdash A \otimes C}{S \vdash B \otimes C}\ \mathcal{R}_b(L_i(\underline{x}))$$

In the above definitions A, B, C are ILL formulae and $L_i(\underline{x})$ is defined as $\vdash \forall \underline{x}(A' \multimap_{L_i(\underline{x})} B')$. Additionally we assume that $\underline{a} \stackrel{def}{=} a_1, a_2, \ldots$ is an ordered set of constants, $\underline{x} \stackrel{def}{=} x_1, x_2, \ldots$ is an ordered set of variables, $[\underline{a}/\underline{x}]$ denotes substitution, and $X = X'[\underline{a}/\underline{x}]$. When substitution is applied, elements in $\underline{a}$ and $\underline{x}$ are mapped to each other in the order they appear in the ordered sets. These sets must have the same number of elements.

PD step for constructing nondeterministic resultants

Nondeterministic resultants can be generated basically in two ways. First, there may exist a particular CSC having nondeterministic effects. Second, there is an internal mechanism for creating such resultants from scratch. Since in the first case nondeterministic resultants are achieved via basic PD forward and backward steps, we consider here only the second case.

Definition 17 *Forward chaining PD step Branch is defined as a rule*

$$\frac{\mathop{\&}\limits_{i=1}^{k} A_i \vdash B_1 \quad \ldots \quad \mathop{\oplus}\limits_{i=l}^{n} A_i \vdash B_m}{\mathop{\otimes}\limits_{i=1}^{n} A_i \vdash \mathop{\otimes}\limits_{i=1}^{m} B_i}\ \mathit{Branch}$$

where $1 \leq l, k \leq n, n > 1$.

This step generates multiple nondeterministic interdependent resultants at once. It means that the set of derived resultants should be considered as a single problem—a solution to the original resultant is found if and only if solutions to *all* derived resultants are found.

3.2.4 Derivation and PD

Definition 18 (Derivation of a resultant) *Let $\mathcal{R}$ be any predefined PD step. A derivation of a resultant R_0 is a finite sequence of resultants: $R_0 \Rightarrow_{\mathcal{R}} R_1 \Rightarrow_{\mathcal{R}} R_2 \Rightarrow_{\mathcal{R}} \ldots \Rightarrow_{\mathcal{R}} R_n$, where $\Rightarrow_{\mathcal{R}}$ denotes an application of PD step $\mathcal{R}$.*

Definition 19 (Partial deduction) *Partial deduction of a CSA $\Gamma; S \vdash G$ is a set of all resultants R_i derivable from CSC $\vdash S \multimap G$.*

It is easy to see that this definition of PD generates the set of all proof trees for CSA $\Gamma; S \vdash G$. Due to the non-monotonicity of LL we need a sort of backtracking mechanism in our formalism for preserving completeness. Therefore we need backtracking ability, which is achieved by keeping the all the proof trees encountered.

Definition 20 *A CSA $\Gamma; S \vdash G$ is executable, iff given Γ as a CS, resultant $\vdash S \multimap_{\lambda a_1,\ldots,a_n.f} G, n \geq 0$ can be derived such that derivation ends with resultant R_n, which equals to $\vdash A \multimap A$, where A is a program stack.*

3.3 A motivating example

To illustrate the PD process in the symbolic negotiation context, let us consider the following example. Let us have 3 agents representing a musician $\mathcal{M}$, a writer $\mathcal{W}$ and an artist $\mathcal{A}$. They all have personal goals they would like to achieve. We would like to emphasise that this example is supposed to demonstrate syntactical and computational aspects only and no pragmatic issues are considered here. More practical examples have been considered in the field of automated Web service synthesis [159]. These examples involve more extra-logical axioms and longer proofs. They are, however, not so sophisticated from formal point of view.

The musician would like to go out with her husband and therefore needs 2 concert tickets. Unfortunately the concert, she is interested in, has been sold out and therefore the only way to acquire the tickets is to ask them from other agents. In return she can grant a certain book and unlimited access to digital version of her albums. Thus

$$G_{\mathcal{M}} = \{Ticket^2\},$$

$$S_{\mathcal{M}} = \{!MP3 \otimes Book\}$$

and

$$\Gamma_{\mathcal{M}} = \emptyset.$$

The artist has promised to perform at a conference and thus needs 2 hours of background music and an MP3 player. Since the performance takes place at the same time as the concert he can give away its concert ticket. Formally,

$$G_{\mathcal{A}} = \{Perf\},$$

$$S_{\mathcal{A}} = \{Ticket\}$$

and

$$\Gamma_{\mathcal{A}} = \{\vdash MP3^2 \otimes MP3Player \multimap_{perform} Perf\}.$$

The writer wants to relax and this can be achieved by reading a book and listening to music. He has both—a CD player and an MP3 player. Additionally he can write CDs from MP3 files. He also has a ticket to the same concert with the artist. However, he prefers staying at home this time. Thus formally this is described as follows:

$$G_{\mathcal{W}} = \{Relaxed\},$$

$$S_{\mathcal{W}} = \{Ticket \otimes CDPlayer \otimes MP3Player\},$$

$$\Gamma_{\mathcal{W}} = \begin{array}{l} \vdash CD \otimes CDPlayer \multimap_{playCD} Music, \\ \vdash MP3 \otimes MP3Player \multimap_{playMP3} Music, \\ \vdash Music \otimes Book \multimap_{relax} Relaxed, \\ \vdash MP3 \multimap_{burnCD} CD. \end{array}$$

Let us describe now how PD can be applied in the symbolic negotiation process between these 3 agents. Since we have not formalised yet the symbolic negotiation process, we only show the PD part of the process. The negotiation is initiated by agents $\mathcal{M}$ and $\mathcal{W}$. Agent $\mathcal{M}$ is unsure whether anyone has two tickets left to the concert. Therefore she decides to propose 2 separate offers (resultants) instead of a single one:

$$!MP3 \oplus Book \vdash Ticket$$

and

$$!MP3\&Book \vdash Ticket$$

Offers $!MP3 \oplus Book \vdash Ticket$ and $!MP3\&Book \vdash Ticket$ were achieved from $S_{\mathcal{M}}$ and $G_{\mathcal{M}}$ by applying inference figure *Branch* in the following manner:

$$\frac{!MP3 \oplus Book \vdash Ticket \quad !MP3\&Book \vdash Ticket}{!MP3 \otimes Book \vdash Ticket^2}\ Branch$$

While $!MP3 \oplus Book \vdash Ticket$ gives a receiver an opportunity to choose between $!MP3$ and $Book$, $!MP3\&Book \vdash Ticket$ states that the receiver would get either $!MP3$ or $Book$, but the choice is made by the receiver of $!MP3 \oplus Book \vdash Ticket$. This is intuitive

since the second receiver has no idea whether the first receiver would choose either !*MP3* or *Book*.

The resultants describe internal and external choices in LL and are represented with operators & and ⊕ respectively. While ⊕ from the sender's point of view gives choice to the receiver, & is the opposite—the sender makes the decision of which resource to deliver.

Agent $\mathcal{W}$ sends out the following resultant:

$$Ticket \otimes MP3Player \vdash MP3 \otimes Book,$$

which means that he can trade a ticket and an MP3 player for an hour of MP3 music and a book. The resultant was achieved in the following way:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{Ticket \otimes MP3Player \vdash MP3 \otimes Book \quad \overline{CDPlayer \vdash CDPlayer}\ Id}{Ticket \otimes CDPlayer \otimes MP3Player \vdash MP3 \otimes CDPlayer \otimes Book}\ L\otimes, R\otimes}{Ticket \otimes CDPlayer \otimes MP3Player \vdash CD \otimes CDPlayer \otimes Book}\ \mathcal{R}_b(burnCD)}{Ticket \otimes CDPlayer \otimes MP3Player \vdash Music \otimes Book}\ \mathcal{R}_b(playCD)}{Ticket \otimes CDPlayer \otimes MP3Player \vdash Relaxed}\ \mathcal{R}_b(relax)$$

After applying the full symbolic negotiation process (see Chapter 4) the result, depicted in Figure 3.1, is achieved. Circles denote there resource exchange events between agents, formalised later as symbolic negotiation primitives, while rounded rectangles represent execution of agents' capabilities determined through PD. The vertical arrows between circles and rectangles represent ordering of activities, which was achieved through symbolic negotiation. The horizontal arrows represent which resources are exchanged at particular time points.

3.4 Soundness and completeness of PD in ILL

3.4.1 PD steps as inference figures in ILL

In this section we prove that PD steps are inference figures in ILL.

Proposition 1 *Forward chaining PD step $\mathcal{R}_f(L_i)$ is sound with respect to ILL rules.*

Proof The proof in ILL follows here:

$$\dfrac{\dfrac{\overline{A \otimes C \vdash A \otimes C}\ Id \quad \overline{\vdash (A \multimap_{L_i} B)}\ Axiom}{A \otimes C \vdash A \otimes C \otimes (A \multimap_{L_i} B)}\ R\otimes \qquad \dfrac{\dfrac{\overline{C \vdash C}\ Id \quad \dfrac{\dfrac{\overline{A \vdash A}\ Id \quad \overline{B \vdash B}\ Id}{A, (A \multimap_{L_i} B) \vdash B}\ L\multimap}{A \otimes (A \multimap_{L_i} B) \vdash B}\ L\otimes}{C, A \otimes (A \multimap_{L_i} B) \vdash B \otimes C}\ R\otimes}{A \otimes C \otimes (A \multimap_{L_i} B) \vdash B \otimes C}\ L\otimes \qquad B \otimes C \vdash G}{A \otimes C \otimes (A \multimap_{L_i} B) \vdash G}\ Cut}{A \otimes C \vdash G}\ Cut$$

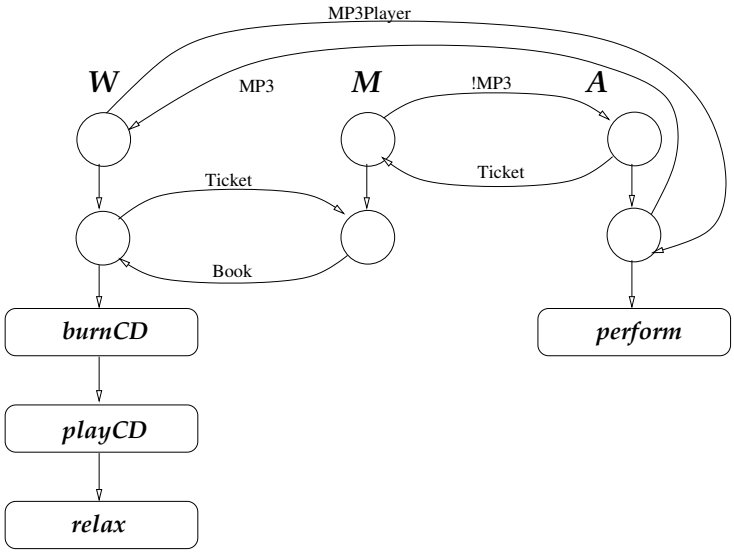


Figure 3.1: The result of symbolic negotiation.

Proposition 2 *Backward chaining PD step $\mathcal{R}_b(L_i)$ is sound with respect to ILL rules.*

Proof The proof in ILL follows here:

$$\dfrac{\dfrac{S \vdash A \otimes C \quad \dfrac{}{\vdash (A \multimap_{L_i} B)}\,Axiom}{S \vdash A \otimes C \otimes (A \multimap_{L_i} B)}\,R\otimes \qquad \dfrac{\dfrac{\dfrac{}{C \vdash C}\,Id \quad \dfrac{\dfrac{\dfrac{}{A \vdash A}\,Id \quad \dfrac{}{B \vdash B}\,Id}{A, (A \multimap_{L_i} B) \vdash B}\,L\multimap}{A \otimes (A \multimap_{L_i} B) \vdash B}\,L\otimes}{C, A \otimes (A \multimap_{L_i} B) \vdash B \otimes C}\,R\otimes}{A \otimes C \otimes (A \multimap_{L_i} B) \vdash B \otimes C}\,L\otimes}{S \vdash B \otimes C}\,Cut$$

Proposition 3 *PD step $\mathcal{R}_{C_l}$ is sound with respect to ILL rules.*

Proof The proof in ILL follows here:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{!A \vdash !A}\,Id \quad \dfrac{}{!A \vdash !A}\,Id}{!A, !A \vdash !A \otimes !A}\,R\otimes}{!A \vdash !A \otimes !A}\,C! \quad \dfrac{}{B \vdash B}\,Id}{!A, B \vdash !A \otimes !A \otimes B}\,R\otimes}{!A \otimes B \vdash !A \otimes !A \otimes B}\,L\otimes \qquad !A \otimes !A \otimes B \vdash C}{!A \otimes B \vdash C}\,Cut$$

Proposition 4 *PD step $\mathcal{R}_{L_l}$ is sound with respect to ILL rules.*

Proof The proof in ILL follows here:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,Id}{!A \vdash A}\,L! \quad \dfrac{}{B \vdash B}\,Id}{!A, B \vdash A \otimes B}\,R\otimes}{!A \otimes B \vdash A \otimes B}\,L\otimes \qquad A \otimes B \vdash C}{!A \otimes B \vdash C}\,Cut$$

Proposition 5 *PD step $\mathcal{R}_{W_l}$ is sound with respect to ILL rules.*

Proof The proof in ILL follows here:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{B \vdash B}\,Id}{!A, B \vdash B}\,W!}{!A \otimes B \vdash B}\,L\otimes \qquad B \vdash C}{!A \otimes B \vdash C}\,Cut$$

Proposition 6 *PD step $\mathcal{R}_{!_l}$ is sound with respect to ILL rules.*

Proof The proof in ILL follows here:

$$\dfrac{\dfrac{\dfrac{!A \otimes A^n \otimes B \vdash C}{\vdots}}{!A \otimes A \otimes B \vdash C}\,\mathcal{R}_{L_l}}{!A \otimes !A \otimes B \vdash C}\,\mathcal{R}_{C_l}$$

Proposition 7 *First-order forward chaining PD step $\mathcal{R}_f(L_i(\underline{x}))$ is sound with respect to first order ILL rules.*

Proof The proof in ILL is the following

$$\dfrac{\dfrac{\dfrac{}{A \otimes C \vdash A \otimes C}\,Id \quad \dfrac{}{\vdash \forall \underline{x}(A' \multimap_{L_i(\underline{x})} B')}\,Axiom}{A \otimes C \vdash A \otimes C \otimes (\forall \underline{x}(A' \multimap_{L_i(\underline{x})} B'))}\,R\otimes \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,Id \quad \dfrac{}{B \vdash B}\,Id}{A, (A \multimap_{L_i(\underline{a})} B) \vdash B}\,L\multimap}{\dfrac{}{C \vdash C}\,Id \quad A \otimes (A \multimap_{L_i(\underline{a})} B) \vdash B}\,L\otimes}{C, A \otimes (A \multimap_{L_i(\underline{a})} B) \vdash B \otimes C}\,R\otimes}{A \otimes C \otimes (A \multimap_{L_i(\underline{a})} B) \vdash B \otimes C}\,L\otimes}{A \otimes C \otimes (\forall \underline{x}(A' \multimap_{L_i(\underline{x})} B')) \vdash B \otimes C}\,L\forall \qquad B \otimes C \vdash G}{A \otimes C \otimes (\forall \underline{x}(A' \multimap_{L_i(\underline{x})} B')) \vdash G}\,Cut}{A \otimes C \vdash G}\,Cut$$

Proposition 8 *First-order backward chaining PD step $\mathcal{R}_b(L_i(\underline{x}))$ is sound with respect to first order ILL rules.*

Proof The proof in ILL is the following

$$\dfrac{\dfrac{S \vdash A \otimes C \quad \dfrac{}{\vdash \forall \underline{x}(A' \multimap_{L_i(\underline{x})} B')}\,Axiom}{S \vdash A \otimes C \otimes (\forall \underline{x}(A' \multimap_{L_i(\underline{x})} B'))}\,R\otimes \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,Id \quad \dfrac{}{B \vdash B}\,Id}{A, (A \multimap_{L_i(\underline{a})} B) \vdash B}\,L\multimap \quad \dfrac{}{C \vdash C}\,Id}{C, A \otimes (A \multimap_{L_i(\underline{a})} B) \vdash B \otimes C}\,R\otimes}{A \otimes C \otimes (A \multimap_{L_i(\underline{a})} B) \vdash B \otimes C}\,L\otimes}{A \otimes C \otimes (\forall \underline{x}(A' \multimap_{L_i(\underline{x})} B')) \vdash B \otimes C}\,L\forall}{S \vdash B \otimes C}\,Cut$$

Proposition 9 *Forward chaining PD step Branch is sound with respect to ILL rules.*

Proof The proof of forward chaining PD step *Branch* is the following

$$\dfrac{\dfrac{\vdots}{\bigotimes_{i=1}^{n} A_i \vdash (\mathop{\&}_{i=1}^{k} A_i) \otimes \ldots \otimes (\bigoplus_{i=l}^{n} A_i)}\,Rewrite \qquad \dfrac{\dfrac{\bigoplus_{i=l}^{n} A_i \vdash B_m}{\vdots}}{\dfrac{\mathop{\&}_{i=1}^{k} A_i \vdash B_1 \quad \ldots \otimes (\bigoplus_{i=l}^{n} A_i) \vdash \bigotimes_{i=2}^{m} B_i}{(\mathop{\&}_{i=1}^{k} A_i), \ldots \otimes (\bigoplus_{i=l}^{n} A_i) \vdash \bigotimes_{i=1}^{m} B_i}\,R\otimes}\,L\otimes}{\bigotimes_{i=1}^{n} A_i \vdash \bigotimes_{i=1}^{m} B_i}\,Cut$$

where $1 \leq l, k \leq n, n > 1$. Additionally, the number of nondeterministic branches cannot be larger than n, since we have only n formulae. Moreover, the number of $\oplus$-resultants is not greater than $n/2$. The latter derives from 2 assumptions: (1) for a choice at least 2 formulae are needed and (2) $\oplus$-resultants must not overlap. Generally, we assume that we can insert as many constants 1 as needed to enlarge m, since m has to be greater than or equal to the number of branches, which is limited by n.

While the right branch of that inference figure generates multiple nondeterministic offers at once, the left branch ensures consistency of the offers. *Rewrite* indicates that the right hand side of a sequent is transformed to disjunctive normal form with respect to the & operator.

3.4.2 Soundness and completeness

Soundness and completeness are defined via executability of CSAs.

Definition 21 (Soundness of PD of a CSA) *A CSC* $\vdash S' \multimap G'$ *is executable, if a CSC* $\vdash S \multimap G$ *is executable in a CSA* $\Gamma; S \vdash G$ *and there is a derivation* $\vdash S \multimap G \Rightarrow_{\mathcal{R}} \ldots \Rightarrow_{\mathcal{R}} \vdash S' \multimap G'$.

Completeness is the converse:

Definition 22 (Completeness of PD of a CSA) *A CSC* $\vdash S \multimap G$ *is executable, if a CSC* $\vdash S' \multimap G'$ *is executable in a CSA* $\Gamma; S' \vdash G'$ *and there is a derivation* $\vdash S \multimap G \Rightarrow_{\mathcal{R}} \ldots \Rightarrow_{\mathcal{R}} \vdash S' \multimap G'$.

Our proofs of soundness and completeness are based on proving that derivation of a resultant is a derivation in a CSA using PD steps, which were defined as inference figures in ILL. However, it should be emphasised that soundness and completeness of PD as defined here have no relation with respective properties of LL.

Lemma 1 *A CSC* $\vdash S \multimap G$ *is executable, if there is a proof of* $\Gamma; S \vdash G$ *in ILL.*

Proof Since the derivation of a resultant is based on PD steps, which represent particular inference figures in ILL, then if there is a ILL proof for $\Gamma; S \vdash G$, based on inference figures in Section 3.4.1, then the proof can be transformed to a derivation of resultant $\vdash S \multimap G$.

Lemma 2 *Resultants in a derivation are nodes in the respective ILL proof tree and they correspond to partial proof trees, where leaves are other resultants.*

Proof Since each resultant $\vdash A \multimap B$ in a derivation is achieved by an application of a PD step, which is defined with a respective ILL inference figure, then it represents a node $A \vdash B$ in the proof tree, whereas the derivation of $\vdash A \multimap B$ represents a partial proof tree.

Theorem 1 (Soundness of propositional PD) *PD for LL in propositional ILL is sound.*

Proof According to Lemma 1 and Lemma 2 PD for LL in propositional ILL is sound, if we apply basic propositional PD steps, steps for managing unbounded resources and *Branch* step. The latter derives from the fact that, if there exists a derivation $\vdash S \multimap G \Rightarrow_{\mathcal{R}} \ldots \Rightarrow_{\mathcal{R}} \vdash S' \multimap G'$, then the derivation is constructed by PD in a formally correct manner.

Theorem 2 (Completeness of propositional PD) *PD for LL in propositional ILL is complete.*

Proof When applying PD with propositional PD steps, we first generate all possible derivations until no derivations could be found, or all proofs have been found. If CSC $\vdash S' \multimap G'$ is executable then according to Lemma 1, Lemma 2 and Definition 20 there should be a path in the ILL proof tree starting with CSC $\vdash S \multimap G$, ending with $\vdash A \multimap A$ and containing CSC $\vdash S' \multimap G'$. There is no possibility to have a path from CSC $\vdash S' \multimap G'$ to $\vdash A \multimap A$ without having a path from CSC $\vdash S \multimap G$ to CSC $\vdash S' \multimap G'$ in the same HLL proof tree.

Then according to Lemma 1 and Lemma 2, derivation $\vdash S \multimap G \Rightarrow_{\mathcal{R}} \ldots \Rightarrow_{\mathcal{R}} \vdash S' \multimap G'$ would be either discovered or it will be detected that there is no such derivation. Therefore PD for LL in ILL is complete.

Theorem 3 (Soundness of PD of a first-order CSA) *PD for LL in first-order ILL is sound.*

Proof The proof follows the pattern of the proof for Theorem 1, with the difference that instead of applying PD steps $\mathcal{R}_b(L_i)$ and $\mathcal{R}_f(L_i)$, we apply their first-order counterparts $\mathcal{R}_b(L_i(\underline{x}))$ and $\mathcal{R}_f(L_i(\underline{x}))$.

Theorem 4 (Completeness of PD of a first-order CSA) *PD for LL in first-order ILL is complete.*

Proof The proof follows the pattern of the proof for Theorem 2, with the difference that instead of applying PD steps $\mathcal{R}_b(L_i)$ and $\mathcal{R}_f(L_i)$, we apply their first-order counterparts $\mathcal{R}_b(L_i(\underline{x}))$ and $\mathcal{R}_f(L_i(\underline{x}))$.

3.5 Partial deduction strategies

The practical value of PD is very limited without defining appropriate PD strategies. These are called tactics and refer to selection and stopping criteria. Successful tactics depend generally quite much on a specific logic application. Therefore we only list some possible tactics here. From agent negotiation point of view the strategies represent to some extent agents' policies—they determine which offers are proposed next.

Tammet [181] proposes a set of theorem proving strategies for speeding up LL theorem proving. He also presents experimental results, which indicate a good performance of the proposed strategies. Some of his strategies remind the usage of our inference figures. Thus some LL theorem proving strategies are already implicitly handled in our PD framework.

We also would like to point out that by using LL inference figures instead of basic LL rules, PD, as we defined it here, could be more efficient than pure LL theorem proving. The latter is due to the smaller search space, which emerges through the usage of inference figures.

Definition 23 *Length l of a derivation is equal to the number of the applications of PD steps $\mathcal{R}$ in the derivation.*

Definition 24 *Two derivations are computationally equivalent, regardless of the length of their derivations, if they both start and end with the same resultant.*

3.5.1 Selection criteria

Selection criteria define which formulae and PD steps should be considered next for derivation of a resultant. We consider the following selection criteria.

- Mixed backward and forward chaining—a resultant is derived by interleaving backward and forward chaining.
- Different search methods—depth-first, breadth-first, iterative deepening, etc could be used. While breadth-first allows discovering shorter derivations faster, depth-first requires less computational overhead, since less memory is used for storing the current search status.
- Prefer resultants with smaller derivation length—the strategy implicitly leads to breadth-first search.
- Apply only one PD step at time.
- Combine several PD steps together. The approach is justified, if there is some domain knowledge available, which states that certain CSCs are executed in sequence.
- Priority-based selection—some literals have a higher weight, which is determined either manually by the user or calculated by the system according to predefined criteria. During PD literals/resultants having higher weights are preferred.

We would like to emphasise that the above criteria are not mutually exclusive but rather complementary to each other.

3.5.2 Stopping criteria

Stopping criteria define when to stop derivation of resultants. They could be combined with the above-mentioned selection criteria. We suggest the following stopping criteria:

- The derived resultant is computationally equivalent to an already derived one—since the resultant was already derived and used in other derivations, proceeding PD again with the same resultant would not yield any new resultants, which would be computationally *not* equivalent compared to already derived ones.
- A generative cycle is detected—if we derived a resultant $\vdash A \multimap B \otimes C$ from a resultant $\vdash A \multimap C$, then by repeatedly applying PD steps between the former resultants we end up with resultants $\vdash A \multimap B^n \otimes C$, where $n > 1$. Therefore we can skip the PD steps in further derivation and reason analytically how many instances of literal B we need. The approach is largely identical to Karp-Miller [85] algorithm, which is applied for state space collapsing during Petri net reachability checking. A similar method is also applied by Andreoli et al [4] for analysing LL programs.

- Maximum derivation length l is reached—given that our computational resources are limited and the time for problem solving is limited as well, we may not be able to explore the full search space anyway. Then setting a limit to derivation length helps to constrain the search space.
- A resultant is equal to the goal—since we found a solution to the problem, there is no need to proceed further, unless we are interested in other solutions as well.
- Stepwise—the user is queried before each derivation in order to determine, which derivations s/he wants to perform. This stopping criterion could be used during debugging, since it provides the user with an overview of the derivation process.
- Exhaustive—derivation stops, when no new resultants are available.

3.6 Summary

In this chapter we formalised PD for ILL. We formalised the PD process and introduced PD steps, which could be considered as a specialisation of theorem proving for a specific application domain. The application domain in our case is fixed to reasoning about executable processes. The specialisation allows us to gain some extra speed while applying PD to solve problems in the domain.

We also analysed soundness and completeness of the proposed formalism. It turns out that, given a certain PD procedure, PD for ILL is sound and complete.

Finally some PD heuristics were described, which determine strategies of applying PD. The strategies included selection and stopping criteria. While selection criteria determine which resultants to derive next, stopping criteria determine when to stop derivation of new resultants. These strategies allow to specialise PD for different applications. One of the applications is described in Chapter 10.

Chapter 4

CPS and Symbolic Negotiation

In this chapter we formalise cooperative problem solving (CPS) and symbolic negotiation process as PD for LL, which was defined in Chapter 3. In heterogeneous multi-agent systems interoperability between agents cannot be taken for granted. Indeed, since agents may enter and leave a system at their will, there should exist a mechanism for automatically adjusting agents' behaviours and goals in order to keep the system in balance. Automatic negotiation and CPS are regarded as mechanisms for granting that sort of on-the-fly system integration and management.

Several attempts have been made in order to formalise CPS (see Chapter 2). Most of them are based on classical or modal logics. In particular, Wooldridge and Jennings [199] provide a formalisation of CPS process where a multi-modal logic is used as a formal specification language. However, since the multi-modal logic lacks a strategy for generating constructive proofs of satisfiability, the formalisation does not lead to direct execution of specifications. Moreover, since modal logics (like classical logic) lack the mechanism for keeping track of resources, it is not possible for agents neither to *count* nor dynamically update the number of instances of the same object belonging to their internal states. In order to overcome the mentioned shortages of classical and modal logics we use a fragment of LL for CPS.

The cooperative problem solving has been considered to consist of four steps [199]: recognition of potential for cooperation, team formation, plan formation and plan execution. An important feature of our approach is that we do not separate team and plan formation into different processes and that negotiation is embedded into the reasoning. Although this approach does not preserve the accepted structure of CPS, we think that it may be more natural for representing computational aspects of CPS, where team and plan formation processes interact with each other.

Basically, we are applying PD for generating constructive proofs summarising the first 3 steps of CPS: recognition, team and plan formation. Negotiation is reformulated as distributed PD. Then a solution, summarising the first 3 steps of CPS process, is extracted from a proof and can be executed.

In CPS models it is often implicitly expected that agents have knowledge about sequences of actions, whose execution leads them to their goals, while the sequence

construction process is not explicitly explained. Our CPS model is more planning-centric. Initially an agent tries to find a plan that allows achieving its goals. Then the agent may discover that either this is not possible or it is more efficient to involve other agents into problem solving process. Since other agents may be self-interested, they may propose their offers and start a negotiation process. The process lasts until a (shared) plan has been found. The plan determines agents' commitments and takes into account requirements determined during the negotiation.

In order to stimulate cooperation, agents should have a common goal [199]. We assume that all agents have a common *meta*-goal: as much agents as possible should become satisfied during run-time. All agents ask for minimum they need and provide maximum they can, during negotiation. This is biased with distributed theorem proving strategies. During negotiation the offers are derived using Partial Deduction (PD) in LL. PD allows determining missing links between proof fragments.

Finally we extend the proposed CPS formalism with plan modification operators and then analyse symbolic negotiation as a whole. We also formalise the coalition formation process and analyse its effect to CPS and symbolic negotiation. Regarding other previously proposed coalition formation methods, which are oriented to task allocation, our method could be described as goal-oriented.

We would like to underline that from a computational point of view, we can regard CPS as AI planning and symbolic negotiation as plan reuse/repair. It has been shown [142] that from problem solving point of view in general neither planning from scratch nor plan repair has an advantage over each-other. Therefore we expect both CPS and symbolic negotiation to be computationally equivalent. Moreover, both CPS and symbolic negotiation lead to the same results as we prove in this paper.

However, compared to CPS, symbolic negotiation provides a more human-like way of problem solving, which can be more naturally followed by human participants. In addition, symbolic negotiation may encode a sort of search heuristics, which would make CPS computationally less demanding. These heuristics, however, are not discussed in this paper.

Our approach supports detection of subgoals during problem solving. If a single agent fails to solve a problem, PD is applied to solve the problem partially. As a result subproblems are detected, which could be solved further by other agents. This would lead to a distributed problem solving mechanism, where different agents contribute to different phases in problem solving—each agent applies PD to solve a fragment of the problem and forwards the modified problem to others. As a result the problem becomes solved in the distributed manner. Usage of PD in such a way provides foundations for advanced interactions between agents.

4.1 Agent representation

An agent is presented with the following CSA:

$$\Gamma; S \vdash G,$$

where Γ is a set of CSCs representing agent's capabilities, S is the initial state and G is the goal state of the agent. Both S and G are multiplicative conjunctions of ILL formulae. Every element of Γ has the form

$$\vdash I \multimap O,$$

where I and O are formulae which are, respectively, consumed and generated when a particular capability is applied. It has to be mentioned that a capability can be applied only, if conjuncts in I form a subset of conjuncts in S. It should be also underlined that in order to achieve their goals, agents have to construct (and then execute) the following plan from the elements of Γ:

$$\vdash S \multimap G.$$

Each agent is assumed to have a single task only. Whenever in an application an agent is supposed to have several task, it would be modelled in our framework as several agents, each having a single task.

Definition 25 *Agent's task is a resultant in PD.*

Modifications of an agent's task are derived through PD. We consider all possible derivations of a task as instances of the same task. Moreover, we constrain an agent to consider only one derivation at time during distributed CPS. This would ensure that instances of the same task would not be solved concurrently.

Definition 26 *Agent's initial task is a CSA with* $\Gamma \equiv \emptyset$ *in PD.*

We model agent task solving as resource generation. This means that agents' tasks specify which resource they have and which resources they would like to obtain. Agents' capabilities are applied for producing new resources by consuming existing ones.

Definition 27 *Resource is defined as a formula in the underlying PD formalism.*

4.2 Agent coalitions

It may happen, that in order to solve a problem, agents should form coalitions. Let us consider a situation, where agent $\mathcal{A}$ possesses 50 dollars and agent $\mathcal{B}$ possesses 500 dollars. Agent $\mathcal{A}$ would like to obtain a radio and agent $\mathcal{B}$ would like to obtain a TV. A third agent C has a special offer for 550 dollars, if someone would buy both a TV and a radio. However, in the current situation none of the agents is able to fill their tasks individually. In order to perform their tasks, agents $\mathcal{A}$ and $\mathcal{B}$ have to first form a

coalition, then combine their resources and after the purchase task has been performed, they would divide the resulting resources as they agreed before forming a coalition.

Thus the purpose of agent coalitions is to merge several tasks into a larger one. Then agents become able to perform tasks, which they would not have been able to perform individually. Thereby coalition formation could be seen as a way to optimize problem solving globally.

Definition 28 *Agent coalition C is a structure $\langle \mathcal{A}, \mathcal{T}, \mathcal{R}, \triangleright, [] \rangle$, where $\mathcal{A}$ is a set of participating agents, $\mathcal{T}$ is a coalition task and $\mathcal{R}$ is a set of resources shared between agents. $\triangleright$ is a mapping from agents to resources ($\triangleright : \mathcal{A} \times \mathcal{A} \rightarrow \mathcal{R}^*$), which one agent would deliver to another if a solution for a coalition task is found. Finally, [] is a mapping from a coalition task to agents in the coalition ($[] : \mathcal{T} \rightarrow \mathcal{A}^+$), which agreed to solve the task together by sharing their resources.*

We interpret a coalition as an agreement between agents to solve their tasks together. In a coalition agents can also exchange resources directly between themselves. Hence mapping $\triangleright$ identifies a possibly empty set of resources, which one agent is required to transfer to another. Mapping [], however returns a non-empty set of agents, which have agreed to participate in CPS.

It should be emphasised that we model tasks as (composite) actions consuming and producing resources. Coalitions are formed by applying rule *merge*.

Definition 29 (Coalition formation rule) *Coalition formation rule merge is defined as follows:*

$$\frac{\vdash A \otimes B \multimap C \otimes D}{\vdash A \multimap C \quad \vdash B \multimap D}\ merge$$

Rule *merge* combines 2 tasks X and Y to create a task with the intended meaning: "task X could be performed by one agent, if another agent performs task Y" and vice versa. From a distributed theorem proving point of view it means that tasks X and Y are combined together and their search spaces are merged. If a coalition includes all agents of a system, then problem solving is seen as solving a single task in a distributed manner. This contrasts with our general case, where multiple tasks are solved concurrently in a distributed way.

In the beginning of a problem solving session there is one coalition for each agent. These coalitions consist only of the agent itself. The rule *merge* can be applied only once by the same agent during each problem solving session. If agents would merge several instances of the same task or its derivations, it would mean that some tasks would occur several times in the same coalition task. Then again the coalition is required to spend more resources than necessary and would achieve more resources than required. Finally, there may not be a solution for such a redundant task, since LL is resource-conscious.

Similarly an agent cannot belong to multiple coalitions with the same task at the same time. This would mean that an agent would agree to exchange particular resources with multiple parties. However, after exchanging it with one agent, it cannot

exchange it with others anymore. Neither can an agent participate in the same coalition with multiple instances of the same task. To summarise, new coalition can be formed through *merge* only if the following holds:

$$([\mathcal{T}_1] \cap [\mathcal{T}_2]) \equiv \emptyset,$$

where $\mathcal{T}_1$ and $\mathcal{T}_2$ are tasks of particular coalitions, which are considered for merging. This means that new coalitions are formed in 3 ways:

1. Between agents not belonging to any coalition
2. Between a coalition and an agent not belonging to any coalition
3. Between coalitions, whose members do not overlap (actually they cannot, since we constrain an agent to belong to maximum one coalition only)

Proposition 10 *Solution of a coalition task $\mathcal{T}$, which was achieved through PD, solves the tasks of all coalition members* $[\mathcal{T}]$.

Proof Since the coalition task $\mathcal{T}$ is composed from the individual tasks of its members, then after a coalition task solution has been found through PD, agents share the achieved resources between themselves. Additionally, resources between coalition members are exchanged such that agent $\mathcal{A}$ would give to agent $\mathcal{B}$ resources $\mathcal{A} \triangleright \mathcal{B}$. This means that coalition members would exchange resources, which were not used for solving a coalition task, but still are required by participating agents.

Definition 30 (Soundness of coalition formation) *If a solution of coalition task $\mathcal{T}$ solves the tasks of all participating agents* $[\mathcal{T}]$*, then coalition formation is sound.*

Proposition 11 *Coalition formation is sound with respect to PD.*

Proof According to Proposition 10 and Definition 30 coalition formation is sound.

Coalition formation could be seen from problem solving point of view as merging several tasks into one. Thus instead of solving multiple tasks concurrently, with coalitions we would solve a composed task.

Proposition 12 *If a set of tasks included in a coalition task is solvable through PD independently, then so is the composed coalition task.*

Proof Let us assume that all tasks are solvable independently and for any agent $\mathcal{A}$ and $\mathcal{B}$ from the coalition $\mathcal{A} \triangleright \mathcal{B} \equiv \emptyset$. Then the coalition task solution would be a concatenation of solutions of individual tasks. However, if there are agents $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A} \triangleright \mathcal{B} \not\equiv \emptyset$, then some resources do not have to be spent and achieved by the coalition task. Therefore the coalition task solution would be shorter than the concatenation of solutions of all individual tasks and yet achievable.

If the tasks of agents are solvable independently, then so are they after joining a coalition. Since during coalition formation particular search spaces are merged, the overall search space would include the search spaces of individual tasks. The reverse of the proposition does not hold, since there may exist resources, which cannot be achieved by any task, but which could be exchanged between agents in a coalition.

Definition 31 (Completeness of coalition formation) *If agents' tasks were solvable individually and their coalition task is solvable as well, then coalition formation is complete.*

Proposition 13 *Coalition formation is complete with respect to PD.*

Proof According to Proposition 12 and Definition 31 coalition formation is complete.

Definition 32 *A coalition task is implicitly solved, if no PD steps are required to solve the coalition task.*

This is the case in the following example where we have 2 agents. Agent $\mathcal{A}$ possesses resource A and would like to obtain resource B. Symmetrically, agent $\mathcal{B}$ possesses resource B and would like to obtain resource A. Their tasks are thus respectively $\vdash A \multimap B$ and $\vdash B \multimap A$. After they have formed a coalition, the coalition task would be $\vdash A \otimes B \multimap A \otimes B$. Thus no PD steps are required to solve the coalition task and the task is implicitly solved. Anyway, in order to solve their individual tasks, agents still need to exchange their resources. They do it by following the following operations: $\mathcal{A} \triangleright \mathcal{B} = \{A\}$ and $\mathcal{B} \triangleright \mathcal{A} = \{B\}$.

4.3 The cost of participating in coalitions

In this section we evaluate analytically the cost of participating in coalitions. We show that in the worst case individual task solving outperforms coalition task solving. However, in the best case coalition task solving is more efficient.

Proposition 14 *The computational complexity of solving coalition tasks is in the worst case exponential with respect to solving individual tasks.*

Proof Given that we have n agents with tasks $\vdash A_i \multimap B_i, i = 1 \ldots n$ in a coalition then their collective task would be $\vdash A_1 \otimes \ldots \otimes A_n \multimap B_1 \otimes \ldots \otimes B_n$. This is how collective tasks are formed. Given additionally that for solving an individual task a plan with m steps is required, then the length of collective plan would be $n * m$. Although the branching factor b of search space would be the same for individual and collective tasks, the search space depth d would be n times larger for collective plans. This means exponential growth since the search spaces for solving individual tasks and a coalition task consisting the individual tasks consist respectively of $n * b^d$ and b^{n*d} nodes.

Therefore it makes sense to form coalitions if the following inequality holds: $n * b^d \geq b^{n*d/k}$, where k is a synergy factor. This factor determines how much the length of a coalition task solution shrinks if agents join a coalition. However, in the best case, for solving a coalition task, no PD steps have to be applied at all. This is the case, when the coalition task is solved implicitly.

4.4 Symbolic negotiation

In this section we define the symbolic negotiation process with respect to PD. We define also symbolic negotiation rules for operating over forward (Θ_f) and backward chaining (Θ_b). Additionally some other negotiation-specific definitions are given. Finally, we prove that symbolic negotiation is sound and complete.

Definition 33 *Negotiation rule $\Theta_f(a_k/a_i)$ is defined as the following operational rule*

$$\frac{\vdash ((((I \setminus O(a_i)) \cup I(a_i)) \setminus I(a_k)) \cup O(a_k)) \multimap_{a_1,\ldots,a_{i-1},a_k,a_{i+1},\ldots,a_n} O}{\vdash I \multimap_{a_1,\ldots,a_{i-1},a_i,a_{i+1},\ldots,a_n} O \quad I(a_k) \subseteq I(a_i) \quad O(a_i) \subseteq O(a_k)} \Theta_f(a_k/a_i)$$

Definition 34 *Negotiation rule $\Theta_b(a_k/a_i)$ is defined as the following operational rule*

$$\frac{\vdash I \multimap_{a_1,\ldots,a_{i-1},a_k,a_{i+1},\ldots,a_n} ((((O \setminus I(a_i)) \cup O(a_i)) \setminus O(a_k)) \cup I(a_k))}{\vdash I \multimap_{a_1,\ldots,a_{i-1},a_i,a_{i+1},\ldots,a_n} O \quad I(a_k) \subseteq I(a_i) \quad O(a_i) \subseteq O(a_k)} \Theta_b(a_k/a_i)$$

$I(a)$ and $O(a)$ in the preceding rules represent respectively resources required to apply and achieved by applying capability a. I represents resources available for solving a task while O represents resources, which must be achieved by the task. The preceding negotiation rules replace a PD step application of a_i in a task derivation with an application of a_k. Rule $\Theta_f(a_k/a_i)$ replaces forward chaining steps while $\Theta_b(a_k/a_i)$ replaces backward chaining steps. The symbolic negotiation rules replace a capability in the derivation with another capability which consumed the same amount or less and generated the same amount or more resources than the previous capability. Therefore negotiation rules allow to optimise a task of agent X by agent Y. This means that some PD steps of agent X in its task can be substituted by Y, if associated capabilities are more efficient in terms of resource consumption and generation.

Definition 35 (Symbolic negotiation) *Symbolic negotiation is PD with coalition formation and symbolic negotiation rules.*

Definition 36 (CPS) *Cooperative problem solving is PD with coalition formation.*

Definition 37 (Computational equivalence) *Two processes are computationally equivalent with respect to PD, if they would construct the same set of resultants.*

Theorem 5 *CPS and symbolic negotiation are computationally equivalent.*

Proof The only difference between symbolic negotiation and CPS is the usage of symbolic negotiation rule(s) in symbolic negotiation. These rules, however, only guide the search procedure of PD and do not construct resultants, which would not be achieved by PD. Moreover, given that the same backtracking mechanism is applied with the symbolic negotiation rules as with PD, all resultants of PD would be achieved by symbolic negotiation as well. Therefore symbolic negotiation and CPS are computationally equivalent.

According to Theorem 5 symbolic negotiation could be view as a special case of CPS, where PD is guided at operational level.

Definition 38 *Symbolic negotiation is sound, if it is sound with respect to PD, and all resultants achieved through symbolic negotiation are achievable through CPS.*

Definition 39 *Symbolic negotiation is complete, if it is complete with respect to PD, and all resultants achieved through CPS are achievable through symbolic negotiation.*

Theorem 6 (Soundness of symbolic negotiation) *Symbolic negotiation is sound.*

Proof According to Theorem 5, Proposition 11 and Definition 38 symbolic negotiation is sound.

Theorem 7 (Completeness of symbolic negotiation) *Symbolic negotiation is complete.*

Proof According to Theorem 5, Proposition 13 and Definition 39 symbolic negotiation is complete.

Soundness and completeness of our CPS formalism are directly derived from Proposition 11, Proposition 13 and completeness/soundness results of PD presented in Chapter 3.

4.5 An example of symbolic negotiation

The following example is a scenario from Küngas et al [96]. However, the symbolic negotiation rules were applied there implicitly. Here we apply the rules explicitly.

In that scenario two students, John and Peter, are looking for ways to relax after long days of studying and a final successful examination. John has a CD and he wants to listen to music:

$$G_{John} = \{Music\}.$$

Unfortunately, his CD player is broken and this makes his goal unachievable. John has to visit also a library to return books and this gives him possibility to return also books of other students when this may be useful for him. John has 10 USD for covering

all his expenses, related to relaxing. Considering that he has a broken CD player and a CD, his initial state is as follows:

$$S_{John} = \{Dollar^{10} \otimes CD \otimes BrokenCDPlayer\}$$

and his capabilities are:

$$\Gamma_{John} = \begin{array}{l} \vdash_{John} Books \multimap_{returnBooks} BooksReturned \\ \vdash_{John} CDPlayer \otimes CD \multimap_{playMusic} Music \end{array}$$

Peter is skilled in electronics and can repair the CD player. He has decided to spend his day in a park with his girlfriend. However, he has to return books to the library. Since he has to take a taxi to reach the library, he has to spend 10 USD to cover his transportation expenses. This does not match well with his goals, because of he has only 15 USD while he needs 25 USD for food, drinks and attractions in the park. Therefore he lacks 20 USD to achieve his goals. Peter's initial state, goal and capabilities are described as follows:

$$S_{Peter} = \{Dollar^{15} \otimes Books\},$$

$$G_{Peter} = \{BooksReturned \otimes Beer\},$$

$$\Gamma_{Peter} = \begin{array}{l} \vdash_{Peter} Dollar^{10} \otimes Books \multimap_{returnBooks} BooksReturned \\ \vdash_{Peter} BrokenCDPlayer \multimap_{repairCDPlayer} CDPlayer \\ \vdash_{Peter} Dollar^{25} \multimap_{buyBeer} Beer \end{array}$$

The sets of extra-logical axioms Γ_{John} and Γ_{Peter} represent capabilities of John and Peter, respectively. We write $\vdash_X$ to indicate that a capability is provided by X, $\multimap_Y$ labels a capability with name Y. The internal state of John is described by the following sequent:

$$\Gamma_{John}; Dollar^{10} \otimes CD \otimes BrokenCDPlayer \vdash_{John} Music.$$

This means that John has 10 USD, a CD and a broken CD player. His goal is to listen to music. Peter's state and goal are described by another sequent:

$$\Gamma_{Peter}; Dollar^{15} \otimes Books \vdash_{Peter} BooksReturned \otimes Beer.$$

In the following we write B, BR, BE, CD, P, BP, M and D to denote *Books*, *BooksReturned*, *Beer*, *CD*, *CDPlayer*, *BrokenCDPlayer*, *Music* and *Dollar* respectively. Given John's and Peter's capabilities and internal states, both agents start individually with theorem proving. Initially they fail, since they are unable to reach their goals individually. Then PD in LL is applied to the same set of formulae and new subtasks are derived. These subtasks indicate problems, which could not be solved by agents themselves and need cooperation with other agents. In particular, John has to ask help for solving the following sequent, which is derived by PD:

$$D^{10} \otimes BP \vdash_{John} P.$$

The sequent is produced by applying the backward chaining PD step as follows (to allow shorter proof, we write here $\vdash$ instead of $\vdash_{John}$):

$$\dfrac{\dfrac{D^{10} \otimes BP \vdash P}{D^{10} \otimes BP \otimes CD \vdash CD \otimes P}\ normalise}{D^{10} \otimes BP \otimes CD \vdash_{John} M}\ \mathcal{R}_b(playMusic)$$

where *normalise* is another LL inference figure, which reduces the number of literals in a given sequent:

$$\dfrac{\dfrac{\overline{A \vdash A}\ Id \qquad B \vdash C}{B, A \vdash C \otimes A}\ R\otimes}{B \otimes A \vdash C \otimes A}\ L\otimes$$

Since Peter can repair the CD player, he agrees partially with the proposal. However, because he needs USD 20 for achieving its goals, Peter combines modified John's resultant with his own. The following is John's resultant after Peter extended it:

$$\dfrac{\dfrac{D^{10} \vdash 1}{D^{10} \otimes BP \vdash BP}\ normalise}{D^{10} \otimes BP \vdash P}\ \mathcal{R}_b(repairCDPlayer)$$

Peter's own resultant is the following:

$$\dfrac{\dfrac{\dfrac{\vdash D^{20}}{D^{5} \otimes BR \vdash BR \otimes D^{25}}\ normalise}{D^{15} \otimes B \vdash BR \otimes D^{25}}\ \mathcal{R}_f(returnBooks)}{D^{15} \otimes B \vdash BR \otimes BE}\ \mathcal{R}_b(buyBeer)$$

These 2 tasks are now merged:

$$\dfrac{D^{10} \vdash D^{20}}{1 \vdash D^{20} \quad D^{10} \vdash 1}\ merge$$

By exploring the partial plan of this resultant, John discovers that both his and John's goals would be reached, if he delivers Peters' books to the library and Peter repairs the CD player for 10 USD.

$$\dfrac{\dfrac{\overline{D^{10} \vdash D^{10}}\ Id}{D^{10} \vdash D^{20}}\ \Theta_b(returnBooks_{John}/returnBooks_{Peter})$$

Peter accepts the offer.

4.6 Summary

This chapter formalised CPS and symbolic negotiation. By extending results from Chapter 3, we formalised symbolic negotiation and CPS with respect to partial deduction. We also sketched soundness and completeness proofs for these formalisations.

Additionally we formalised the process of coalition formation and analysed its effect on symbolic negotiation and CPS. The analysis emphasises that coalition formation should be considered with great care. Moreover, coalitions should be avoided in the general case, if agents can solve their problems alone.

Part II

Distributed Semantic Web Service Composition

Chapter 5

Semantic Web Service Representation

In this chapter we describe how we represent Web service operations with extra-logical axioms of LL. LL, as a resource-conscious logic, enables us to capture the essential features of Web services, such as input/output parameters, states and non-functional attributes, formally. After Web service operations have been presented as extra-logical LL axioms, PD or symbolic negotiation can be applied to the encoded domain. Then composite solutions can be extracted from particular PD derivations. A method for extracting Web service compositions from LL proofs has been already proposed by Rao et al [161]. The latter approach is based on results of Milner [136], Abramsky [1], Bellin and Scott [12].

There are several advantages of LL over other formalisms while encoding Web service operations. Since LL is a resource-conscious logic, we can distinguish information transformation and state change effects of Web services. Moreover, we can encode both qualitative and quantitative non-functional attributes of Web services. Because of soundness of our PD formalism, correctness of constructed composite Web services is guaranteed with respect to the initial specifications and no further verification of the composite Web service is required. Symmetrically, completeness of the formalism ensures that a composable solution would be found, if there is any at all.

5.1 Semantic Web services in LL

There are by now several languages and ontologies available for representing Web services in the Semantic Web context, including WSML, WSMO, OWL-S, etc. Furthermore, a mapping from DAML-S to a LL representation has been described by Rao et al [161]. Thus there exist some activities for binding LL and the Semantic Web services. In this section we only describe how the semantics of Web services can be presented in LL. No references to specific representation languages are made. We specify available Web service operations with extra-logical LL axioms and a requested Web service as a LL theorem to be proven.

In general, LL provides us with the following features for presenting useful properties of Web services that are hard or even impossible to present in other formalisms:

1. Propositional LL enables us to present quantities of consumable resources in Web services, such as price, time and the size of cache.

2. The "of course" modality (!) enables us to distinguish two aspects of the service functionalities: information transformation and state change, which is triggered by the execution of a service. Information transformation is presented by input/output parameters of a Web service operation. Since that information is reusable, the input values are not consumed after the execution of a service. Reusability is determined by using the "of course" modality. State change, in contrary, is modelled through resource manipulation: some elements of the initial state are consumed and the new elements are generated. Hence, the state variables are presented by propositions without "of course" modality.

3. LL can be used to represent concurrent processes that are used for modeling composite Web services. In particular, a translation from proofs in LL into Milner's π-calculus [136] has been extensively studied in [1, 12, 135]. For example, multiplicative conjunction ($\otimes$) can represent *composition* in π-calculus while additive disjunction ($\oplus$) can represent *choice*, and the "of course" modality (!) can present *replication*.

Web service operations are described in terms of functionalities and non-functional attributes. The functionalities include inputs, outputs, preconditions, effects and exceptions. The non-functional attributes are classified, according to Rao et al [161], into three categories: consumable quantitative attributes, qualitative constraints and qualitative results. Generally, a required composite Web service can be expressed with the following LL formula

$$(\Gamma_c, \Gamma_v); \Delta_c \vdash ((I \otimes P) \multimap (O \otimes E) \oplus F) \otimes \Delta_r,$$

where both Γ_c and Γ_v are sets of extra-logical axioms representing respectively available *value-added* Web services and *core* Web services. Δ_c is a multiplicative conjunction of non-functional constraints. Δ_r is a multiplicative conjunction of non-functional results. We shall explain these two concepts later.

$I \otimes P \multimap (O \otimes E) \oplus F$ is a functionality description of the required Web service. Both I and O are multiplicative conjunctions of literals with "of course" modality. While I represents a set of input parameters of the service, O represents output parameters returned by the service. P and E are respectively multiplicative conjunctions of preconditions and effects, while F is an additive disjunction representing possible exceptions.

Intuitively, the formula can be explained as follows: given a set of available Web 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 E. If the execution of the required Web service fails, an exception in F is thrown. Every element in Γ_c and Γ_v is in form

$$\Delta_c \vdash ((I \otimes P) \multimap (O \otimes E) \oplus F) \otimes \Delta_r,$$

where meanings of Δ_c, Δ_r, I, P, O, F and E are the same as described above.

5.1.1 Functionalities

There are two kind of functionalities of Web services: information transformation and state change produced by the execution of the service. Information transformation is represented as a transformation from Web service operation's input parameters to its output parameters. State change, however, describes what the Web service actually does and how execution of a Web service operation changes the computational environment. Therefore it is modeled as transformation from the preconditions to the effects.

A typical example for a state change is a Web service for logging into a Web site. Its input parameters are user name (*Username*) and password (*Password*), and its output is a confirmation message (*LoginOk*). After the execution, the state of environment changes from "not logged in" (*NotLoggedIn*) to "logged in" (*LoggedIn*). The latter Web service operation is represented as follows

$$\vdash !Username \otimes !Password \otimes NotLoggedIn \multimap_{login} !LoginOk \otimes LoggedIn$$

5.1.2 Non-functional attributes

Non-functional attributes are useful in evaluating and selecting Web services when there are several services having the same functionalities. In service presentation, the non-functional attributes are specified as results and constraints. We classify the non-functional attributes into the following three categories:

- **Consumable Quantitative Attributes:** these attributes limit the amount of resources that can be consumed by a composite service. For instance, the price of a composite service is the sum of prices for all included existing services. This kind of attribute could represent total cost, total execution time, etc.

- **Qualitative Constraints:** attributes which can not be expressed by quantities are called qualitative attributes. Qualitative constraints are qualitative attributes which specify requirements to Web service execution. For example, some services may respond only to some authorised calls.

- **Qualitative Results:** another kind of qualitative attributes (such as Web service type, service provider or geographical location) , which specify the results regarding services' context.

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

- **Constraints for a service**

$$\Delta_1 = Consumable^x \otimes !Constraint$$

- **Results produced by a service**

$$\Delta_2 = !Fact$$

5.1.3 An example of representation

To illustrate the LL presentation of Web services, let us consider the following example. The example is adapted from Chapter 10 and demonstrates both the usage of functionalities and non-functional attributes. We write *PL*, *SL*, *BR*, *MO*, *HC*, *WK*, *LC*, *LI*, *PU* and *PN* to refer respectively to *PRICE_LIMIT*, *SKILL_LEVEL*, *BRAND*, *MODEL*, *HEIGHT_CM*, *WEIGHT_KG*, *LENGTH_CM*, *LENGTH_IN*, *PRICE_USD* and *PRICE_NOK*.

The available services, both value-added and core services, are specified as follows:

$$\begin{aligned} \Gamma = \; & NOK^{10} \vdash PL \otimes SL \multimap_{selectBrand} BR \\ & \vdash HC \otimes WK \multimap_{selectModel} LC \otimes MO \\ & NOK^{20} \vdash LC \multimap_{cm2inch} LI \\ & CA\_MICORSOFT \vdash PU \multimap_{USD2NOK} PN \\ & \vdash LI \otimes BR \otimes MO \multimap_{selectSki} PU \otimes LOC\_NORWAY \end{aligned}$$

NOK^{10} in the preceding means that 10 NOK are consumed by executing *selectBrand* Web service operation. Web service operation *cm2inch* costs 20 NOK. It is also specified that the currency exchange operation *USD2NOK* only responds to the execution requests which have been certificated by Microsoft. The *selectSki* operation is located in Norway.

The required composite service is specified by the following formula

$$(\Gamma_c, \Gamma_v); \Delta_1 \vdash (PL \otimes SL \otimes HC \otimes WK \multimap PN) \otimes \Delta_2$$

Constraints for the composite service are as follows

$$\begin{aligned} \Delta_1 &= NOK^{35} \otimes !CA\_MICROSOFT \\ \Delta_2 &= !LOC\_NORWAY \end{aligned}$$

The constraints determine that we would like to spend at most 35 NOK for executing the composite service. The composite service consumer requires that all location-aware services are located within Norway (*!LOC_NORWAY*) and it has a certificate

from Microsoft (*!CA_MICROSOFT*). 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 or equal to the number of available resources, the services can be included into composite service. Otherwise, if, for example, the *selectBrand* service would require 20 NOK for execution then the total required amount would be 40 NOK and the composition would not be valid.

For qualitative constraints (for example, location), a 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 a service description, the constraint is not applied to this service at all.

5.2 Mapping WSDL documents to LL

In this section we briefly describe how WSDL documents can be mapped into a LL representation. Indeed, we are using the same method later in Chapter 11 within our empirical research while constructing LL representations of Web service operations.

5.2.1 WSDL structure

The most essential elements, from mapping point of view, in WSDL are portType, operation, message and types. Generally speaking, element portType defines a set of operations, which have input and output messages. While input messages represents operation's input parameters, output messages encapsulate data returned by operations.

Based on the structured information in these XML nodes, LL descriptions can be easily constructed. However, the most crucial aspect is mapping information in message and type definitions. The latter is due to the ambiguity of interpreting this information. For example, if you consider the following WSDL document example, you can see that in type definitions of GeoIP and GetGeoIP, there are elements called respectively IP and IPAddress, which refer to the same concept, but have different names.

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://www.webservicex.net"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://www.webservicex.net"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://www.webservicex.net">
      <s:complexType name="GeoIP">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="IP" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="CountryCode" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="CountryName" type="s:string" />
```

```
      </s:sequence>
    </s:complexType>
    <s:element name="GetGeoIP">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="IPAddress" type="s:string" />
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="GetGeoIPResponse">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="GetGeoIPResult" type="tns:GeoIP" />
        </s:sequence>
      </s:complexType>
    </s:element>
  </s:schema>
</wsdl:types>

<wsdl:message name="GetGeoIPSoapIn">
  <wsdl:part name="parameters" element="tns:GetGeoIP" />
</wsdl:message>
<wsdl:message name="GetGeoIPSoapOut">
  <wsdl:part name="parameters" element="tns:GetGeoIPResponse" />
</wsdl:message>

<wsdl:portType name="GeoIPServiceSoap">
  <wsdl:operation name="GetGeoIP">
    <wsdl:input message="tns:GetGeoIPSoapIn" />
    <wsdl:output message="tns:GetGeoIPSoapOut" />
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="GeoIPServiceSoap" type="tns:GeoIPServiceSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
  <wsdl:operation name="GetGeoIP">
    <soap:operation soapAction="http://www.webservicex.net/GetGeoIP" style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>

<wsdl:service name="GeoIPService">
  <wsdl:port name="GeoIPServiceSoap" binding="tns:GeoIPServiceSoap">
    <soap:address location="http://www.webservicex.net/geoipservice.asmx" />
  </wsdl:port>
</wsdl:service>

</wsdl:definitions>
```

Thus there are cases where it is not possible to automatically map Web service operations in a WSDL document into knowledge-richer representation in LL. Therefore mapping from WSDL to LL is currently done mostly manually. See Chapter 11 for more detailed discussion in this matter.

5.2.2 From WSDL to LL

Generally, Web service operations in WSDL documents can be encoded as follows:

$$\vdash input\_msg \multimap output\_msg$$

For instance, the WSDL document in Section 5.2.1 can be represented as the following LL specification

$$\vdash IPAddress \multimap_{GetGeoIP} CountryName \otimes ISO3166CountryCode \otimes IPAddress,$$

where *CountryName*, *ISO3166CountryCode* and *IPAddress* refer to particular concepts in an ontology. The LL specification contains a single operation. As one can see we have manually renamed some elements in the WSDL to map them into our ontology—CountryCode to ISO3166CountryCode and IP to IPAddress.

The preceding encoding could be seen as an optimised version of the encoding. An alternative encoding, which represents the structure of the operation in a more detailed way, is the following

$$\vdash GetGeoIP \multimap_{GeoIPServiceSoap\_GetGeoIP} GetGeoIPResponse$$

$$\vdash IPAddress \multimap GetGeoIP$$

$$\vdash GetGeoIPResponse \multimap GeoIP$$

$$\vdash GeoIP \multimap CountryName \otimes ISO3166CountryCode \otimes IPAddress$$

The encoding represents more closely the structure of data types defined in the WSDL document. However, since the encoding includes more computability statements, it is computationally more demanding at composition time.

5.3 Summary

In this chapter we explained how to represent Web services as LL formulae. We presented a general LL sequent structure which can encapsulate most of the properties related to Web services. Moreover, we proposed some guidelines on how to transform Web service operations in WSDL documents into the LL form. After Web service operations have been encoded in LL, PD or symbolic negotiation can be applied for automated Web service composition.

Chapter 6

Agent System Architecture

This chapter presents an architecture and a methodology for agent-based Web service discovery and composition. We assume that Web services are described with declarative specifications like OWL-S or WSML documents. Based on the declarative information about particular Web services, symbolic negotiation is applied for dynamic Web service discovery and composition.

Symbolic negotiation, as we defined earlier, is a mixture of distributed planning and information exchange. Therefore, by using symbolic negotiation for automated service composition, we support information collection and integration during service composition. The latter aspect has been largely neglected in automated service composition until now.

Several articles address automatic composition of Web services [134,172,184,200]. However, they all require existence of a central directory of Web service specifications, which contrasts largely to the dynamic nature of the Web. In the Web the set of available Web services changes rapidly—new Web services are created, old ones are modified or removed. Keeping track of all these changes is a huge burden for a centralised directory. Some essential issues in decentralised Web service provision have been addressed by Papazoglou et al [148].

Another disadvantage of centralised approaches is that it only allows service requesters to locate services, while service providers lack an ability to attract potential customers. The agent-based architecture, we propose here, gives service providers a more proactive role in the service composition process. Our service provision architecture is based on the multi-agent system AGORA [127], which provides an infrastructure, where service providers and requesters can meet each-other's goals.

We propose that symbolic agent negotiation could be used as a mechanism for discovering available Web services and composing new ones automatically. If no service, satisfying user's requirements, is found, symbolic negotiation between agents is initiated and a new composite Web service is constructed dynamically.

6.1 The symbolic negotiation process

In our applications we consider mainly self-interested cooperative agents. This means that agents cooperate with each other as long as it does not prevent them achieving their own goals. To understand how agents are supposed to apply symbolic negotiation during problem solving, we present in this section our general symbolic negotiation model.

We define messages as instances of the following structure:

$$(id_{req}, S, R, T),$$

where id_{req}, S, R and T denote respectively a message identifier, its sender, its receiver and a task itself in a declarative language. The message identifier is needed to keep track of different symbolic negotiation threads. The sender and the receiver are identifiers of participating agents and the offer content is represented with a LL formula.

While naming message identifiers, we apply the following conventions:

- if agent $\mathcal{A}$ sends out an initial task, then a is its message identifiers name
- if a represents a task, then a' represents another task derived from it
- if an agent $\mathcal{A}$ sends out more than one task, then their messages are indexed as $a_1, a_2, \ldots$

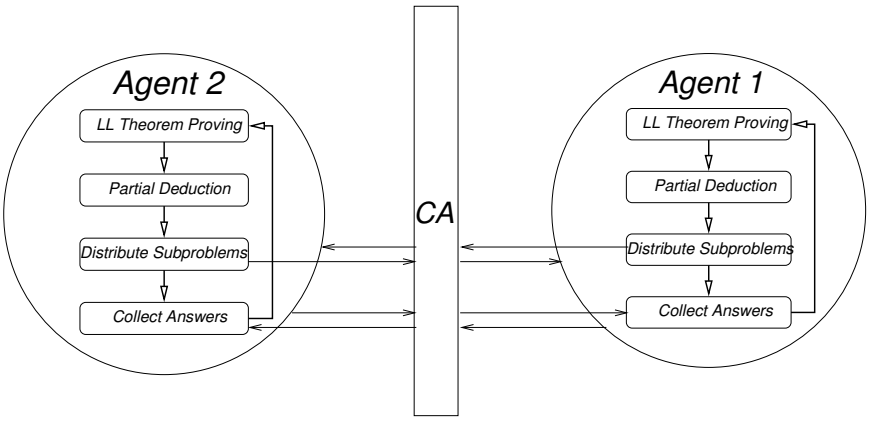


Figure 6.1: General symbolic negotiation model.

Our general symbolic negotiation model is presented in Figure 6.1. In this model each agent initially tries to solve its problem alone. If the agent cannot find a solution then subproblems are generated. The subproblems are distributed among the partners and they are treated as subtasks to other agents. In other words, they present what

an agent can provide and what it expects to get in return. If it should happen that an agent dies during symbolic negotiation, then possibly some negotiation steps should be cancelled or revised. *CA* in Figure 6.1 denotes to Communication Adapter, which facilitates agent communication and discovery.

6.2 The architecture

The AGORA multi-agent environment [127] was developed with the intention to support cooperative work between agents. The system consists of 2 types of components (nodes)—agents and agoras (see Figure 6.2). Agoras are cooperative nodes which facilitate agent communication, coordination and negotiation. Moreover, agoras encapsulate a method for (partial) service composition. An agora node contains *default agents* and *registered agents*. In our scenario, the default agents are Agora Manager and Negotiator. Agora Manager implements general agora functions, such as service matchmaking and agent/service registration, while Negotiator applies symbolic negotiation.

Service matchmaking basically involves finding an atomic service, which satisfies agent requirements for a service. Negotiator applies symbolic negotiation for composing new services automatically. Negotiation is applied, if Agora Manager fails to find an atomic service satisfying agents' requirements.

Service provider agents register their services at specific agoras according to their service domains. For example agents providing services for selling, buying and managing hardware register themselves and available services at agoras devoted to hardware. Specific agoras may represent also coalitions of service providers.

Service requester agents, however, register requests for services at the central agora. Then the central agora negotiates with specific agoras to find services satisfying requester agents' requirements. The central agora also mediates information exchange between different requester agents. Moreover, the central agora might also form requester agent coalitions, if it is needed for certain services. It may happen for instance that a (composite) service requires input from more that one service requester agent.

A specific service composition architecture can be specified through refinement and instantiation of the generic architecture described above. An instance of the generic architecture is elaborated in Section 6.4 and depicted in Figure 6.4.

6.3 Inter-agent communication protocol

Figure 6.3 presents the proposed interaction protocol in Agent UML notation. This protocol uses the FIPA (The Foundation for Intelligent Physical Agents) reserved communicative acts, allowing thus interoperability with other FIPA compliant agent systems. The agent interaction process is summarised in the following.

The central agora is a search control point for requester agents' queries. Specific agoras are cooperative nodes gathering agents who provide the same sort of services.

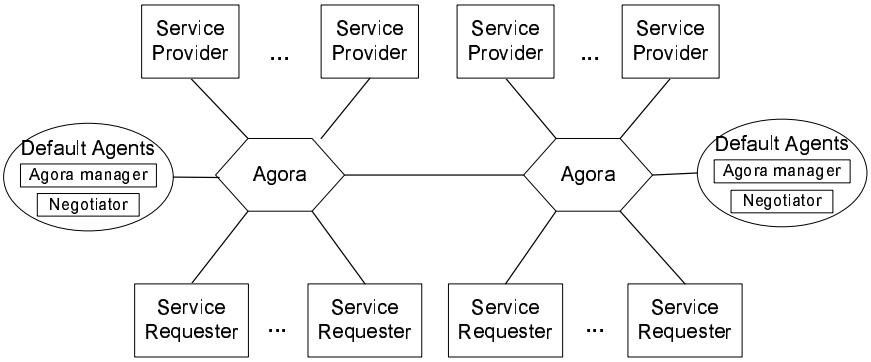


Figure 6.2: Multi-agent system architecture.

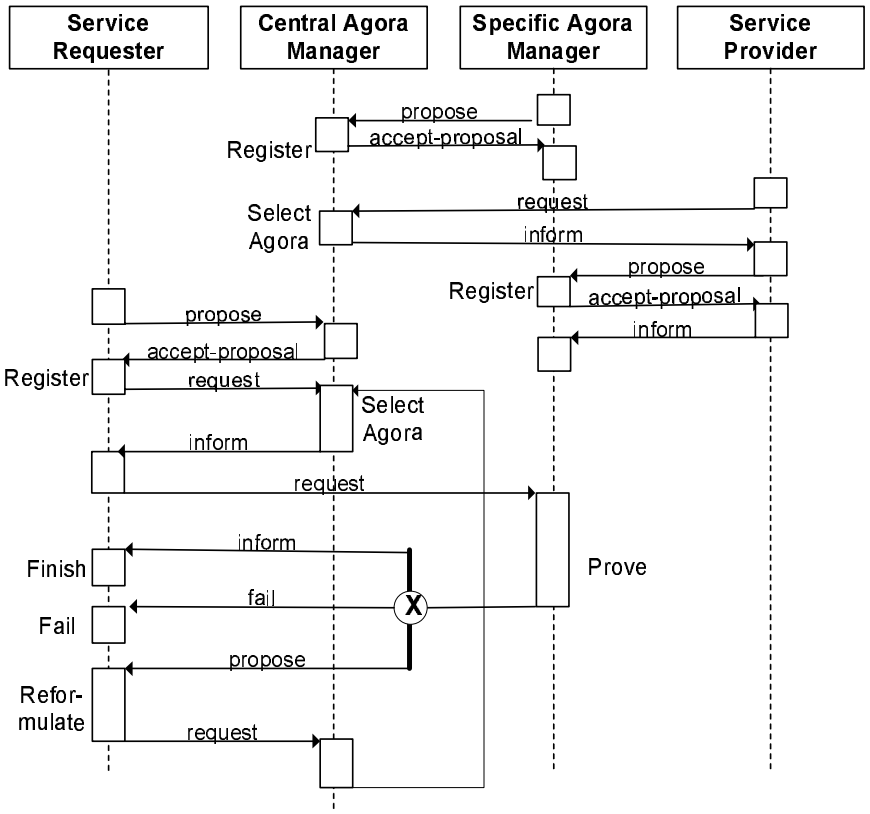


Figure 6.3: Agent interaction protocol.

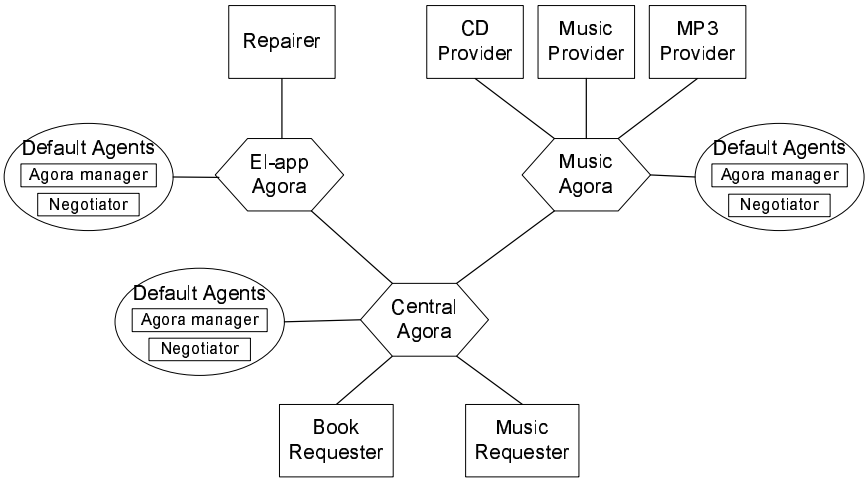


Figure 6.4: The example architecture.

Specific agoras register themselves to the central agora. After receiving the registration request from a service provider agent, the central agora locates a specific agora where the service provider could register itself. In our case, the service providers registered to the same specific agora provide services in the same domain.

Service requester agents register themselves to the central agora. Additionally they publish their requirements in a declarative language. After that the central agora tries to locate an atomic service satisfying the requirements. However, if no suitable atomic service is found, symbolic negotiation is initiated. During symbolic negotiation the central agora contacts specific agoras to receive a composite service for satisfying the particular requirements.

It might happen that services from different agoras are needed for a composition. In that case the central agora receives a partial composition from one specific agora and forwards it to another specific agora for further composition. The process is called symbolic negotiation—the central agora negotiates with specific agoras to compose a solution for a service requester agent. Finally a composite service is constructed and returned to the requester agent, who initiated the symbolic negotiation.

6.4 An example

In this section we demonstrate the usage of symbolic negotiation in a distributed problem solving scenario. It should be noted that distributed Web service composition is an instance of distributed problem solving. Hence the example presented here explains also how we are going to apply distributed Web service composition.

The particular system architecture and its components are depicted in Figure 6.4. In our scenario we have two requester agents—$\mathcal{R}_1$ and $\mathcal{R}_2$. The goal of $\mathcal{R}_1$ is to listen a

music (*Music*). Initially $\mathcal{R}_1$ has a book (*Book*), a broken CD player (*BrokenCDPlayer*) and 5 dollars ($Dollar^5$). Goals, resources and capabilities of $\mathcal{R}_1$ are described in LL with the following formulae.

$$G_{\mathcal{R}_1} = \{Music\}, \qquad S_{\mathcal{R}_1} = \{Book \otimes BrokenCDPlayer \otimes Dollar^5\}, \qquad \Gamma_{\mathcal{R}_1} = \emptyset.$$

Another query agent $\mathcal{R}_2$ is looking for a book (*Book*) and is in possession of 10 dollars ($Dollar^{10}$). Goals, resources and capabilities of the query agent $\mathcal{R}_2$ are described in LL with the following formulae.

$$G_{\mathcal{R}_2} = \{Book\}, \qquad S_{\mathcal{R}_2} = \{Dollar^{10}\}, \qquad \Gamma_{\mathcal{R}_2} = \emptyset.$$

In addition we have several service provider agents. However, since they published their services through particular agoras and these agoras take care of advertising the services, we do not list here the internal states of the service provider agents. Instead we present the internal states of agoras. Although agoras may have their own resources and goals, which determine their policies, we consider here only a simple case, where agoras take advantage of registered services/capabilities only.

According to the literals service providers can "produce", the providers are aggregated into two agoras—one for music and another for electrical appliances. In the Music Agora $\mathcal{M}$, three agents, *MusicProvider*, *CDProvider* and *MP3Provider*, can provide services related to music. Services *playCD* and *playMP3* provide respectively knowledge about requirements for playing CD-s and MP3-s. Services *buyMP3* and *buyCD* provide means for ordering particular music media.

$$\Gamma_{\mathcal{M}} = \begin{array}{l} \vdash_{MusicProvider} CD \otimes CDPlayer \multimap_{playCD} Music, \\ \vdash_{MusicProvider} MP3 \otimes MP3Player \multimap_{playMP3} Music, \\ \vdash_{CDProvider} Dollar^5 \multimap_{buyCD} CD, \\ \vdash_{MP3Provider} Dollar^3 \multimap_{buyMP3} MP3. \end{array}$$

The agora $\mathcal{E}$ is an aggregation of the agents who can provide electrical appliances. It only advertises one service *repair* from agent *Repairer*. The service description declares that the agent can repair a CD player by charging 10 dollars.

$$\Gamma_{\mathcal{E}} = \vdash_{Repairer} Dollar^{10} \otimes BrokenCDPlayer \multimap_{repair} CDPlayer.$$

Let us look now how symbolic negotiation is applied for constructing dynamically services, which satisfy users' goals. Initially the query agent $\mathcal{R}_1$ sends out a query to agora $\mathcal{M}$ for finding a service satisfying its requirements:

$$(o_1, \mathcal{R}_1, \mathcal{M}, Book \otimes BrokenCDPlayer \otimes Dollar^5 \vdash Music).$$

The query would be satisfied by a service

$$\vdash Book \otimes BrokenCDPlayer \otimes Dollar^5 \multimap Music.$$

Unfortunately the service requirement is too specific and no matching Web service is found. However, agora $\mathcal{M}$ modifies the received offer and sends back the following offers:

$$(o_2, \mathcal{M}, \mathcal{R}_1, Book \otimes BrokenCDPlayer \vdash CDPlayer)$$

and

$$(o_3, \mathcal{M}, \mathcal{R}_1, Book \otimes BrokenCDPlayer \otimes Dollar^2 \vdash MP3Player),$$

which were deduced through PD in the following way:

$$\dfrac{\dfrac{\dfrac{Book \otimes BrokenCDPlayer \vdash CDPlayer}{Book \otimes BrokenCDPlayer \otimes Dollar^5 \vdash Dollar^5 \otimes CDPlayer}\ normalise}{Book \otimes BrokenCDPlayer \otimes Dollar^5 \vdash CD \otimes CDPlayer}\ \mathcal{R}_b(buyCD)}{Book \otimes BrokenCDPlayer \otimes Dollar^5 \vdash Music}\ \mathcal{R}_b(playCD)$$

$$\dfrac{\dfrac{\dfrac{Book \otimes BrokenCDPlayer \otimes Dollar^2 \vdash MP3Player}{Book \otimes BrokenCDPlayer \otimes Dollar^5 \vdash Dollar^3 \otimes MP3Player}\ normalise}{Book \otimes BrokenCDPlayer \otimes Dollar^5 \vdash MP3 \otimes MP3Player}\ \mathcal{R}_b(buyMP3)}{Book \otimes BrokenCDPlayer \otimes Dollar^5 \vdash Music}\ \mathcal{R}_b(playMP3)$$

where *normalise* is another LL inference figure, which reduces the number of literals from a particular sequent:

$$\dfrac{\dfrac{\dfrac{}{A \vdash A}\ Id \quad \dfrac{}{B \vdash C}}{B, A \vdash C \otimes A}\ R\otimes}{B \otimes A \vdash C \otimes A}\ L\otimes$$

Agent $\mathcal{R}_1$ chooses the offer o_2 and forwards it to the electrical appliance agora $\mathcal{E}$:

$$(o_4, \mathcal{R}_1, \mathcal{E}, Book \otimes BrokenCDPlayer \vdash CDPlayer).$$

Agora $\mathcal{E}$ modifies the offer further and ends up with the following counteroffer:

$$(o_5, \mathcal{E}, \mathcal{R}_1, Book \vdash Dollar^{10}),$$

which was derived in the following way:

$$\dfrac{\dfrac{Book \vdash Dollar^{10}}{Book \otimes BrokenCDPlayer \vdash Dollar^{10} \otimes BrokenCDPlayer}\ normalise}{Book \otimes BrokenCDPlayer \vdash CDPlayer}\ \mathcal{R}_b(repair)$$

Since no service provider can produce the *Dollar* literal, the message is sent to the central Agora, which has an overview of requester agents' requirements. Fortunately, it turns out that agents $\mathcal{R}_1$ and $\mathcal{R}_2$ can satisfy mutually their requirements such that requester $\mathcal{R}_1$ gets 10 dollars and requester $\mathcal{R}_2$ gets a book. The resulting service composition can be translated to a process description languages like BPEL4WS.

6.5 Summary

This chapter described an agent architecture for distributed automated composition of Web services. Agent-specific aspects provide Web service composition with proactivity, reactivity, social ability and autonomy, while usage of FIPA ACL and application domain specific ontologies provide a standardised medium for Web service deployment.

We assume that Web services are represented in a OWL-S or WSML like declarative language. Then these descriptions can be translated to LL formulae for internal agent reasoning. A possible mapping mechanism for DAML-S is presented in [161]. Given such representation, agents can employ symbolic agent negotiation, as proposed in Chapter 4, for composing new Web services according to their requirements. This approach leads to distributed composition of Web services.

Chapter 7

P2P-Based MAS

The increasing popularity of P2P systems (such as Overnet, Kazaa and Gnutella) for file sharing, indicates general interest in resource sharing. However the current P2P systems suffer at least from two drawbacks. First, they are mostly designed for sharing either data or CPU power, but not both in the same system. Moreover, in the case of CPU sharing, the executable computational processes are expected to be known *a priori* for each participant (like in SETI@Home). Second, the current P2P network nodes still lack a degree of proactivity, which would provide higher degree of autonomy, rationality and fairness.

In contrary, multi agent systems (MAS) still seem to lack enough capabilities to reorganise themselves in dynamic environments. In particular, despite of the intelligent behaviour assigned to agents, MAS architectures are currently mostly designed manually. Therefore combining MAS-s and P2P networks would extend the capabilities of both architectures.

Recently many articles, related to automated composition of (Semantic) Web services [134,200], agent technologies and P2P networks [2,11,27,37,137] (see Chapter 2 for a review) have been published. Although many of them [9, 13, 47, 106, 147] discuss a combined approach, to the best of our knowledge there are currently no systems available, which apply agent technologies to distributed composition of Semantic Web services over structured P2P networks.

Our goal is to construct a system, which would allow users to seamlessly integrate the available Web services and to support data exchange. Emergence of the Semantic Web has resulted in a uniform view to data and computational resources. Describing both data and Web services as semantic objects allows to move from data sharing and service sharing to *resource* sharing in a unified infrastructure. We are going to exploit this unified view while discovering required resources, data and Web services, in our system.

In this chapter we describe an implementation of a MAS where agents cooperatively apply distributed symbolic reasoning for discovering and composing Semantic Web services. A structured P2P network is used to self-organise MAS infrastructure for efficient resource discovery.

Using Web service descriptions in a semantic-rich language significantly increases the amount of semantic information available for discovering requested services. In addition, if no services satisfying user requirements are found, then cooperative problem solving (CPS) or symbolic negotiation is applied for dynamic construction of new composite Web services. The general structure of our system, supporting Semantic Web services composition, is depicted in Figure 7.1.

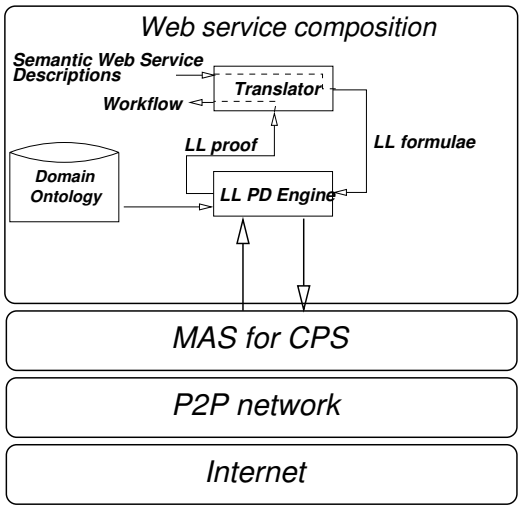


Figure 7.1: The P2P system architecture.

Our Web service composition process reads in available atomic Semantic Web services and the requested one, transforms it into Linear Logic (LL) formulae and applies Partial Deduction (PD) to find (partial) solutions for a request. During PD attached domain ontologies are used to reason over the semantics of Web services' inputs and outputs. Partial solutions can be extended through our symbolic negotiation framework until a complete solution has been found. Complete solutions are returned to the requester.

This approach allows exploitation of Web services in a MAS, which is expected to be distributed over the Internet. The usage of agent technologies allows us to take advantage of agent communication languages, which are well-suited for delivering semantic information. Additional agent techniques could be used as means for controlling access to Web services and other resources that agents possess.

7.1 Distributed composition with P2P

In Chapter 6 we proposed a mediator-based architecture for automated distributed Web service composition. However, we did not explain how Web service provider agents

select mediators where to register themselves and how new mediator agents are selected and organised. In this chapter we extend the former framework and show how new mediators are selected and organised in P2P manner. We also show that P2P approach gives greater scalability of the system with many nodes compared to a manual approach of setting up mediators.

In order to understand better how we map Semantic Web services, agents' goals into a P2P network, let us consider a scenario, where we have two agents—a traveller ($\mathcal{T}$) and a flight company ($\mathcal{F}$). The scenario would we used throughout this chapter. Let S, G and Γ denote respectively available resources, goals and capabilities (Semantic Web services) of agents. Available resources and goals represent respectively the inputs and outputs of a (composite) Web service, which is required by an agent.

The goal of $\mathcal{T}$ is to make a booking (*Booking*) for a specific itinerary. Initially $\mathcal{T}$ knows only its starting (*From*) and final (*To*) location. Additionally the agent has 2 local Web services running, *findSchedule* and *getPassword*, for finding a schedule (*Schedule*) for a journey and retrieving a password (*Password*) from its internal database for a particular Web site (*Site*).

Goals, resources and capabilities of agent $\mathcal{T}$ are described in LL with the following formulae.

$$G_{\mathcal{T}} = \{Booking\}, S_{\mathcal{T}} = \{From \otimes To\},$$

$$\Gamma_{\mathcal{T}} = \begin{array}{l} \vdash From \otimes To \multimap_{findSchedule} Schedule, \\ \vdash Site \multimap_{getPassword} Password. \end{array}$$

For booking tickets, traveller agent $\mathcal{T}$ should contact an airline company. The airline company $\mathcal{F}$ does not have any explicit declarative goals (which is common for companies). The only fact that $\mathcal{F}$ exposes, is a reference to the company Web site (*Site*). Since the fact is unbounded it can be delivered to customers any number of times (this is denoted by ! in the example).

Agent $\mathcal{F}$ has 2 local Web services running—*bookFlight* for booking a flight, and *login* for identifying customers and creating secure channels for information transfer. We assume that a customer has created a personal profile at the airline company including customer's credit card information. Therefore the customer does not have to provide this information explicitly. Goals, resources and capabilities of the airline company $\mathcal{F}$ are described in LL with the following formulae.

$$G_{\mathcal{F}} = \{1\}, S_{\mathcal{F}} = \{!Site\},$$

$$\Gamma_{\mathcal{F}} = \begin{array}{l} \vdash SecureChannel \otimes Schedule \multimap_{bookFlight} Booking, \\ \vdash Password \multimap_{login} SecureChannel. \end{array}$$

7.2 P2P network layer

Since we assumed that in our application we can map each resource to an integer key, we chose to take advantage of a structured P2P network, namely Chord [176]. Structured P2P systems provide scalable resource-location mechanism compared to non-structured networks, where the network is flooded with messages in order to locate resources.

Generally, the Chord protocol consist of a consistent hashing function to provide unique key assignments for each node/object in the network. With the key's value each node can determine its logical position in the system. In Chord the logical position of a node is a point in a circular key space. For example, Figure 7.2 presents an instance of a Chord network topology for a key space with length 32. Black dots represent nodes in the network and white dots represent keys that are not used.

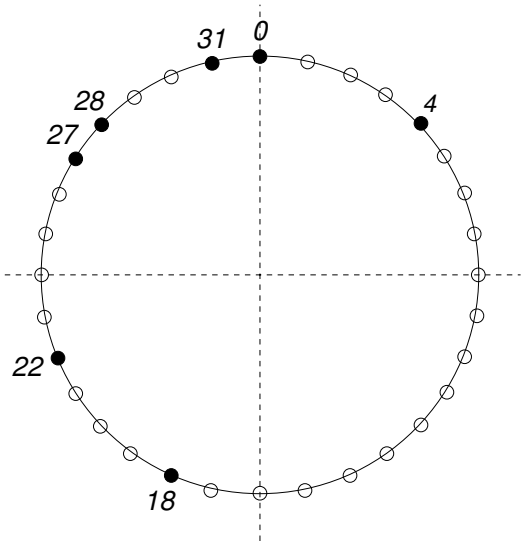


Figure 7.2: A Chord network example.

In order to maintain the ring structure of the network, each node constantly updates its predecessor and successor nodes in the network. These are the nodes which immediately precede or succeed, respectively, a node in the circular key space. As long as predecessors and successors of all nodes are updated, nodes are guaranteed to be found in the network. Thus if one node has to locate a peer holding a particular key, a message could be sent either to its successors or predecessors in the circular key space until it reaches the correct location. This process is made more efficient by using routing tables, which allow to bypass many nodes at once when forwarding a message to its destination.

Each node in the network maintains its personal routing table with N records for

Table 7.1: Routing table of node 27 in Figure 7.2.

Index	Key	Node
1	$27 + 2^0 = 28$	28
2	$27 + 2^1 = 29$	31
3	$27 + 2^2 = 31$	31
4	$27 + 2^3 = 3$	4
5	$27 + 2^4 = 11$	18

2^N key space. Each record points to a successor of a key, which is at distance of $2^i, i = 0 \ldots N - 1$ from the key, which identifies the node. A routing table of node 27 of Chord network in Figure 7.2 is represented for example in Table 7.1. Now, if peer with key 27 wants to deliver a message to the peer with key 22, then according to this routing table the message would be sent initially to a peer with key 18 (the peer with a closest preceding key to 22) and the peer with key 18 would forward the message further.

In order to apply Chord network and its object location mechanism for Semantic Web service composition, we have to implement a mapping from objects to indices. In this article we consider objects to be the names of inputs and outputs of Semantic Web services. Additionally we assume that agents share the same ontology. Thus we can just apply a hash function from an object name to an integer key such that the objects with the same intended meaning would have the same key.

However, in large P2P networks different agents tend to use different ontologies. Therefore we recognise the need for a function, which would transform concepts from different ontologies, but with the same meaning, to the same key or to similar keys. One way to overcome this problem might be to annotate all concepts with sets of keywords. Then *Latent Semantic Indexing* or some other information retrieval algorithm [155] could be applied for computing a unique value to a particular set of keys.

Alternatively, if objects have been annotated with keywords, Hilbert space filling curves (SFC) could be applied for mapping an n-dimensional keyword space to 1-dimensional hash value space. The latter approach has been used by Schmidt and Parashar [168] for locating Web services at Chord P2P network. Unfortunately we could not apply their results directly in our system, since Schmidt and Parashar described Web service *classification* with keywords, while we need to annotate the *inputs* and *outputs* of Web services.

7.3 Integrating P2P into MAS

Our MAS architecture is designed as a layer on top of Chord P2P network. While P2P handles issues related to indexing and efficient location of resources, agents initiate these actions in P2P networks. In our case agents use the P2P network for discover-

ing other agents, whose Web service descriptions include particular literals (names of inputs and outputs). Thus our MAS could be seen as an application layer of a P2P network, whereas P2P network is just another communication medium for MAS.

In order to facilitate efficient location of related agents, one agent per each literal is designated to mediate access to other agents interested in particular literals. Since an agent specification usually includes more than one literal, a single agent may mediate several keys. When an agent joins the network, it first determines whether there are already agents mediating some of its keys. If there is no mediator for particular keys, the agent joins the network as a mediator for these keys. In the case there exists a key mediator, the agent registers itself at the particular mediator. Mediators are organised according to Chord algorithm.

A mediator could be seen as a kind of super-peer, which facilitates communication between agents sharing a particular key. In order to apply Chord P2P network for our purposes, literals in Web service specifications are transformed into integer keys, where a key is the result of the mapping from a literal (concept name). An instance of our network topology is presented graphically in Figure 7.3. The inner circle there represents mediators in the Chord network while auxiliary nodes represent mediated agents.

If an agent has to send a task to other agents, then literals in the task are identified, transformed to keys and the task would be delivered to mediators taking care of particular keys. Then these mediators shall multicast the message to agents, which are registered at these mediators. If the mediated agents would like to deliver a message to other mediators, then they first send a message to their mediator and this mediator shall forward the message to other mediators. If an agent has registered itself at several mediators, then the messages would be sent to the most preceding mediator of a particular key. In this case multicast is implemented on top of a P2P network. If a mediator considers leaving the system, then it delegates its tasks to one of the agents which is registered at this mediator. However, if a mediator does not have any agents registered at itself, then the key disappears from the network.

Since the network of agents and the set of literals is constantly evolving, we would not be able to manually set up mediators, unless we would designate a single agent for mediating all others. The manual approach could work in small or static systems but not in large and dynamic ones.

One disadvantage of P2P networks is that extra efforts are needed to keep them stable and consistent. A Chord network is defined to be stable if successors and predecessors of all nodes are correct. If a network is not stable, then it may break into clusters and there is no guarantee anymore that required resources will be located.

In order to keep the network stable, we would still like to preserve some degree of centralisation in future. Namely, we envisage that there are entities, which monitor the evolution of the network and try to detect and resolve anomalies. Anyway, indexing and search would be still organised in the distributed manner.

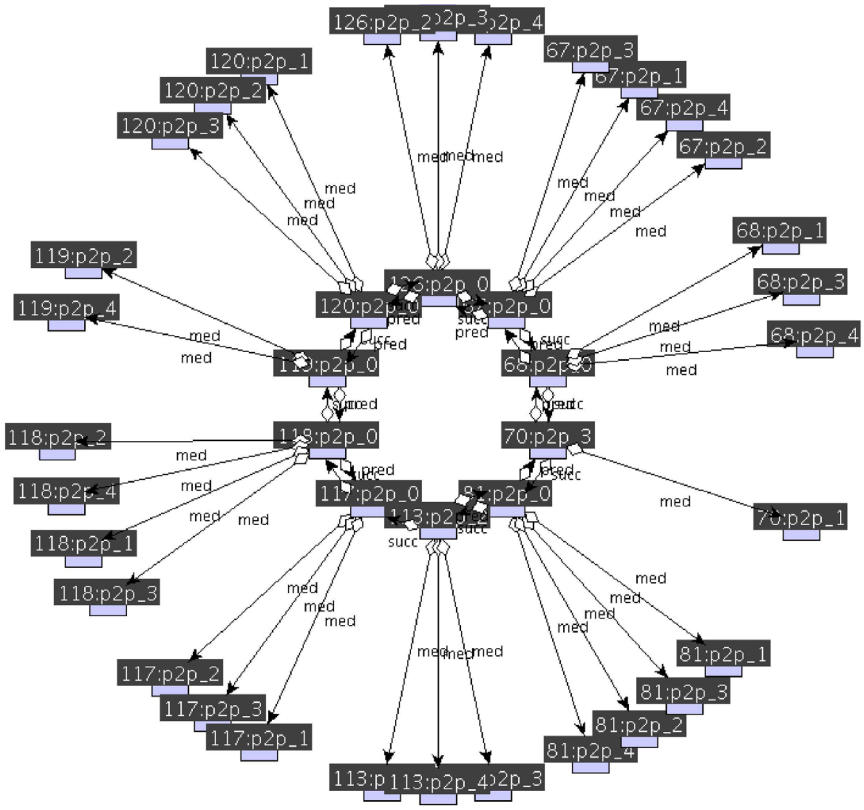


Figure 7.3: Example P2P network topology.

7.4 Elaboration of the example

Let us consider again the agent/service specifications from Section 7.1. The specifications of agents $\mathcal{T}$ and $\mathcal{F}$ form a domain, which consists of 7 literals—*From*, *To*, *SecureChannel*, *Booking*, *Schedule*, *Site* and *Password*. The 4 last literals are shared by both agents. This means that they shall compete for the right to mediate these literals.

Let us assume that these 7 literals are mapped to keys 0, 4, 18, 22, 27, 28 and 31, respectively. To demonstrate the interaction between agent- and P2P-related concepts we additionally assume that agent $\mathcal{T}$ would mediate keys 0, 4, 22 and 28, while $\mathcal{F}$ would mediate 18, 27 and 31. This configuration is summarised in Table 7.2.

Given its specification, agent $\mathcal{T}$ derives and sends out the following task (see [106] for how this and the following tasks were derived):

$$\mathit{Schedule} \vdash \mathit{Booking}.$$

This task would be sent to mediators of literals *Schedule* and *Booking*, which are Chord nodes 27 and 22. These mediators would start solving the task and also multicast

Table 7.2: Keys and mediators of literals.

Literal	Key	Mediator
From	0	$\mathcal{T}$
To	4	$\mathcal{T}$
SecureChannel	18	$\mathcal{F}$
Booking	22	$\mathcal{T}$
Schedule	27	$\mathcal{F}$
Site	28	$\mathcal{T}$
Password	31	$\mathcal{F}$

the task to registered agents. Since we have currently only 2 agents in the network, then the message would be sent only to agent $\mathcal{F}$.

Agent $\mathcal{F}$ merges the task with its current state $!Site \vdash 1$ and as a result achieves task $!Site \otimes Schedule \vdash Booking$. Since $\mathcal{F}$ cannot satisfy the proposal, it derives a new task and forwards it to agent $\mathcal{T}$:

$$Site \vdash Password.$$

Agent $\mathcal{T}$ deduces the task further and constructs the final composite Web service. Thereby $\mathcal{T}$ produces, with help of $\mathcal{F}$, a composite service, whose execution achieves the goal of agent $\mathcal{T}$. The resulted composite service is graphically represented in Figure 7.4. The service composition is finally translated to a process description languages like OWL-S process model or BPEL4WS. The exact translation process is described in [160].

7.5 Empirical and analytical evaluation

In order to evaluate our architecture and the CPS method, we chose to measure the number of messages, which were sent by agents until all agents solved their problems (each agent had to compose a Web service). We considered 4 different methods for message distribution:

1. multicast—each agent delivers its messages through a mediator to agents, whose domain includes any of the literals in a derived partial solution
2. broadcast—each agent delivers its messages to all other agents in the system
3. simple P2P—before delivering each message, the mediators of potentially interested agents are located and then the message is delivered to them
4. P2P with caching—the same as simple P2P with the only difference that the location of each mediator is discovered only once per runtime and is cached for further use

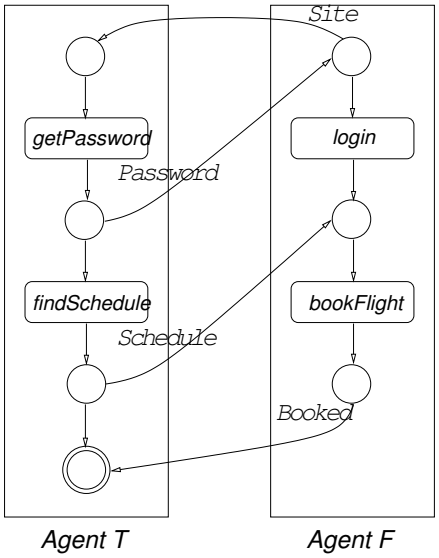


Figure 7.4: The composite Web service.

While data for multicast and broadcast was acquired through experiments, the results for the P2P versions are estimated analytically. For analytical evaluation we assumed that our P2P architecture performs equally with the mediator-based agent architecture with a difference that extra messages should be sent to discover particular mediators. Additionally we assumed that our key space is 1024 to accommodate 1000 concepts. This implies that in order to discover a mediator, generally $log_2 1024 = 10$ messages should be sent in Chord network. Therefore, to evaluate the maximum cost of P2P, we multiplied the number of messages, exchanged during multicast, by 10. However, if we assume that each peer applies caching, then we could use a function $max(N * A + m, p)$ to evaluate the message burden. N, A, m, p in the formula represent respectively the numbers of concepts, agents, multicast messages and worst case P2P communication messages sent. This function reflects that in the worst case each peer has to discover and cache the locations of all keys/concepts in the systems. We do not consider the number of stabilisation messages, while evaluating the cost of using P2P.

Experiments with multicast and broadcast were performed with 10, 20, 50 and 100 agents. With each set of agents the same set of service and task specifications was used with both broadcast and multicast. We ran each experiment with each set of agents 5 times. The overall domain, where the names of services' inputs and outputs were randomly selected, consisted of 1000 concepts. We made experiments with 2 configurations:

1. each agent published 4 Web services, a required composite Web service consisted of at least 3 Web services

2. each agent published 5 Web services, a required composite Web service consisted of at least 5 Web services

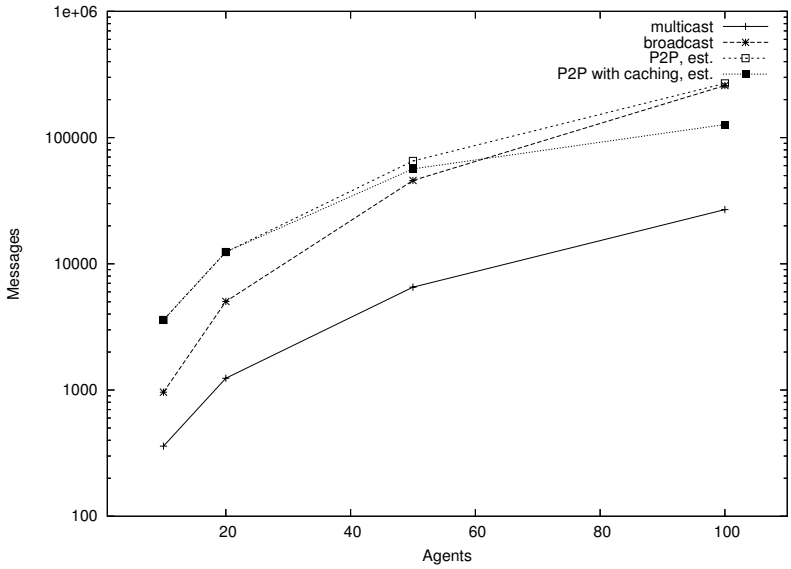


Figure 7.5: Minimum solution length 3.

The results of configuration 1 and 2 are respectively summarised in Figure 7.5 and Figure 7.6. Both figures show how many messages in average were sent during problem solving, before all agents found their solutions. Figure 7.7 demonstrates the exponential complexity of the problem solving methodology (with 10 agents), which is bound to the complexity of LL. Although there exist logics with polynomial complexity [109] for solving the similar problem, their expressive power is far behind LL.

Figure 7.5 and Figure 7.6 show clearly that with few agents in a network (less than 50 in Figure 7.5) broadcast is generally better than P2P topology. However, if the number of agents and services increases, P2P with caching becomes a better choice than broadcast. Moreover, both figures show a tendency that while the number of agents and services grows in the network, the difference between P2P with caching and multicast becomes proportionally smaller. Therefore we conclude that if the number of concepts in the network is constant and the number of peers approaches infinity, P2P is almost as good as multicast, if we do not consider the number of messages sent during Chord stabilisation procedure. Anyway, the assumption that the number of concepts is fixed and the number of agents grows, could be interpreted as that in small networks several concepts should be clustered together to achieve higher efficiency. The smaller concept space would mean less mediators and less messages for mediator discovery.

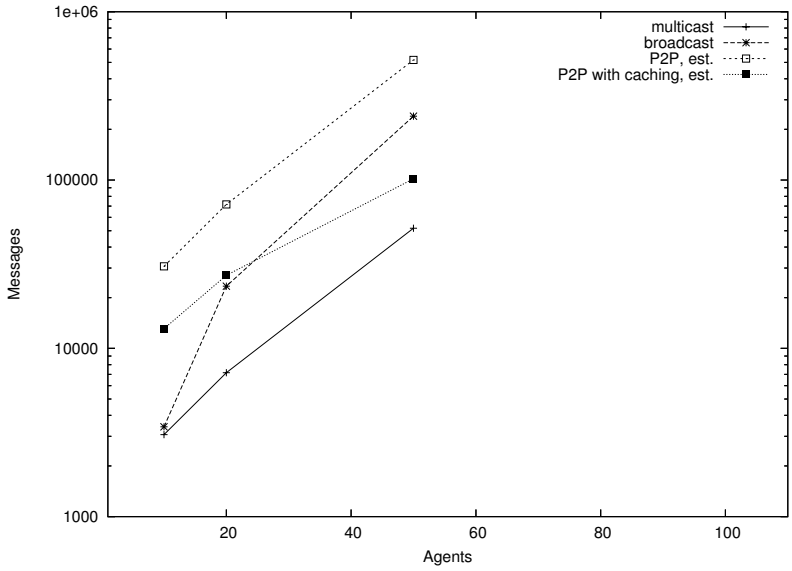


Figure 7.6: Minimum solution length 5.

7.6 Summary

In this chapter we described P2P extension of MAS, which was presented in Chapter 6, for distributed composition of Semantic Web services. The MAS applies this P2P extension for reorganising and configuring its mediators. The main purpose of the mediators is to group agents which share a part of a domain. From service composition point of view these are agents, whose services' inputs or outputs include a common object (literal at the formalisation level). If agents have been gathered in such a way, their location over a distributed system is more efficient and reliable than in non-structured distributed systems.

Although MAS in Chapter 6 can function without the P2P architecture, we believe that P2P would give some added value, especially when it comes to balancing message load between agents. In fact, our empirical/analytical results show that in a system with an increasing number of Semantic Web services and agents, our P2P approach would mean almost the same message load as a system with mediator-based multicast. However, with P2P architecture message load between agents is balanced more evenly compared to a system with multicast, where all messages are routed through a central mediator. Additionally, the usage of P2P would eliminate the central point of failure in the whole system.

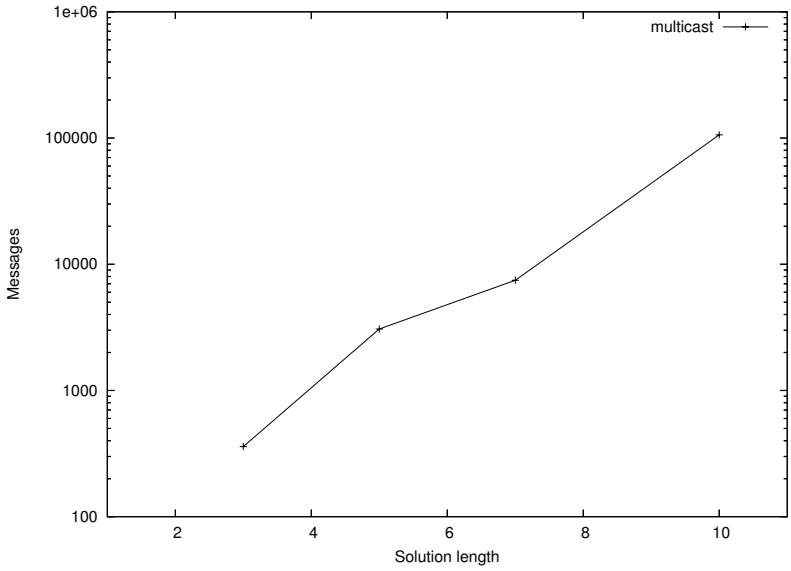


Figure 7.7: Problem solving complexity.

Part III

Applications and Evaluation

Chapter 8

Agent System Implementation

This chapter explains our multi-agent system architecture, which was presented in Chapter 6, from the implementation point of view. The architecture was constructed to support distributed Web service composition, as described earlier, and to facilitate user interaction. The architecture has been implemented in Java by using JADE agent development environment.

JADE (Java Agent DEvelopment framework) is a platform built to help developers implementing multi-agent systems and it is compliant with the FIPA specification. It is written in Java, and is free software. The copyright holder is TILAB which also distributes it as open source software under the terms of the LGPL (Lesser General Public License Version 2). More information about JADE can be found at http://jade.tilab.com/.

8.1 The implementation architecture

The general implementation architecture of our multi-agent system (MAS) is presented in Figure 8.1. The architecture consists of a mediator agent, a database agent, a testbed agent, a monitoring agent and a number of negotiator and graphical user interface (GUI) agents. Additionally there is a Web server for bootstrapping the agent system, database for storing operational information and GUIs. The GUI part of the system is described in a more detailed way in Chapter 9.

The aim of the architecture is to make the system easily reconfigurable. For instance, it is possible to determine, which JADE message transportation protocols (MTP) agents use for communication. Furthermore, new agents with specific capabilities can be added as long as they conform to the overall communication protocol. For instance, we can remove the mediator agent and extend negotiator agents with P2P capabilities to eliminate centralised communication. Instead of a monitoring agent we can also use a testbed agent during performance evaluation and testing.

The negotiator agents can operate without a database and GUI agents. Anyway, a GUI agent can be attached to any negotiator agent, given that it passes the introduced security measures. For system evaluation, testbed agent can simulate GUI agents.

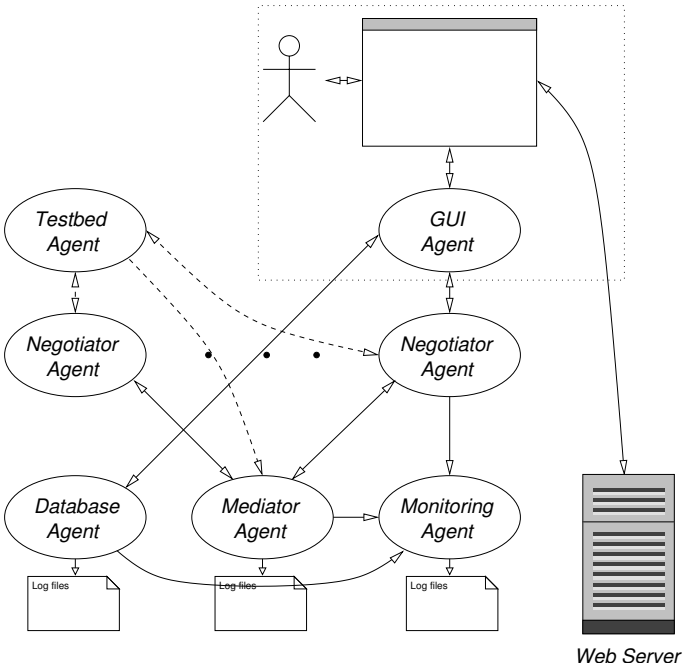


Figure 8.1: Agent system architecture.

The proposed architecture can be seen as an implementation of AGORA [127], which provides an infrastructure, where service providers and requesters can meet with each-other. An instance of AGORA system is depicted in Figure 6.2.

8.2 Agents

In this section we describe the agents, which participate in the proposed agent system. While some agents are application-specific, like the database agent and GUI agents, others, like the mediator agent and negotiator agents, are part of the general agent system.

8.2.1 Database agent

The database agent receives requests from a GUI agent, and facilitates access to the user database. The agent allows the creation of new users, sending forgotten passwords to previously identified e-mail addresses. In addition, the database agent handles registration and removal of negotiator agents from the global agent directory. Through that directory GUI agents can find available agents where to connect.

Moreover, Web services' annotations, found gaps and Web service composition

queries are stored there. The database agent also allows registered users to determine their friend users. Friends can monitor the progress of each-other's agents and share solutions found by their agents.

8.2.2 Monitoring agent

The monitoring agent constantly receives status messages from other participating agents and tries to resolve potential abnormalities in the network. It also measures the efficiency of the system and tries to detect information flow bottlenecks. The received status messages include messages exchanged between agents and statistics about agent operations.

The monitoring agent is especially useful, if negotiator agents participate in a P2P network. In order to keep the network stable, sometimes the monitoring agent should bias the reorganisation of the system. However, the monitoring agent should not become a new potential bottleneck.

The monitoring agent can be replaced with a testbed agent, if the agent system is under evaluation. Furthermore, the testbed agent could be seen as a monitoring agent with extended capabilities.

8.2.3 Negotiator agent

Negotiator agents are the main cooperative problem solving entities of the agent system. They apply symbolic negotiation for solving their declarative tasks. Commands and new problems can be sent to negotiator agents either through a testbed or a GUI agent.

In order to attach GUI agents to negotiator agents, GUI agents have to register themselves at a database agent as trusted parties for particular negotiator agents. Then the database agent would notify the affected negotiator agents about the GUI agent. Additionally, a GUI agent can be connected to a negotiator agent directly, if it knows appropriate user name/password combination.

It should be noted that any number of GUI agents can be attached to each negotiator agent. In this case all of them can control a particular negotiator agent concurrently. Similarly, all results achieved by the negotiator agent are delivered to all registered GUI agents.

Negotiator agents can distribute their messages to each-other either through a mediator agent or a P2P network. In the latter case the mediator agent can be removed from the system.

8.2.4 GUI agent

GUI agents link together the agent system and its applications with GUI-s. They transform user requests to the format internally applied by negotiator agents for problem

solving. Symmetrically, they interpret answers from other agents according to an application logic and display them according to the attached GUI. In other words, GUI agents transforms GUI events to ACL messages used in JADE for agent communication and vice versa.

A GUI agent can only relate to one GUI component and a GUI component can only relate to only one GUI agent. Moreover, each GUI agent can only interact with one negotiator agent. Additionally GUI agents interact with the database agent. Thus the database cannot be accessed directly from the application. Although GUI agents cannot communicate with several negotiator agents at any given time, they can attach themselves to any negotiator agent.

8.2.5 Mediator agent

The purpose of the mediator agent is to mediate messages between negotiator agents. In order to do it, all negotiator agents are required to register themselves at the mediator agent. The negotiator agents would register their contact addresses and their interests at the mediator agent.

The mediator implements two modes of message distribution—broadcast and multicast. While broadcast forwards a message to each known negotiator agent of the agent system, multicast inspects the content of the message and delivers it only to interested parties. It must be noted that the mediator agent is not used together with negotiator agents, which communicate in a P2P fashion.

8.2.6 Testbed agent

Testbed agent is intended for measuring system's performance. It generates test cases for problem solving and initiates a required number of negotiator agents. If required, a mediator agent is spawned as well. After that the testbed agent distributes the generated test cases between spawned negotiator agents and initiates distributed problem solving. Finally, it collects achieved results from participating negotiator agents and measures agent system performance according to collected statistics and results.

The testbed agent interacts with negotiator agents as a GUI agent. However, in contrary to GUI agents, it can communicate with all negotiator agents simultaneously. Also a GUI can be attached to the testbed agent to visualise experimentation results.

8.3 Security

Not every GUI agent can connect to any negotiator agent. There are two ways for GUI agents to register at negotiator agents. The simplest way is to use a password and a user name, which is set when a negotiator agent is spawned. Then a GUI agent uses these credentials to connect to the negotiator agent.

A more sophisticated way is use a database agent to notify particular negotiator agents about trusted GUI agents. These GUI agents would access the negotiator agents without authentification. Anyway, in order to use the database agent, application users must create a user account at the database. The credentials associated with the account are given to a GUI agent for further operations, if an application is started. Then, if a negotiator agent is created by the GUI agent, the GUI agent registers the negotiator agent at the database, and the agent is associated with the particular user.

After that, knowing the credentials, any GUI agent can access the negotiator. Moreover, the users associated as friends to a particular user can attach their GUI agents to user's negotiator agents. The friends are created through the end-application by a GUI agent and through the database agent.

8.4 Supporting infrastructure

In order to support the agent system a number of software components in addition to JADE is required. JADE currently supports the following message transportation protocols (MTP) for inter-agent communication:

- FIPA Mailbox by Owen Cliffe
- JXTA by Shenghua Liu
- IIOP by JADE team
- HTTP by JADE team

While JADE IIOP and HTTP allow agent communication over the Internet, they cannot handle firewalls and network address translation (NAT). However, most of the mainstream Internet community is affected by these technologies. Anyway, FIPA Mailbox and JXTA MTP allow to overcome these issues. Therefore, by using the latter MTP-s, the agent system can be exploited over the Internet without restrictions.

Finally we set up a MySQL server for maintaining the database and a Web document for bootstrapping the agent system. The document describes the locations of a mediator and a database agent. If negotiator agents interact in a P2P manner, then another document describes the location of P2P nodes which are required for bootstrapping the network.

8.5 Summary

In this chapter we described general implementation details of the MAS described in Chapter 6. We identified the key technologies required to set up the MAS. Moreover, we presented the implemented MAS architecture.

The architecture consists of a mediator agent, a database agent, a testbed agent, a monitoring agent and a number of negotiator and GUI agents. Additionally there

is a Web server for bootstrapping the agent system, database for storing operational information and graphical user interfaces (GUI).

Chapter 9

Web Service Composition Tool

Recent progress in the field of Web services has made 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 process and outsource other application services over the Internet. Unfortunately this process relies heavily on the human factor—initially essential Web services are located manually, then business leaders negotiate over the terms of integrating the services and finally programmers integrate the Web services. Due to the low performance of humans (compared to machines), the process is slow and expensive.

Based on the technologies and architectures described in previous chapters we developed a software tool, which automates Web service discovery, composition and integration. The system gets from its user the specification of a required service and a set of conditions, which have to be satisfied in order to integrate the service. Then the system automatically composes the composite Web service. A composite Web service expresses a business process, which captures a particular intra or inter enterprise workflow. In this way the required amount of human efforts is reduced significantly.

The tool is connected to a Web service provision network, where both, service providers and requesters, can proactively negotiate over their Web services' requirements. The Web service provision network is based on the previously described agent system, which facilitates automated Web service composition in a distributed manner. The tool is targeted generally to Web service developers and IT managers. They both would benefit from tools, which would assist them during Web service deployment and suggest potential Web service compositions. Moreover, since the tool hides most of the technical details related to Web services standards, it requires less technical competence to operate.

9.1 Tool description

In order to apply the tool you have to first log into the Web service provision network either by using local or global authentification as shown in Figure 9.1.

Figure 9.1: Login window.

After you have logged in, a GUI is spawned, which allows you to annotate semantically existing Web services and publish the annotations into the network through the underlying MAS such that others could use the annotations as well. The annotation window is depicted in Figure 9.2. During annotation, a developer has to specify the location of a WSDL document and press "Parse" button. If the entered URL represents a valid WSDL document, the document is parsed and the developer can annotate semantically operations described in the document. By annotation we mean giving logical/symbolic names for input and output fields of available Web service operations. In the current state we do not use any tool to manage ontologies explicitly. Thus the developer has to be careful when introducing new logical names, which refer to particular concepts.

Finally, the developer can determine, whether an operation is a core service or not. This piece of information is applied during gap detection, when no solution for a required composite Web service is found. Our gap detection process is described in Chapter 10.

The tool allows developers to specify requirements for composite Web services in terms of required inputs and outputs as depicted in Figure 9.3. Furthermore, gap detection heuristics can be chosen for the case there is no solution for the determined requirements. Additionally user and system context can be inserted into the requirements.

While user context determines a set of user-defined logical name/value pairs for inputs of a required composite Web service, system context queries the operating system (OS) for values of specified parameters (logical names). System context includes for instance computer's IP address, OS type, developer's e-mail address, geographical location, current time, etc. Values in contexts are used later during solution execution.

Finally "Restrict Search" mode can be chosen, which would constrain that found solutions would have only the specified inputs and outputs. Solutions, which would

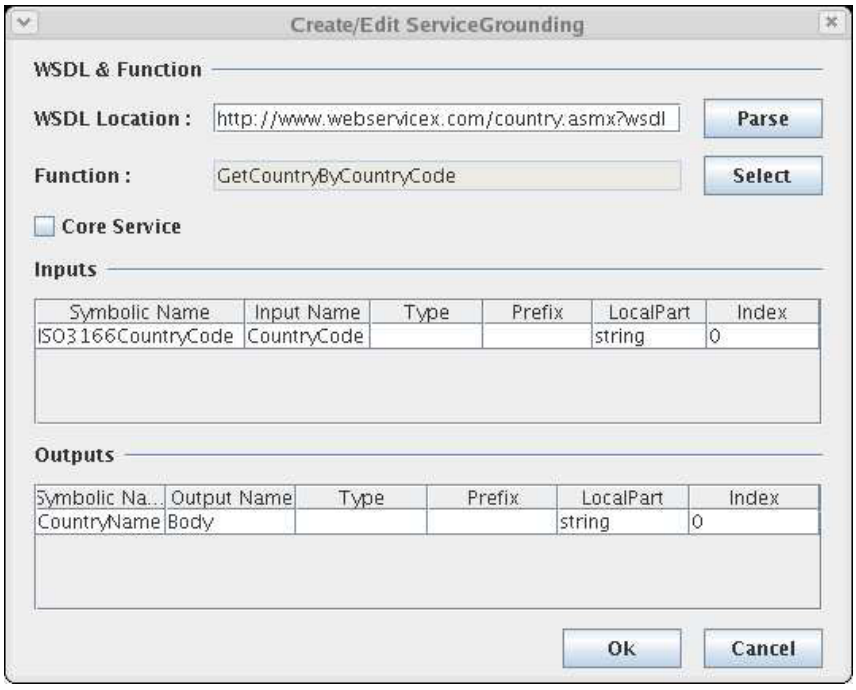


Figure 9.2: Web service annotation window.

have other outputs as a side-effect, would be discarded. Thus "restricted" search applies symbolic negotiation as formalised in Chapter 4, while "non-restricted" search is biased towards practicality and deviates a bit from the proposed formalism.

After button "Search" is pressed, automated Web service composition is initiated. Results are graphically depicted as shown in Figure 9.4. The developer can select between different views for the solution. Currently available views include operational and logical ones. While operation views show data flows from the computation point of view, logical views represent the flow from formal point of view. The developer can also experiment with different graph layout structures to get a better understanding of composed solutions.

Finally, solutions can be executed, if they do not include gaps. Figure 9.5 shows the main execution pane. One can select from the pane a solution and execute it by pressing "Execute" button. The execution engine first looks through the specified contexts for identifying input values of the composite solution. If an input value is not found from the contexts, a dialog is spawned where the user can set it. SOAP messages, which are sent and received during execution, are displayed in the "Execution Result" text area. Execution progress is displayed as in Figure 9.6.

It often happens that a Web service operation outputs a list of values, while the workflow determines that only a single value should be forwarded to another operation.

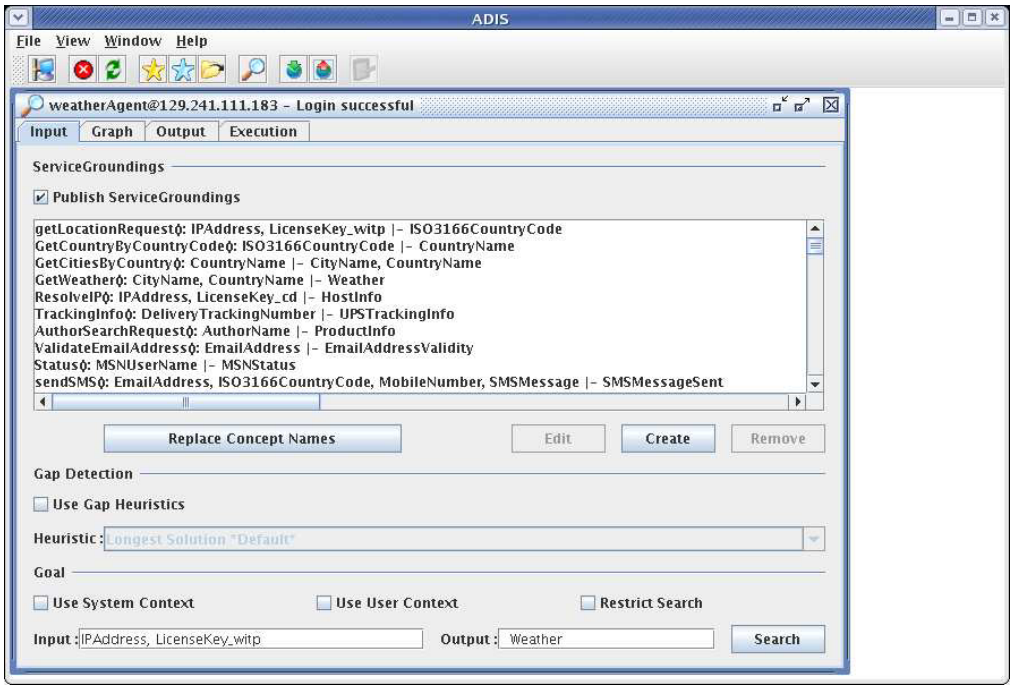


Figure 9.3: Automated composition window.

In these cases a dialog is spawned like in Figure 9.7 and the user is asked to select a value from the list, which would be applied for further computation.

Advantages of the tool are the following:

- users can share search results
- agents can be controlled remotely
- if no complete solution according to requirements can be found, partial ones are proposed
- graphical display of composite Web services
- embeds a full development process starting with annotation and ending with execution plus a BPEL4WS description of the solution
- flexible communication between agents—JXTA, HTTP, IIOP or FIPA Mailbox protocols can be used for communication during agent interaction
- detection of required and missing Web services

Anyway, there are disadvantages of the tool as well. Some of them are listed in the following:

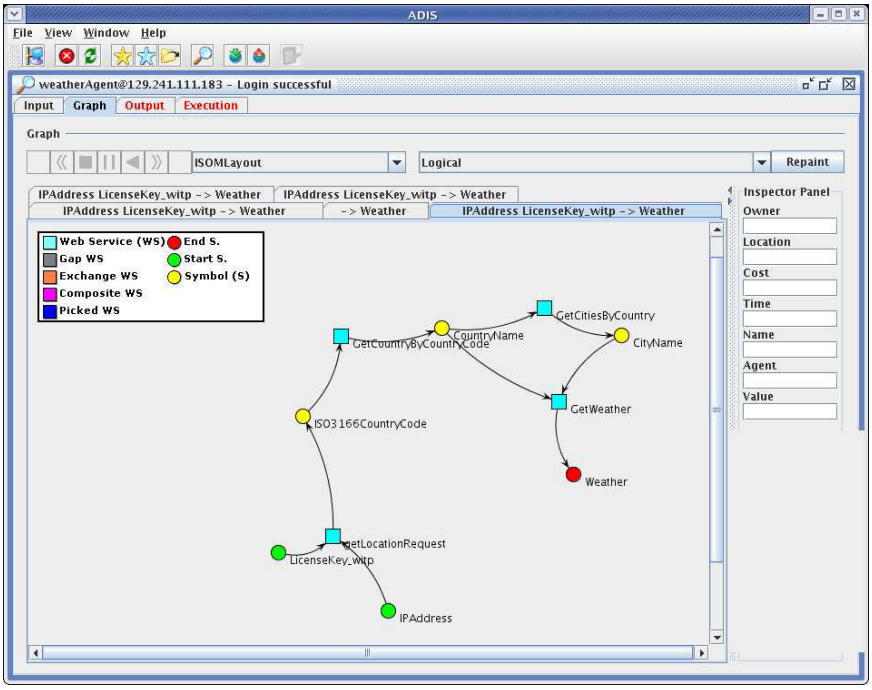


Figure 9.4: Composite Web service solution window.

- specifying requirements for a composite Web service is not user-friendly
- composite solution execution can be improved
- collaboration environment could provide more options, like sharing annotations between participating entities
- annotation is done currently manually

9.2 Tool usage scenario

In order to illustrate usage of the tool and how the main functionality of our software could be applied, let us consider the following example. The example would be refined further and described in greater depth in Chapter 10. It should be mentioned that this example only considers a fragment of the functionality provided by our software and services. We assume that a developer has to construct a composite Web service for selling skis. The service should take as input the body height measured in centimeters (cm), the body weight measured in kilograms (kg), user's skill level and the price limit. The composite service would return a price of recommended pair of skis in Norwegian krone (NOK).

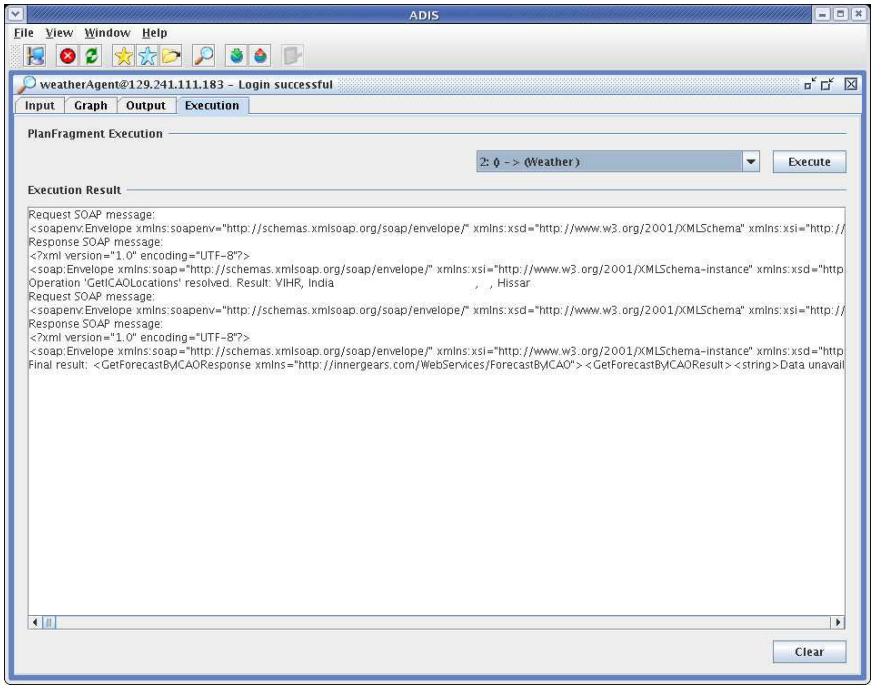


Figure 9.5: Composite Web service execution window.

The core service *selectSkis* accepts the ski length measured in inches, ski brand, ski model and gives the ski price in US dollars (USD). The available value-added services are the following:

- *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 ski model

Developer's view to the composition task under consideration is presented in Figure 9.8. The publishing part represents the available Web services, while the goal part represents requirements for the expected Web service. Currently there are only 3 Web services available. However, in real cases there would be thousands of Web services. After a developer presses the "Search" button, potential solutions are computed.

A solution for the Web service composition, proposed by our system, is presented in Figure 9.9. The figure shows a workflow, which represents a composite Web service, which would compute the price of a pair of skis from identified inputs. GAP0 and GAP1 in the solution identify new Web services, which have to be implemented in order to exploit the composite Web service. GAP0 is a converter from centimeters to inches, while GAP1 represents a currency converter from USD to NOK. After these converters are implemented, the composite Web service can be integrated to company's

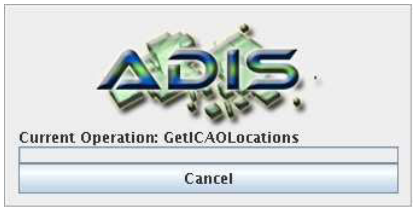


Figure 9.6: Composite Web service execution progress dialog.

Figure 9.7: Interactive execution dialog.

business process by means of exported WSDL and BPEL documents, which describe the proposed workflow. WSDL and BPEL documents are exported automatically from proposed workflows. The gap detection method is described in Chapter 10.

This scenario demonstrated the following functionality of our software:

- automated construction of composite Web services (workflows, business process fragments)
- automated customisation of business processes and Web services
- detection of additional Web services, which have to be implemented, in order to compose and exploit a required Web services

The demonstrated functionality would allow developers to save a lot of time while looking for suitable compositions. Although we demonstrated here a case with 3 Web services only, the real advantage of our software would be perceived in real cases when there are hundreds or even thousands of Web services available.

9.3 Comparison with other tools

A selection of current major Web services tools are presented in Table 9.1. As it can be seen, none of the mentioned tools really provide *automated* Web service composition

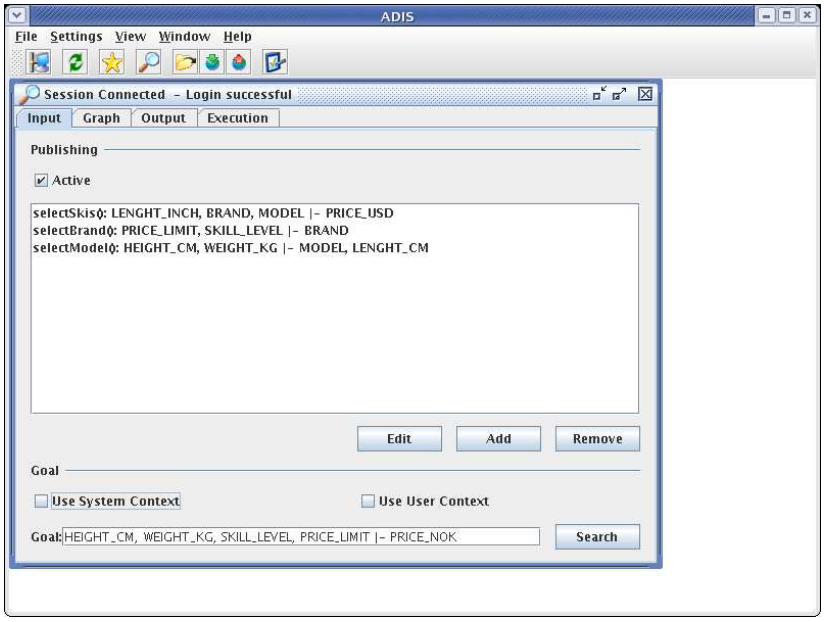


Figure 9.8: Input to our program.

or customisation. Thus our tool and accompanied services would significantly improve these processes through automation. Our tool is most closely related to VCAB from Vergil, which is targeted to IT managers and business analysts. Anyway, our technology would provide an automated approach for solving the similar problem as VCAB does in a goal-oriented way. Therefore we would ease up the task of business analysts, IT-managers and programmers even more than VCAB does.

Web services market is a rapidly expanding market, where tools, which would facilitate easy integration and creation of Web services are urgently required. Although there is already a number of tools available for supporting Web services development, analysis, integration and verification, there are only few commercial tools for facilitating the composition of Web services without writing a single line of code. These tools

Company	Product	Composition	Testing	Managing	Cost (USD)
Mindreef	SOAPScope	no	yes	no	99
Empirix	e-Test Suite	no	yes	no	9,995
Collaxa	Collaxa 2	manual	???	yes	20,000
Infravio	Ensemble	manual	???	yes	50,000
Digital Evolution	Management Server	no	???	yes	150,000
Flamenco Networks	Flamenco WSM	no	???	yes	100,000
Blue Titan Software	Network Director	no	???	yes	150,000
Vergil	VCAB	manual	???	???	24,000

Table 9.1: Comparison of major Web services tools.

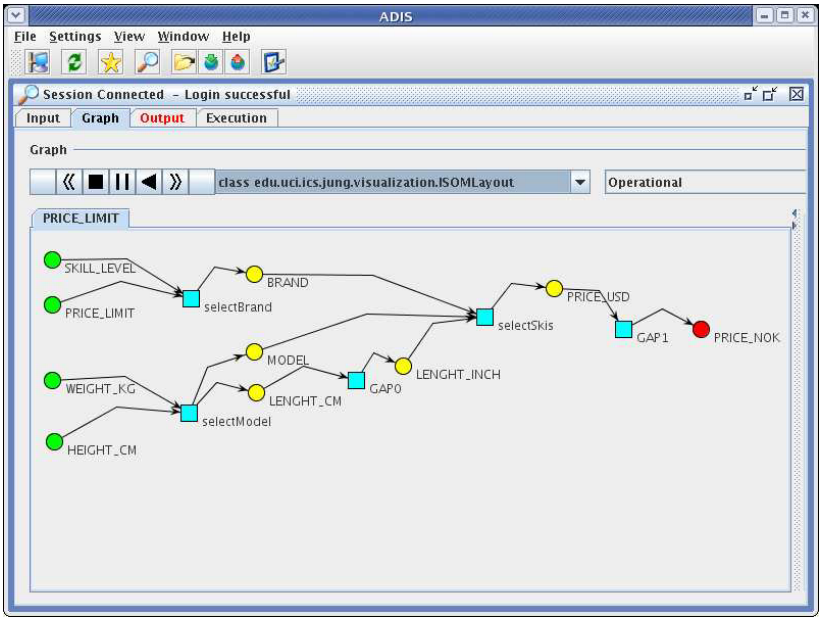


Figure 9.9: A constructed composite Web service.

would allow business analysts and IT managers to design their business processes and incorporate required Web services into the processes. In the simplest case the whole business process would consist of Web services only, which are orchestrated according to business needs. Our tool tries to follow this principle.

9.4 Summary

This chapter describes a tool, which we used to experiment with distributed Web service composition in an automated way. The tool implements the symbolic negotiation framework, which was proposed in Chapter 4. In order to distribute the composition, MAS architecture, described in Chapter 6 and Chapter 8, was deployed.

As a result the tool is connected to a Web service provision network, which brings together users over the Internet. The set of users consists of both, service providers and requesters. Since the tool hides most of the technical details related to Web services standards, it requires minimal technical competence to operate. However, understanding of the basics of semantic Web services is required.

Chapter 10

Detection of Missing Web Services

Many methods have been recently proposed, including ours, for composing automatically Web services from existing ones. The methods range from AI planning to automated theorem proving and graph search algorithms. However, the practical usability of these methods is greatly affected by two assumptions. Firstly, it is assumed that developers provide consistent declarative descriptions of Web services. Secondly, it is assumed that there exists a sufficient set of atomic Web services, which would facilitate the composition of all other Web services. Unfortunately these assumptions are not always satisfied in practice.

In this chapter we propose a method to assist automated composition, if these two assumptions do not hold. In particular, we apply partial deduction for identifying possible inconsistencies in Web service descriptions. Our method also determines possibly missing atomic Web services, which should be implemented in order to compose a requested composite Web service. The method would assist automated composition in situations where theorem proving or any other method would not lead to any composition.

10.1 Generic method description

The generic Web services composition process, involving detection of missing Web services, is presented in Figure 10.1. First, it assumes that descriptions of existing Semantic Web services are translated into extra-logical axioms of LL, and the requirements to the composite service are specified in form of a LL sequent to be proven. Then LL theorem proving is applied to determine whether a composition can be found. If no composition is found then PD and heuristics for detecting missing Web services (gaps) are applied iteratively. While PD generates the set of all possible gaps, heuristics are applied to reduce this set in order to help finding practically useful missing services.

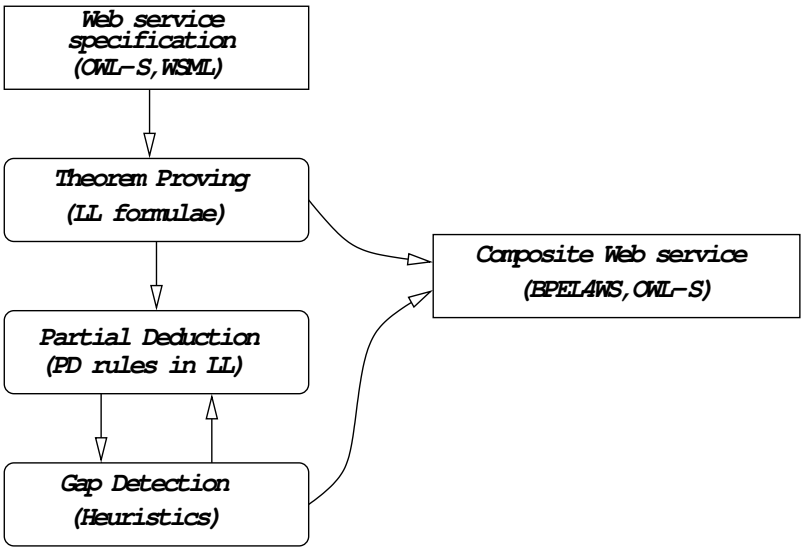


Figure 10.1: The generic composition process.

10.2 LL Web service representation

Generally, we represent the requirements for a composite Web service, as proposed by Rao et al [160], with the following LL sequent:

$$(\Gamma_v, \Gamma_c); \Delta \vdash I \multimap O$$

where both Γ_v and Γ_c are sets of extra-logical axioms representing available value-added Web services and core services respectively, Δ is a conjunction of non-functionality constraints. The constraints could be used either for specifying quantitative or qualitative attributes of required services. $I \multimap O$ is a functionality description of the required composite service. Both I and O are conjunctions of literals. While I represents the set of inputs of the service, O represents the set of outputs produced by the service. Intuitively, the formula can be explained as follows: given a set of available atomic services and the constraints, try to find a combination of services that computes O from I. Every element in Γ_v and Γ_c is in form $\Delta \vdash I \multimap O$, where meanings of Δ, I and O are the same as described above.

Here, we illustrate the LL presentation with an example adapted from [159]. The example considers composition of a ski buying service. We assume that a user provides her body height measured in centimeters (*LENGTH_CM*), body weight measured in kilograms (*WEIGHT_KG*), skill level (*SKILL_LEVEL*) and a price limit (*PRICE_LIMIT*). The user would like to get a price of recommended pair of skis in Norwegian krone (*PRICE_NOK*). The core service *selectSkis* accepts ski length measured in inches (*LENGTH_INCH*), ski brand (*BRAND*), ski model (*MODEL*) and gives the ski price in

US dollars (*PRICE_USD*).

The available value-added services are the following:

- *selectBrand*—given price limit (*PRICE_LIMIT*) and skill level (*SKILL_LEVEL*), provides a brand (*BRAND*)
- *selectModel*—given body height in cm (*LENGTH_CM*) and body weight in kg (*WEIGHT_KG*), provides ski length in inches (*LENGTH_INCH*) and a ski model (*MODEL*)
- *cm2inch*—given ski length in cm (*LENGTH_CM*) provides ski length in inches (*LENGTH_INCH*)
- *USD2NOK*—given ski price in USD (*PRICE_USD*) provides ski price in NOK (*PRICE_NOK*)

The available value-added services are specified as follows (here we omit the step corresponding to translation of the Web service description from OWL-S to LL):

$$\Gamma_v = \begin{array}{l} \vdash PRICE\_LIMIT \otimes SKILL\_LEVEL \multimap_{selectBrand} BRAND \\ \vdash HEIGHT\_CM \otimes WEIGHT\_KG \multimap_{selectModel} LENGTH\_CM \otimes MODEL \\ \vdash LENGTH\_CM \multimap_{cm2inch} LENGTH\_INCH \\ \vdash PRICE\_USD \multimap_{USD2NOK} PRICE\_NOK \end{array}$$

The core service is specified as the following sequent:

$$\Gamma_c = \vdash LENGTH\_INCH \otimes BRAND \otimes MODEL \multimap PRICE\_USD$$

The constraints for the composite service are empty, since we would like to keep the example simple: $\Delta = \emptyset$.

Finally, the requirements for the composite service are specified as follows:

$$(\Gamma_v, \Gamma_c); \Delta \vdash HEIGHT\_CM \otimes WEIGHT\_KG \otimes PRICE\_LIMIT \otimes SKILL\_LEVEL \multimap PRICE\_NOK$$

Using LL prover the required service can be proven to be composable (and then extracted from the proof) from the specification of available value-added services and the core service. The services can be graphically depicted as circles with input and output arcs as it is presented in Figure 10.2, Figure 10.3 and Figure 10.4.

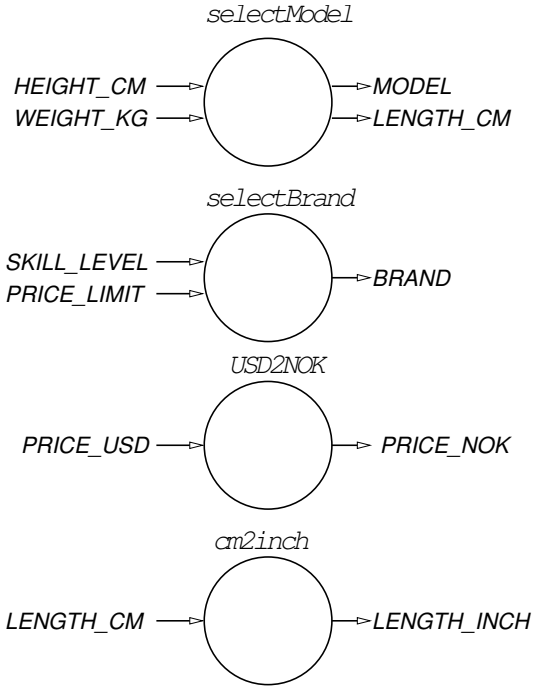


Figure 10.2: Available value-added services.

10.3 Partial deduction and gap detection

In order to demonstrate our gap detection technique, let us continue the example from Section 10.2. Given the above-described Web services we would have a solution depicted in Figure 10.5. However, if we would discard unit conversion services *cm2inch* and *USD2NOK*, there would be no solution available, which would satisfy user's requirements. By discarding these services we assume a situation where these services have not been implemented yet or there are errors in descriptions of these services.

At the same time, by applying PD we would be able to discover these missing Web services. PD is known as one of optimisation techniques in logic programming. Although the original motivation behind PD was to deduce specialised logic programs with respect to a given goal, we apply PD for determining potentially missing Web services. Similar approach has been applied in [128] for automatic software synthesis. One of the motivations there was debugging of declarative software specification. In our work we apply similar technique for debugging and analysing Web services' descriptions.

We formalised PD for LL already in Chapter 3 and proved its soundness and completeness. However, the applicability of PD is relatively limited without domain-dependent heuristics. These heuristics are presented in Section 10.4.

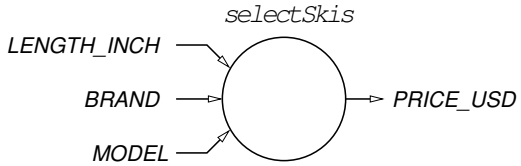


Figure 10.3: The core service for buying skis.

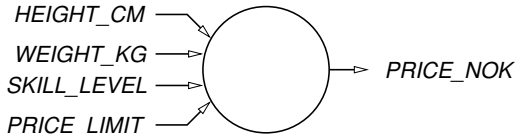


Figure 10.4: The required service for buying skis.

If we return to our example then a partial solution for the Web service composition task is presented in Figure 10.6. GAP0 and GAP1 identify in the figure new Web services, which have to be implemented in order to achieve a composite Web service. It may be also possible that GAP0 and GAP1 represent Semantic Web service description parts, which have to be modified. It could be possible that the developers, who wrote the semantic descriptions of particular Web services, introduced some mistakes into the descriptions.

While constructing the solution in Figure 10.6 initially through PD the following partial solution was deduced:

$$\vdash LENGTH\_CM \otimes BRAND \otimes MODEL \multimap PRICE\_NOK$$

The partial solution represents the solution fragments outside the black box in Figure 9.9. Since the core Web service *selectSkis* does not exist in the partial solution, we place it between the head and the tail of the partial solution constructed through PD. This is done according to a chosen gap detection heuristic, which in the current case is based on differentiation of core and value-added Web services. Anyway, it turns out that additional Web services (represented with GAP0 and GAP1) should be implemented in order to finalise the solution.

10.4 Gap detection heuristics

In this section we propose and evaluate heuristics for gap detection to be used together with PD.

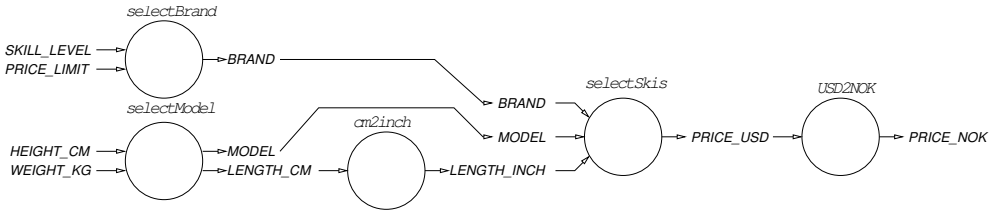


Figure 10.5: The final service structure for buying skis.

10.4.1 The proposed heuristics

We propose the following heuristics for gap detection:

- Heuristic 1: Construct all possible combinations of Web services. Between each two Web services, there may be a gap. Any number of instances of a Web service may exist in a solution. PD is not used in this heuristic.
- Heuristic 2: Construct all possible partial solutions using PD. Each solution includes one gap between the head and the tail of a partial solution.
- Heuristic 3: Select the longest partial solutions from the set of all possible partial solutions constructed using Heuristic 2. Since longer partial solutions are more specific, they may most precisely describe a desired solution. A developer can always cut it shorter or modify further.
- Heuristic 4: Place core Web service(s) into partial solution gaps. A developer can describe core Web services, which must be included in solutions. If the core Web service(s) are not included in a partial solution, we try to place them between the head and the tail of the partial solution.

The heuristics can be extended with the following techniques:

- Measuring the lexical similarity between input/output names. Since a developer might have made syntactical mistakes while describing Web services, just unifying the names might help.
- Using ontologies for deducing subtypes/supertypes and generalising/specialising Web service descriptions on-the-fly.

These techniques would significantly help to debug declarative specifications of Web services and extend the usability of the proposed strategies. For instance, consider a case where a user wants to compose a Web service, which returns *temperature* at a particular location. However, with PD we find a solution, which computes *weather*. Fortunately, *temperature* would be a field in the computed *weather* record. Thus after *weather* has been computed, *temperature* would be extracted from it by applying an ontological knowledge.

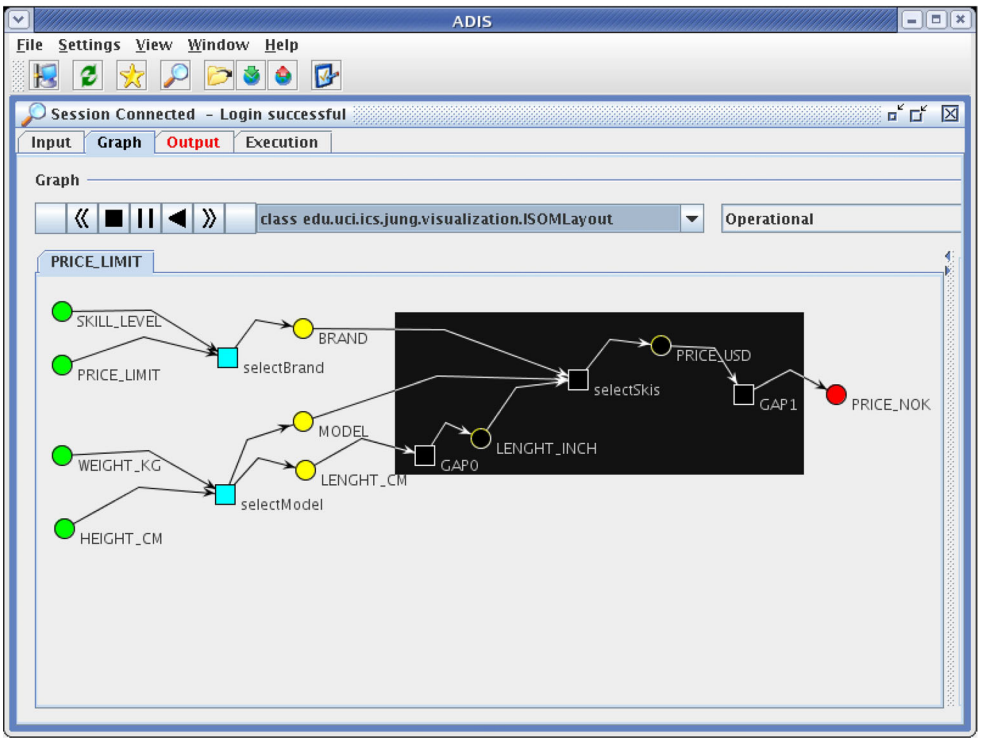


Figure 10.6: Constructed partial composite Web service.

10.4.2 Analytical evaluation of the proposed heuristics

Heuristic 1

All possible permutations of Web services sequences are generated without applying PD. Between each instance of a Web services in a sequence, there may be a gap, which has to be filled. From practical point of view it really would not make sense to apply this method. We listed it just as the worse case strategy in our hierarchy of partial solution construction methods. Given that we have n Web services, there are $n!$ possible combinations for constructing sequences of these Web services. Therefore the construction of all possible partial solutions without PD has factorial complexity.

Heuristic 2

In this case all partial solutions are constructed by applying forward and backward steps of PD. Each solution includes only one gap. The gap exists between the head and the tail of each partial solution. This method suits well for domains with a small number of Web services. In this case the number of partial solutions would be small as well and thereby yet feasible for developers to investigate. Since all possible PD resultants should be constructed, the complexity is the same as for PD itself, which is exponential. The same computational complexity applies to all other heuristics involving PD.

Heuristic 3

The set of all possible partial solutions constructed by Heuristic 2 can be significantly reduced if only the longest solutions are selected. Since longer solutions tend to be more specific, they give a developer better overview of existing options. Compared to the method of exposing all partial solution achieved through PD, this method returns less results to a developer. Therefore, the method suits for larger domains as well.

Heuristic 4

Heuristic 3 can be further specialised by inspecting whether partial solutions already include core Web services. After longest solutions have been selected, core Web services are inserted into gaps. Then new potential gaps between core services can be further detected. The method is especially useful, if there exists only a single core Web service in a solution as in our example in previous sections.

10.5 Query expansion and ontologies

The previously described heuristics were proposed to help discovering gaps in potential composite services. However, sometimes it would be practically useful to modify users' queries in order to get better results or any results at all before detecting gaps

and filling them with potentially missing services. For this, we use ontologies and query expansion techniques during automated composition.

10.5.1 Query expansion

As emphasised already in the introduction, proposing the right queries is a difficult task even for experienced users. Query expansion is a mechanism, which allows users to express partial queries. During composition these queries are automatically expanded to grant better results. To explain the mechanism, let us consider the following case.

We assume that there exists a Web service for the Internet search. The service has 2 inputs—*LicenseKey* and *SearchString*. Additional inputs could be search parameters, restrictions, etc. The output is called *SearchResult*. Since a user is looking for a Web service for a generic search, s/he specifies only *SearchString* for input and *SearchResult* for output. Anyway, s/he might have no idea about additional inputs and outputs, since there are many different search operations available with different input/output signature.

Query expansion takes the original query as an input, looks for potential Web service operations, which partially match the service, and extends the query in such way that particular Web services would be discovered and used during composition. The procedure extends the input part of the query with these input names of Web service operations, which cannot be reached through PD from the initial query in forward-chaining manner.

Symmetrically the output part of the query is extended with output names of Web service operations, which cannot be reached in backward-chaining manner. This mechanism is especially useful in the configuration where a composite Web service, rather than an atomic service, has to be discovered. The query expansion can be also considered as filling gaps by adding parameters into the query rather than adding new services.

10.5.2 Ontologies

In order to demonstrate the usage of ontologies during composition and gap detection procedure, we slightly modify an example from the previous subsection. We assume that there exists a Web service for Google search. The service has 2 inputs—*LicenseKeyGoogle* and *SearchString*. The output is called *GoogleSearchResult*. Since the user is looking for a Web service for general search (not for specific Google search), s/he specifies *SearchString* and *LicenseKey* for input and *SearchResult* for output. However, there is no service available that matches this particular query. Therefore an ontology can be used for making the generic queries more specific.

In the current case our ontology may contain subtyping/specialisation relations between *LicenseKeyGoogle* and *LicenseKey* as well as between *GoogleSearchResult* and *SearchResult*. Then the initial query can be modified to have *LicenseKeyGoogle* and *SearchString* as input and *GoogleSearchResult* as output. By using ontologies in such

a way during automated composition we can use generic templates for matching semantically similar Web service operations. This approach is especially useful, when there are many semantically similar Web service operations available.

10.6 Summary

In this chapter we applied partial deduction for determining possible missing Web services and for identifying possible inconsistencies in Web service descriptions. These issues, despite being important for practical systems, have not yet been described in the literature of automated Web service composition.

The heuristics described here can be applied also in a more generic setting. Namely, for analysing which Web services in a particular *domain* could have large potential for exploitation but have not been implemented yet. This information would be especially useful for Web services provision companies, since it identifies new market opportunities. A case study regarding this issue is presented in Chapter 11.

In order to apply our composition and gap detection methods to "non-semantic" Web services (for example, described in WSDL), we have to annotate existing Web services with semantic information. Until now it has been done manually. However, alternative methods are required to extract semantics automatically from WSDL documents.

Chapter 11

Applicability of Automated Composition

Recently the field of Web services has gained attention both in industry and academia. While industry has been mostly interested in standardisation and promotion of the technology, academia has been looking for ways to fit the technology into other frameworks, such as the Semantic Web. Anyway, despite of the increased academic and commercial interest to Web services, there are currently only few case studies available about Web services in the Semantic Web context. Moreover, according to authors' knowledge, there is no publicly available study analysing which data is currently mostly provided/required by Web services and which Web service domains are available. Neither is there any case study regarding applicability of automated Web service composition.

In this chapter we target these shortcomings by providing a case study of semantically annotated commercial and governmental Web services. We analyse interaction and potential synergy between commercial and governmental Web services. Also the role of ontologies for semantic integration of Web services is analysed. Moreover, we identify the most common data exploited by current Web services.

Automated Web services composition has been largely seen as a methodology for constructing new Web services from existing ones. We propose that there are other applications of automated Web services composition than those considered traditionally. This chapter demonstrates that, even, if automated composition might not be a killer application of software engineering, it helps to analyse and package existing Web services.

The roots of the work presented in this chapter are strongly related to our previous and current work on automated Web service composition [106, 160]. After developing a system for automated Web service composition, we wanted to evaluate its performance and applicability in a "real-world" configuration. While performing the evaluation we became interested in the current Web services roadmap. More specifically, we wanted to identify which kind of Web services are available and which are the most common inputs and outputs of current Web services. A special focus was set to

analysis of potential interactions between commercial and governmental Web services.

11.1 Structural analysis of data types

Before the full power of automated Web service composition could be harnessed, a methodology for mapping existing Web service descriptions in WSDL into semantically enriched ones should be developed. This methodology would provide a bridge between academic and industrial efforts. In this light it is important to understand in which extent existing Web services annotations can be reused. Since the number of available Web services is rapidly growing, reuse becomes a central issue in annotation. In this section we try to estimate upper and lower bounds for the reuse of Web services' annotations.

Kim and Rosu [88] are similarly to us concerned with determining generic properties of Web services. However, their main emphasis is set to learn which basic types are mostly used in WSDL descriptions, while we focus here to measuring reusability of data types. Based on that statistics they estimate average size of SOAP messages, which are delivered during Web service execution. They also measure average Web service execution time from two different servers.

For analysing the general commercial Web services' and data type structures, we first used Google search engine to collect a set of WSDL URLs. Altogether we collected 1276 URLs containing 13398 operations. Then we extracted data type structures from WSDL document operation descriptions. This resulted in 26796 WSDL message definitions including both input and output messages of available operations. These message definitions contain data structure trees starting with the part element of WSDL and ending with basic types such as int, string, etc. as leaf nodes.

Given these trees, we analysed the overlapping of data structures. Basically, we measured how many data structures are unique, strictly unique, structurally unique and strictly structurally unique. These properties are defined as follows:

- uniqueness—a data structure is unique, if there is no other data structure matching its data field *names* and corresponding data field *types* from the root node down to the leaf nodes

- strict uniqueness—a data structure is strictly unique, if all complex types, that it includes, are unique with respect to all complex types of all other data structures

- structural uniqueness—a data structure is structurally unique, if there is no other data structure matching its data field *types* from the root node down to the leaf nodes

- strict structural uniqueness—a data structure is strictly structurally unique, if all complex types, that it includes, are structurally unique with respect to complex types of all other data structures

Table 11.1: Uniqueness of data structures.

	Data structures	% of all data structures
Unique	17950	67
Structurally unique	11562	43
Strictly unique	13639	51
Strictly structurally unique	3659	14

To illustrate these properties, let us consider Figure 11.1. All data structures in this figure are unique. However, data structures in Figure 11.1(b) and Figure 11.1(c) are not strictly unique, since they share a common complex type. Data structures in Figure 11.1(a) and Figure 11.1(b) are not structurally unique since they have a common structure, if we consider only type information. Finally, none of the data structures is strictly structurally unique. We should mention that the order of elements in data structures is not relevant when measuring uniqueness.

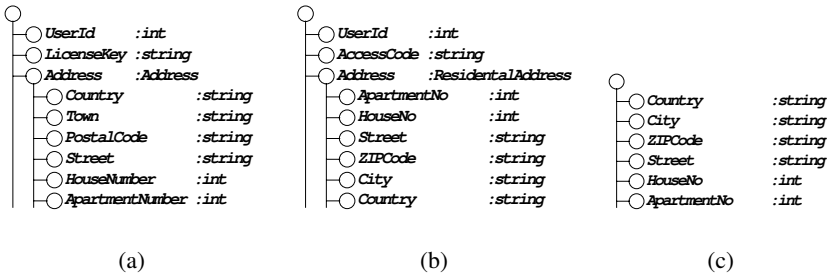


Figure 11.1: Data structure examples.

From 26796 data structures 17950 were unique and 11562 were structurally unique. From 17950 unique data structures 4311 were partially overlapping, which means that 13639 were strictly unique. From 17950 data structures 14291 were partially structurally overlapping, which means that only 3659 data structures were strictly structurally unique. The results are summarised in Table 11.1.

The statistics in Table 11.1 allows to evaluate the degree of reusability of existing semantical descriptions during annotation. While uniqueness gives the higher boundary for data types, which should be annotated in the worst case, partial uniqueness shows in which extent existing annotations could be partially reused. Structural and strictly structural uniqueness could be viewed as a measure for potential improvement, which would be achieved through automated annotation. Thus in the best case only 14% of all data structures should be annotated manually when automated annotation is applied.

Figure 11.2 depicts statistics about data structure sizes. The size is measured in

terms of the overall number of leaf nodes in a data structure. The leaf nodes represent basic types in type definitions. While most of the data structures include only few leaf nodes, there are a few structures, which include about 200 leaf nodes. Some types are even recursive. However, we count recursive types only once and do not follow their recursive branches.

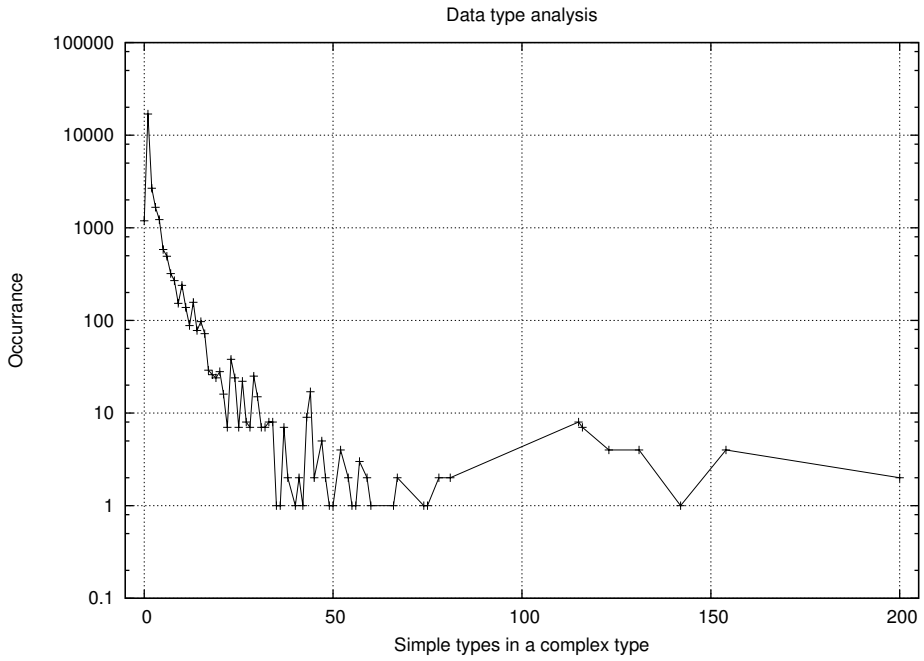


Figure 11.2: Data type complexity in terms of size.

11.2 Annotating Web services

In order to analyse available Web services with our automated composition method, we first annotated semantically Web service operations under consideration. By annotating we mean a process of giving logical names to inputs and outputs of Web service operations. These logical names refer to particular concepts in an ontology and represent the semantics of data, which is exploited by Web services. In the rest of this chapter, when counting the number of relations in developed ontologies, we state explicitly only the number of relations, which represent subclass/superclass relations. Relations, which represent links between Web service operations and data, are not counted.

Although there are some tools available, which allow semi-automatic annotation of Web services, our annotations were constructed manually. The reason is that the

Table 11.2: Annotation overview.

Domain	Operations	Concepts	Ontology size
X-Road	96	595	128
Commercial	493	578	189
Merged	589	1149	317

current annotation tools are not flexible and mature enough for general usage. Additionally we had to develop new ontologies for describing data structures and relations between their fields.

Commercial Web services' WSDL documents were retrieved through the list at SalCentral.com in March 2005. From the list of available Web services we annotated most of the available operations, whose semantics was clear. Altogether we annotated 493 commercial Web service operations. Additionally we developed an ontology for commercial Web services, which consists of 189 relations. The overall commercial Web services domain contains 578 concepts.

For governmental Web services we chose the services from X-Road [143] project, which was initiated by Estonian government. X-Road is a middle-tier data exchange layer enabling government databases to communicate with their clients. The system allows officials, as well as legal and natural persons, to search data from national databases over the Internet within the limits of their authority. The system ensures sufficient security for the treatment of inquiries made to databases and responses received.

X-Road project was initiated in 2001 and by March 2005 X-Road had already 41 databases providing services plus 354 institutions and companies using the services. The overall number of available Web service operations was 687. We annotated 96 of them. The domain and the developed ontology consists of 595 concepts and 128 relations. The reason of having a larger ontology for commercial Web services (compared to governmental services) is potentially due to the larger heterogeneity of data in this domain. While governmental Web services are centered around queries about citizens and companies, commercial Web services provide a wider set of Web services.

After merging X-Road and commercial Web service domains, the merged domain consisted of 1149 concepts and 589 operations. 24 concepts were shared between the domains. These concepts represent potential interactions between commercial and governmental services. Table 11.2 summarises the number of annotated Web service operations, concepts and relations in developed ontologies.

In order to visualise Web services roadmaps for commercial and X-Road Web services, we constructed graphs showing potential data flows. The roadmaps of commercial Web services, governmental Web services and the domain containing them both are depicted respectively in Figure 11.3, Figure 11.4 and Figure 11.5. Nodes in the graphs represent concepts and edges represent potential data flows implemented by

Web services. The size of a node shows proportionally its importance in the domain—larger nodes are used by a larger number of Web service operations.

Surprisingly the topologies of the roadmaps have a similar structure like the Web itself [10, 26]. There are tubes, tendrils, disconnected components and strongly connected components. The graphs identify strict inputs (which appear only as inputs of Web service operations) strict outputs (which appear only as outputs of Web service operations) and intermediary data objects (which appear both as inputs and outputs of the operations).

11.3 Challenges of annotation

While annotating Web services and building ontologies we encountered a number of challenges, which either limited our efforts or made in some cases annotation even impossible. A very important factor is the usage of a wide variety of languages in WSDL files. Although most of the WSDL files were documented in English and the same language was used for naming inputs and outputs, many services contain data in other languages as well. This naturally complicates the extraction of semantics from WSDL files.

Moreover, there is often too little or even misleading information available about Web services and data fields. For instance data field name *country* may refer to a country name in a particular language or represent a country code according to any related standard. There is a general bias not to document data fields in commercial Web services. While X-Road services had mostly data fields commented, only one percent of all available commercial Web services had comments for data fields. Anyway, the situation for service and operation documentation in commercial Web services was much better. In particular, 127 of 404 available Web services and 2443 of 3764 operations were documented.

Data with the same meaning is encapsulated in different data types. For instance, an address may be represented with a string or a data type containing fields for each element of an address. Furthermore, sometimes an address contains a country name while in other cases it represents only a street name and house/apartment number.

In summary, the main challenges are as follows:

- different languages
- lack of documentation in WSDL documents
- different data structures with the same meaning
- dynamically changing WSDL documents
- availability of WSDL documents

Therefore, in order to facilitate automated annotation and further usage of annotated Web services, it is desirable to look for alternative descriptions of Web services,

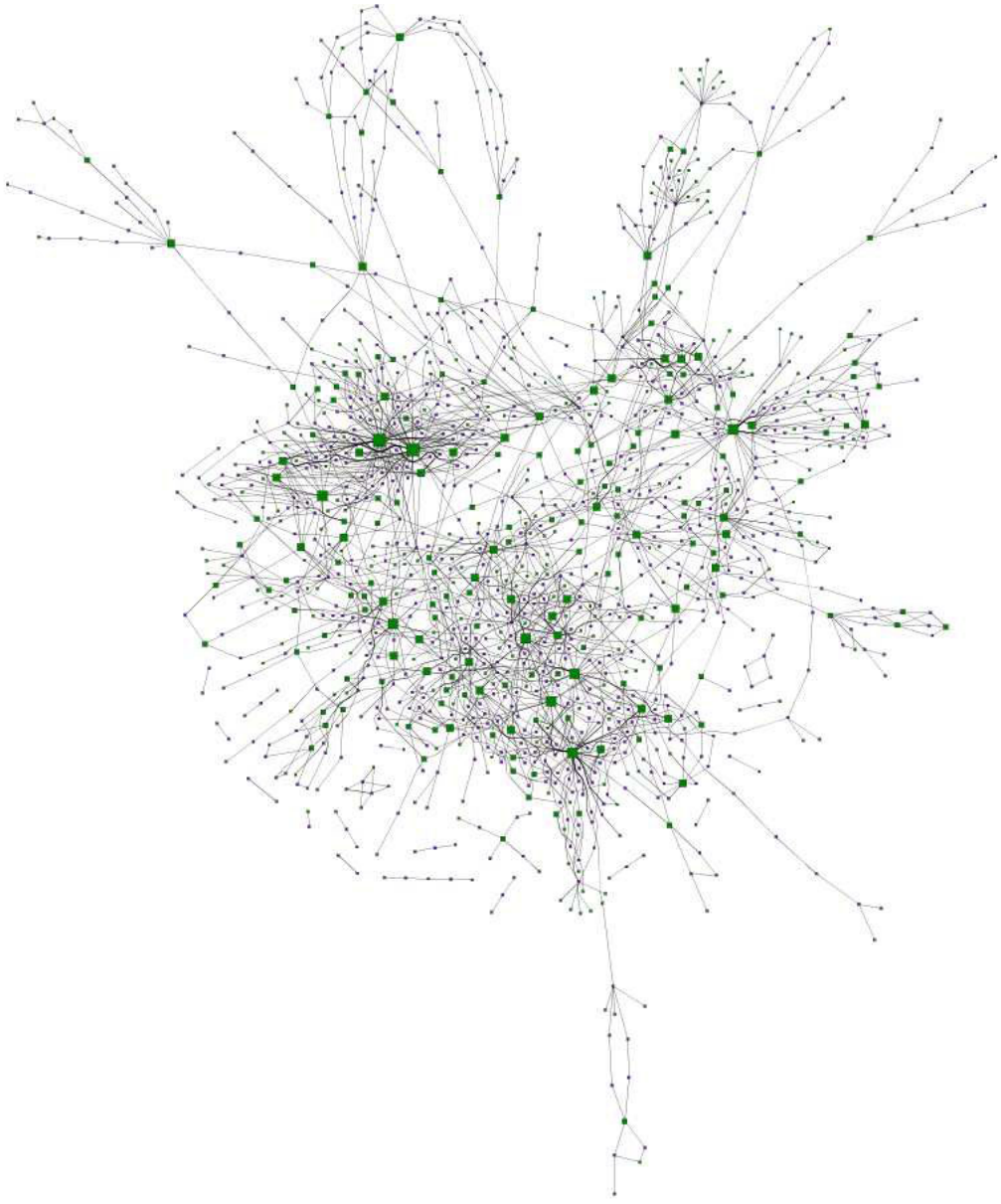


Figure 11.3: Roadmap of commercial Web services.

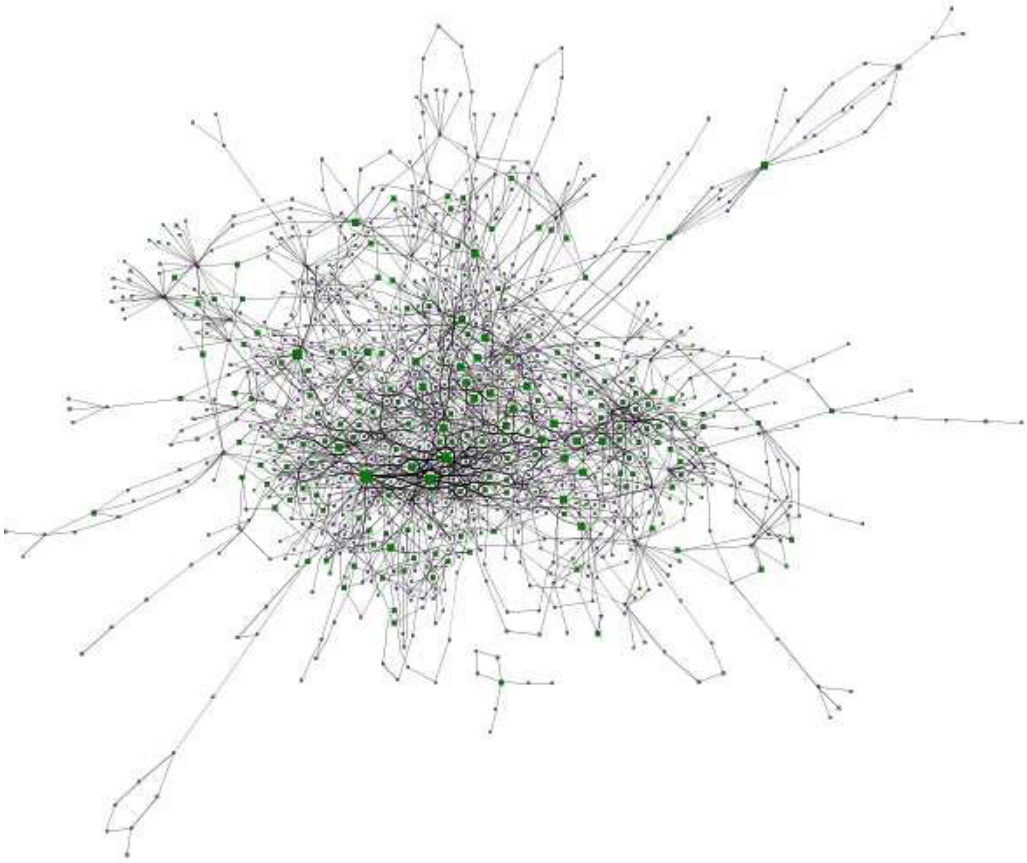


Figure 11.4: Roadmap of governmental Web services.

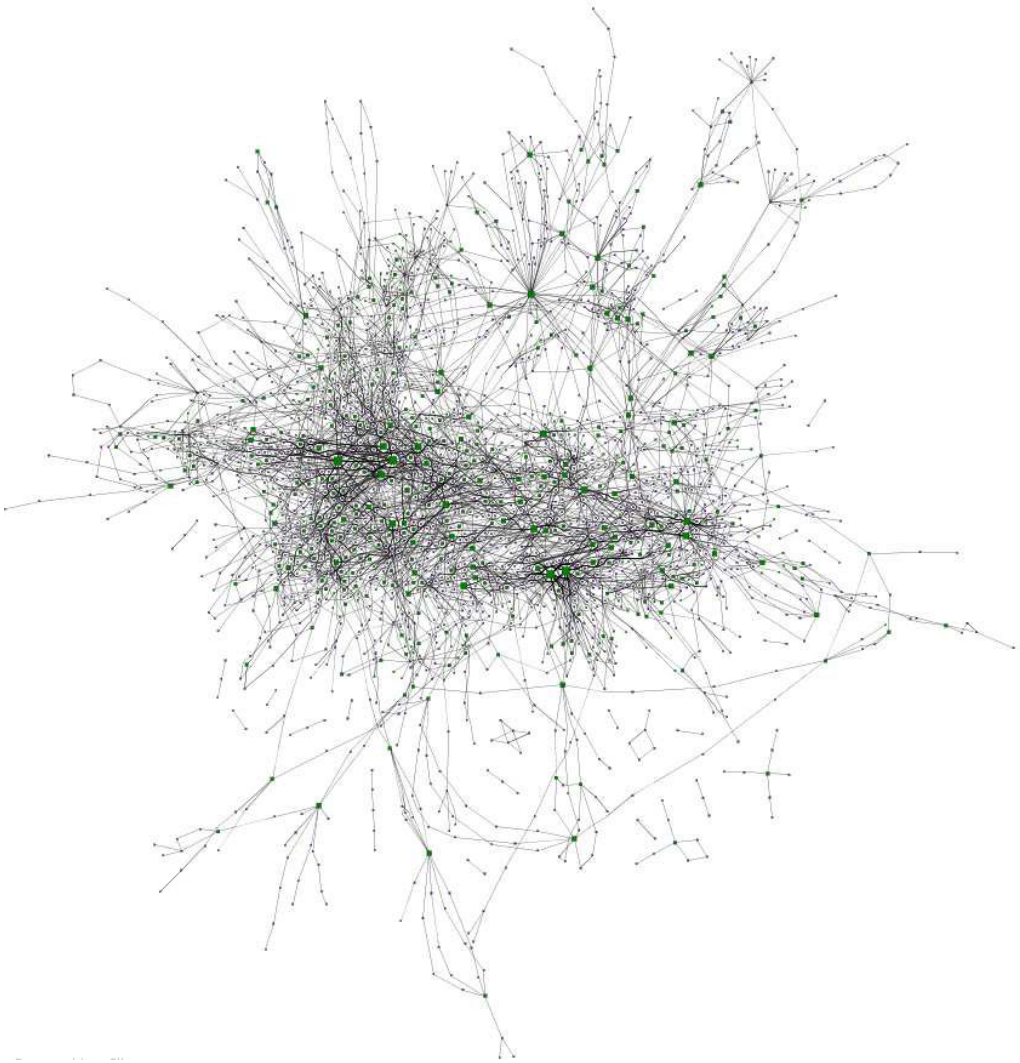


Figure 11.5: Roadmap of all annotated Web services.

like their source code, as done by Sabou [163], or UDDI tModels. Additionally, online dictionaries like WordNet could be exploited to cope with a variety of languages which are used to document WSDL documents. The latter of course requires that there is a way to identify natural languages, which are used within WSDL documents.

11.4 Commercial vs. governmental Web services

Our case study identifies the general Web services domain structure as depicted in Figure 11.6. The structure was extracted from a previously constructed data flow graph including all annotated Web services. There are 3 components of the domain:

- strict input data
- strict output data
- intermediary data

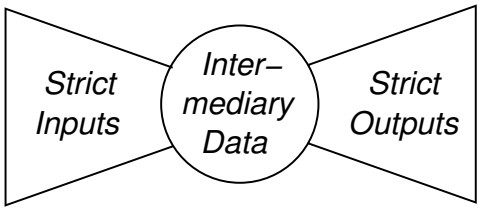


Figure 11.6: General Web services' domain structure.

Strict input data is the data, which only serves as input to any Web service, while strict output data serves solely as output of any Web service. Intermediary data is presented both in inputs and outputs of Web services. From automated Web services composition point of view strict input and output data can exist respectively only in the inputs and outputs of composite Web services. However, intermediary data is the most crucial for the composition—intermediary data allows to compose multiple Web services into a required workflow.

To clarify what we mean by strict inputs, strict outputs and intermediary data, let us consider the Web services domain consisting of the following Web service operations:

$$\vdash \mathit{IPAddress} \multimap_{\mathit{getGeoIP}} \mathit{CountryName} \otimes \mathit{ISO3166CountryCode},$$
$$\vdash \mathit{CountryName} \multimap_{\mathit{getCapitalCity}} \mathit{CityName},$$
$$\vdash \mathit{CityName} \multimap_{\mathit{getPopulationCount}} \mathit{CityPopulationCount},$$
$$\vdash \mathit{CityName} \multimap_{\mathit{getWeather}} \mathit{Weather},$$
$$\vdash \mathit{CurrencyCode} \multimap_{\mathit{getRate}} \mathit{CurrencyRate}.$$

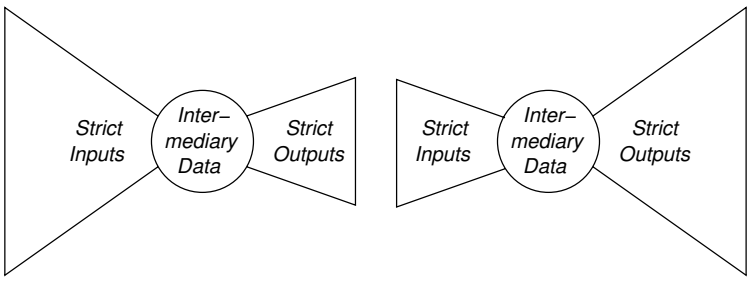
In this domain we have 8 concepts, which are in the following roles:

- strict input data—*IPAddress*, *CurrencyCode*
- strict output data—*ISO3166CountryCode*, *CityPopulationCount*, *Weather*, *CurrencyRate*
- intermediary data—*CountryName*, *CountryCode*

One may argue here that the concept of strict inputs/outputs is too restrictive since data structures' roles change in time and depend on particular contexts. However, the concept allows to measure the maximum length of automatically composable workflows and to evaluate limitations and applicability of automated Web service composition algorithms.

Our case study identified a fundamental difference between commercial and governmental Web services domains. While governmental Web services tend to have relatively simple data types for input and more complex data types in outputs, commercial Web services have rather complex data types as inputs and much simpler ones as outputs. This tendency is depicted in Figure 11.7. The tendency could be explained by considering the main aims of Web services in these domains. Governmental Web services mostly facilitate access to databases and thus return rich data objects according to simple queries. Commercial Web services, however, are more computation-oriented. They accept rich data structures as input and return compact results of particular computations.

Understanding this difference between commercial and governmental Web services is crucial while developing applications involving Web services from both domains. Furthermore, composite Web services with simple inputs and outputs can be composed by combining Web services from both domains. However, these composite Web services would involve a heavy data transfer between them. Symmetrically composite Web services with rich inputs and outputs can be composed under similar conditions. Anyway, these composite Web services would involve less data transfer between atomic Web services compared to preceding composite Web services.



(a) Commercial Web services. (b) Governmental Web services.

Figure 11.7: Domain-specific Web services' domain structures.

Table 11.3: Domain overview before removing isolated Web service operations.

Data	Commercial	X-Road	Merged
Strict input data	201	20	208
Strict output data	129	205	332
Intermediary data	66	65	123

Table 11.4: Domain overview after removing isolated Web service operations.

Data	Commercial	X-Road	Merged
Strict input data	156	18	162
Strict output data	97	197	293
Intermediary data	66	65	123

By analysing graph structures in Figure 11.3, Figure 11.4 and Figure 11.5 representing potential data flow between Web services, we removed respectively 81, 14 and 90 isolated Web service operations from commercial, X-Road and the merged domain. Isolated Web services have only strict inputs and strict outputs, respectively for inputs and outputs. For instance *getRate*, from our preceding domain example, is an isolated operation, since its only input *CurrencyCode* is a strict input and its only output *CurrencyRate* is a strict output.

Since these services are not engaged with data flows, they would not be a part of composite Web services anyway. However, isolated Web services may indicate potential missing Web services, which have to be implemented in order to place them into composite Web services. Table 11.3 and Table 11.4 summarise the number of strict input data, strict output data and intermediary data respectively before and after removing isolated Web service operations from considered domains.

11.5 Analysis of the Web services roadmap

In this section we analyse the Web services domains presented in Figure 11.3, Figure 11.4 and Figure 11.5.

11.5.1 The effect of ontologies

The usage of ontologies allows us to reduce the number of isolated Web services in a domain. While ontologies had no effect on X-Road domain itself, they allowed to bind commercial Web service operations to X-Road operations. Moreover, the usage of ontologies allowed to reduce the number of isolated commercial Web service op-

erations from 46 to 22. The ontologies had no effect to X-Road operations because governmental Web services are more homogeneous compared to commercial ones.

11.5.2 Available Web services

Generally governmental and commercial Web services provide different services. Overlapping areas are related to queries about companies/businesses and contact information for persons and businesses. However, governmental Web services in that area facilitate access to more sensitive information and have therefore limited access.

Available commercial Web services

Most common commercial Web services are currently related to yellow pages services—locating and finding information about businesses, persons and Internet hosts. Also services for measuring distance between physical locations are included in this category. The second category of services deals with communication—instant messaging, sending e-mails, faxes and SMS-es. Third category of services is devoted to financial sector—fetching stock exchange information, market news and performing currency conversions are the key services here.

Naturally, there are services for general Internet search and text translation. Related to this group are services dealing with data conversion, such as mapping documents from one format to another. Some services have been implemented for getting weather forecasts, airport and travelling information. There are also some services for arithmetics, statistics and encryption. Services for direct e-commerce and e-business are still in minority—there exist only few services for product search, processing credit cards and package tracking. Finally a couple of services have been published for generating graphs, charts and bar codes.

Available governmental Web services

From the overall set of Web services in X-Road we enlist here only Web services, which might be accessible for public usage. They include Web services for checking validity of particular documents, such as driving licenses, passports, diplomas and certificates, which have been assigned by participating institutions. Services for providing contact information for particular persons, companies and other organisations are also important. Finally, there are Web services for checking ownership of real estate properties and vehicles. By making these services public, citizens would become able to check information related to buying/selling real estate and vehicles without intermediaries.

Governmental Web services are characterised by high redundancy. There are many Web services exploiting similar data, both in inputs and outputs. However, a slight variation in inputs and outputs mainly arises from regulations identifying which data should be accessible to specific parties.

11.5.3 Synergy between commercial and governmental Web services

Given the restrictions associated with accessing governmental Web services, these Web services could be viewed mostly as core services, while commercial Web services could serve as value-added services. In other words, due to the heterogeneity of the commercial Web services domain, these Web services could be applied for customising governmental Web services. Governmental Web services could also be used for accessing particular persons or companies while commercial Web services could provide generic statistical functions for facilitating this process.

Most of the currently available commercial Web services could be used for customising governmental Web services. For example, instant messaging could be exploited as an alternative medium for accessing citizens and officials. Internet search and translation can be applied for fetching more information about companies and persons and facilitating thereby more adequate decision making.

There is still a lot of space for additional Web services. X-Road, for instance, could provide Web services for fetching statistics, which is essential for businesses and decision-making. There is a strong need for commercial Web services, which would take generic person or company data, such as name and birth/foundation date, and return some useful information about them.

There could also exist governmental Web services for delivering information, such as warnings to the citizens travelling in a particular region. This information can be also delivered to travellers by travel agents themselves. Thus there are a lot potential for new Web services combining governmental and commercial Web services.

11.5.4 Most common semantic data

In this section we analyse, which kind of data is most commonly exploited in the currently available Web services. We consider X-Road and commercial Web service domains and the domain resulting from merging these two domains.

X-Road domain

From 595 concepts in X-Road domain, 25 concepts were involved in 10 or more operations either as inputs or outputs. The most important concepts were national identification code, first name and last name. Other concepts include different dates (birth date, event occurrence date, etc.), business registry code, address, e-mail address, street name, postal code, country name and code, company name and various messages.

Commercial Web services domain

From 578 concepts in commercial services domain, 40 concepts were involved in 10 or more operations. The most important ones were user name and password followed by ZIP code, date, license key, city name, postal code and e-mail address. User name,

password and license key are currently most important mechanisms for controlling access to commercial Web services. While governmental Web services handle authentication at system level, commercial Web services limit access by expecting users to provide user names, passwords, license keys, access/account codes, etc.

Additional important concepts were country name, currency, stock symbol, message content (for instance messaging, SMS and e-mails), search string, name, IP address, URL and words to be translated. Other concepts include weather, (US) state name, interest rate type, location, country codes, date, distance, postal address and delivery tracking numbers.

Merged domain

From 1149 concepts in the merged services domain, 64 concepts were involved in 10 or more operations. Most important of them were user name, password, national identification code, first name, last name, ZIP code, date, e-mail address, postal code, license key, city name, country name and message content.

24 of 1149 concepts were overlapping between X-Road and commercial services with ontologies. These concepts represent the data, which can be currently directly passed between commercial and governmental Web services. With other words, these concepts represent intersection between these two domains.

The overlapping concepts are birth date, city name, company name, country code/name, county, currency, date, day, e-mail address, fax number, first name, last name, gender, IP address, message content, mobile phone number, month, name, phone number, postal code, street name, year and a general string. In general, these concepts are mostly related to everyday communication.

11.6 Automated composition for analysis

In this section we describe how we applied an implementation of our method [160] for automated Web service composition to analyse the semantically annotated subset of Web service operations. Our aim is to figure out whether automated Web service composition (in particular, our implementation) is applicable for industrial applications. Furthermore, we would also like to figure out whether the methodology could be applied for analysing existing Web services domains for industrial and academic purposes.

We applied our automated composition method in the following manner. First we included all strict inputs and intermediary data nodes into inputs of the required composite Web service. For each element from strict outputs and intermediary nodes we applied automated composition such that the output of the required composite Web service consisted of the selected element. An intermediary node in the input part was deleted, if it also existed in the output part. The pseudocode of the algorithm is presented in Figure 11.8.

```
Algorithm AnalyseDomain(ops, I, M, O)
begin
    results ← ∅
    for ∀output ∈ M ∪ O
        inputs ← I ∪ M \ output
        results ← results ∪ compose(ops, inputs, output)
    end for
    analyseCompositeServices(results)
end AnalyseDomain
```

Figure 11.8: Automated Web service composition for analysis.

The algorithm takes a set of annotated Web service operations *ops*, strict inputs I, intermediary nodes M, strict outputs O as an input. Then all possible compositions are computed through *compose*, which refers to our composition method, and then analysed further by *analyseCompositeServices*. Method *analyseCompositeServices* basically analyses, which composition problems were solved (and which not), which compositions included Web service operations from different domains and which Web service operations mostly occurred in compositions. Additionally composition lengths are analysed. According to that knowledge one can derive which Web service operations are most popular, which are possible interaction points between different domains and which Web service operations do not belong to any composition.

To illustrate the algorithm, let us consider the same domain from Section 11.4. Given that *getRate* was removed from the domain, since it was an isolated operation, we have the following domain topology:

- strict input data—*IPAddress*
- strict output data—*ISO3166CountryCode*, *CityPopulationCount*, *Weather*
- intermediary data—*CountryName*, *CountryCode*

According to the algorithm we have to apply automated composition to the following Web service descriptions:

$$\vdash IPAddress \otimes CountryName \otimes CityName \multimap_{s_1} ISO3166CountryCode,$$
$$\vdash IPAddress \otimes CountryName \otimes CityName \multimap_{s_2} Weather,$$
$$\vdash IPAddress \otimes CountryName \otimes CityName \multimap_{s_3} CityPopulationCount,$$
$$\vdash IPAddress \otimes CountryName \multimap_{s_4} CityName,$$
$$\vdash IPAddress \otimes CityName \multimap_{s_5} CountryName,$$

A selection of possible compositions for description s_2 are represented by the following operation sequences:

1. getWeather
2. getCities;getWeather
3. getGeoIP;getCities;getWeather

We have to emphasise that none of the inputs of the required Web service is mandatory for the required service and they serve as a list of potential inputs for a composite Web service. However, the identified output is mandatory. The composite Web service could have other outputs besides the mandatory one. Thus constructed composite Web services typically involve much less inputs and more outputs than identified initially.

From constructed plans, redundant operations were removed. Redundant operations are operations, which do not contribute to achieving a determined output. They are typically included into composite Web services as side-effects. An example of redundant operations is a Web service operation, which does not have any inputs, but returns current date, for instance. Moreover, the current date is not used as an input to other Web service operations. Due to our composition method, redundant operations are often included in resulting composite Web services.

We repeated the procedure, both in forward- and backward-chaining manner, for 3 domains: commercial Web services, X-Road Web services and the merged domain consisting both former domains. While separate analysis of commercial and X-Road Web services allowed to analyse the general characteristics of governmental and commercial Web services, analysis of the merged domain allowed to analyse interactions and potential synergy between commercial and governmental Web services.

The analysis should answer to the following questions:

1. which composite Web services can be composed
2. which Web service operations are most popular in possible composite Web services
3. which Web service operations are not included in any potential composite Web service
4. which Web service operations from two different domains exist together in composite Web services
5. what is the maximum, the average and the mean length of possible compositions
6. which data is most common for inputs and outputs of composite Web services
7. which outputs are achievable through composition

These questions help to answer for instance to the following questions in industry:

1. whether to implement a new Web service from scratch or to construct a composite one

2. which Web services are currently most applicable
3. which new Web services could be developed
4. how to increase the applicability of existing Web services
5. which commercial Web services could be useful for governmental Web services
6. which governmental Web services could be useful for commercial Web services
7. which data is currently most popular/easily available
8. how long workflows could be constructed automatically

After the experiments were conducted, we analysed them to answer to previously listed questions. The analysis and results are presented in Section 11.7.

11.7 Experimental results

In this section we analyse, which semantic data types were most popular for inputs and outputs of constructed composite Web services. While most popular inputs represent data, which is mostly required for computing the requested output, most common outputs represent data, which is computed usually as a side-effect to the requested output.

We also identify, which Web service operations were most popular in composite solutions. This determines currently most applicable Web services. Additionally we summarise average, mean and the longest length of constructed composite Web services.

It has to be mentioned that due to the depth-first search and limited composition time (maximum 10 seconds for each required Web service specification), Web service operations with smaller sequence index may have been preferred in composite solutions during automated Web service composition. Anyway, despite of this nuance, the results presented in this section still give an approximate estimation of most popular Web services.

11.7.1 Commercial Web services

For 163 Web service specification 678 solutions were found. The longest composite Web service involved 5 Web service operations. Average composition length was 1.95, mean length was 2. Composite solution lengths are summarised in Figure 11.9.

Most popular Web service operations returned a location according to computer's IP address and handled geographical information. Next came operations designed for verifying and determining postal codes. Interestingly, there was also an operation taking a site address and returning a password for accessing this site. Top 10 of most popular commercial Web service operations are listed in Table 11.5. In the table columns "Inputs" and "Outputs" list respectively inputs and outputs of a particular Web service

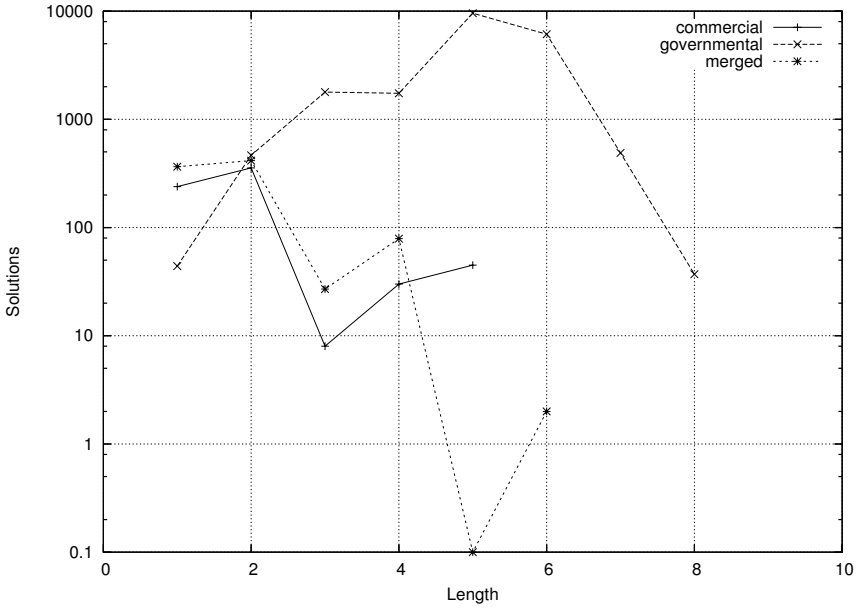


Figure 11.9: Solution lengths.

operation. Column "No." represents the number of composite Web services where an operation was presented.

Most commonly used input data was user name and password, since they are currently used to control access to Web services. Additionally license keys are used to control access to Web services. Table 11.6 summarises and orders data, which occurred in inputs and outputs of constructed composite Web services. We should emphasise that data names do not refer to any specific data structure—the names refer to the semantics of particular data.

11.7.2 Governmental Web services

For 262 Web service specification 20247 solutions were found. A reason for such a huge number of solutions, compared to the commercial Web service domain, is that governmental Web services are more homogeneous and thus can be combined in larger extent. The longest composite Web service involved 8 Web service operations. Average composition length was 5.02, mean length was 5. Composite solution lengths are summarised in Figure 11.9.

Most commonly used input data was national identification code. Additionally location-related data was important. Table 11.7 summarises and orders data, which occurred in inputs and outputs of constructed composite Web services.

Table 11.5: Most applicable commercial Web service operations.

Inputs	Outputs	No.
IPAddress, LicenseKey	CountryCode	87
CountryCode	CountryName	84
CountryName	CityNames	66
URL	Password	29
PostalCode, Radius	PostalCode	28
PostalCode, Radius, PlaceName	PostalCode	28
CityName, County, PostalCode	AddressValidity	23
—	Date, Time	22
CityName, StateCode	PostalCode	22
CityName	PostalCode	19

Table 11.6: Most popular data in commercial composite Web services.

Input name	No.	Output name	No.
Password	159	CountryName	50
Username	127	PostalCode	38
PostalCode	125	OMSResult	37
IPAddress	87	MessageSent	30
LicenseKey	87	Time	28
Date	84	Rate	25
CityName	66	AddressValidity	23
Radius	61	CityName	21
StockSymbol	45	Distance	17
SiteId	41	Chart	17
Value	39	LocationInfo	16
URL	39	Weather	15
Currency	38	NetPresentValue	14
StateCode	34	TimeZoneCode	13
RateType	33	CreditCardProc	12
County	31	USAreaCode	12
PlaceName	30	Location	11
Name	28	InterestRate	11
EmailAddress	28	ZIPCodeInfo	10
Distance	27	IndexQuote	10

Table 11.7: Most popular data in governmental composite Web services.

Input name	No.	Output name	No.
NationalIdCode	23128	Year	418060
NumberOfAnswers	6680	Day	250578
CityName	6651	Month	248717
PersonRole	5741	Language	124243
OwnerIdCode	5674	StudyForm	86386
PersonStatus	5567	StudyProgram	85382
LandRegistryIndication	4959	School	82146
EmailAddress	4787	FinancingSource	82146
CompanyName	4347	LastName, BusinessName	76575
EHAKParishCode	3765	PropertyOwner	73282
EHAKCountyCode	3739	PostalCode	72164
AdministrativeUnitName	3450	ApartmentNumber	71181
Place	3341	StreetName	68959
RegistryIdCode	2319	HouseNumber	68959
AdministrativeUnit	2244	CountryName	68518
QueryType	2213	County	44681
DrivingLicenseNumber	1971	Gender	44006
DocumentType	1769	HighSchoolLastClass	43266
DocumentSerie	1672	GroundSchoolLastClass	43266
MaxAnswers	1627	Citizenship	42051

Table 11.8: Most applicable Web service operations in the merged domain.

Inputs	Outputs	No.
IPAddress, LicenseKey	CountryCode	100
CountryCode	CountryName	98
13 inputs	60 outputs	47
—	Time, Year, Month, Day	40
—	Year, Month, Day	35
PostalCode, Radius, PlaceName	PostalCode	34
PostalCode, Radius	PostalCode	34
PostalCode, Distance	PostalCode	33
17 inputs	64 outputs	30
CountryName	CityNames	29

11.7.3 Merged commercial and governmental Web services

For 416 Web service specification 889 solutions were found. The longest composite Web service involved 6 Web service operations. Average composition length was 1.81, mean length was 2. Composite solution lengths are summarised in Figure 11.9.

The maximum length of 6 and mean length of 2 operations in a composite Web service could be due to several factors:

1. small domain size
2. large amount of partially identical Web service operations
3. limitations of the composition algorithm
4. limitations of automated Web service composition in general

Most popular Web service operations considered either geographical or postal information. Next came operations designed for verifying a postal address and some governmental Web service operations providing company information. Additionally there were operations for fetching e-mail, processing credit cards and general Internet search. Top 10 of most popular commercial Web service operations are listed in Table 11.8. Web service operations at positions 3 and 9 were database queries to the national business registry. Since the number of inputs and outputs to governmental Web service operations is large, we only identified the number of inputs and outputs.

Most commonly used input data were postal code, names and registry codes, while date-related data was the most popular output. Table 11.9 summarises and orders data, which occurred in inputs and outputs of constructed composite Web services.

Table 11.9: Most popular data of composite Web services in the merged domain.

Input name	No.	Output name	No.
PostalCode	250	Year	1106
LastName, BusinessName	238	Day	1012
RegistryIdCode	214	Month	1004
ApartmentNumber	171	LastName, BusinessName	353
PropertyOwner	155	PostalCode	314
StreetName	154	ApartmentNumber	245
HouseNumber	150	StreetName	224
CityName	137	HouseNumber	220
Year	134	PropertyOwner	218
Password	128	RegistryIdCode	189
Username	124	CountryName	152
Month	124	PropertyValue	142
Day	101	Message	125
IPAddress	100	OwnerNationalIdCode	108
LicenseKey	100	BusinessStatus	106
NumberOfAnswers	84	CityName	105
PersonRole	79	BusinessArea	104
Radius	72	LegalForm	102
EmailAddress	70	Currency	102
OwnerNationalIdCode	65	AdministrativeUnitLevel	98

11.7.4 Synergy between commercial and governmental Web services

Throughout our experiments we recorded commercial Web service operations that were used together with governmental ones in composite Web services. Altogether 25 out of 493 commercial Web service operations were applied together with governmental Web service operations. These operations are summarised in Table 11.10.

11.8 Summary

In this chapter we analysed the general structure of Web services. The analysis gave a rough estimation to the reusability of existing semantic annotations. Additionally we analysed general differences between commercial and governmental Web services. It turns out that governmental Web services are more data-intensive compared to commercial ones. Having an overview of general characteristics of Web services would greatly improve design of new annotation methods, while knowledge of the presented challenges would contribute to the design of annotation environments.

Then we extracted and analysed the most important semantic data objects in the available Web services. We also provided an overview of the available Web services. Although the analysed Web services operations and concepts are quite representative, we reviewed only a limited number of Web services compared to what is currently available on the Web. More thorough analysis needs advanced automated methods for Web services' annotation and analysis.

We also presented a methodology for analysing Web services' domains through automated Web service composition. The methodology also allows to analyse interactions between Web services' domains and individual Web services. The methodology provides methods for evaluating uniqueness, applicability and other properties of Web services. While our representative set of governmental Web services was selected from X-Road [143] project, commercial Web services were retrieved through Google API.

By applying this methodology we analysed interaction and potential synergy between commercial and governmental Web services using Web service composition approach. The main conclusion is that although most of the Web services were developed independently of eachother there is great opportunity for their composition. Such a composition is possible not only within commercial or governmental services but also across the border in a merged domain.

Another conclusion is that composition methods are fruitful for Web services analysis. They provide a set of options for further analysis employing deeper domain knowledge. The analysis may suggest which new services should be implemented to take advantage of currently available ones, or how to extend the usage of existing services.

There are yet no common guidelines for evaluating Web services composability within a domain. This chapter suggests some aspects of composition and also how to

Table 11.10: Commercial services, which were applicable with governmental ones.

Inputs	Outputs
IPAddress, LicenseKey	CountryCode
CountryCode	CountryName
CountryName	CityNames
PostalCode, Distance	PostalCodes
EncodedString	String
PostalCode, Radius	PostalCodes
PostalCode, Radius, PlaceName	PostalCodes
Username, Password, MessageNumber, POP3Server, POP3ServerPort	Message
PostalCode	CityName
HostName	IPAddress
—	Time, Year, Month, Day
—	Year, Month, Day
—	CountryNames
CityName, StateCode	PostalCode
Username, Password	Message
Username	EmailAddress, Username
CityName, StateName	PostalCode
CityName	PostalCode
Year	Year, Month, Day
County, StateCode	PostalCodes
USAreaCode	PostalCodes
TimeZoneCode	PostalCodes
CityName, StateCode	PostalCodes
Location	PostalCode
Currency	CountryName

measure them. Anyway, a set of common guidelines would allow deeper analysis of Web services' domains and composition algorithms.

Part IV

Synopsis

Chapter 12

Conclusions

12.1 Summary of results and contributions

The main contributions of this thesis are summarised in the following sections.

12.1.1 Partial deduction for linear logic

In this thesis we formalised PD for LL. More specifically, we introduced PD steps for basic forward-/backward-chaining, handling nondeterministic choices and unbounded access to resources. We also extended the framework with first-order PD steps and thus increased expressiveness of offers. As a result, the proposed formalism allows reasoning about resource dependencies and thus makes it possible to describe distributed rational systems.

We defined PD steps as special LL inference figures. While applying these inference figures during proof search instead of basic LL rules we can gain higher efficiency compared to pure LL theorem proving. Indeed, our inference figures could be seen as a sort of domain-dependent search heuristics. Finally we proved that our PD formalism is sound and complete.

12.1.2 Symbolic negotiation

We formalised symbolic negotiation and cooperative problem solving (CPS) with respect to partial deduction. We also sketched soundness and completeness proofs for these formalisations. Additionally we formalised the process of coalition formation and analysed its effect to symbolic negotiation and CPS. The analysis emphasised that coalition formation should be considered with great care. Moreover, coalitions should be avoided in the general case, if agents can solve their problems alone.

12.1.3 MAS architecture

In order to apply our symbolic negotiation formalisation, we described a MAS architecture. The MAS determines the protocol for agent communication. It also provides an environment to plug in new agents and communication protocols for ensuring scalability and correct functioning of the MAS.

We also implemented the MAS, by applying a previously implemented linear logic planner, RAPS, within symbolic negotiation for constructing new offers. The agent system is based on JADE and has been applied to automated distributed Web service composition.

12.1.4 P2P for symbolic negotiation

In order to increase scalability of the proposed MAS, we extended it with Chord P2P algorithms. The MAS applies P2P networking for reorganising and configuring its mediators. The main purpose of the mediators is to group agents which share a part of a domain described with a set of objects (literals). From Web service composition point of view these are agents, whose services' inputs or outputs include a common object (literal at the formalisation level). If agents have been gathered in such a way, their location over a distributed system is more efficient and reliable than in non-structured distributed systems.

Anyway, we still preserve some degree of centralisation—there are peers, which monitor the evolution of the network and try to detect and resolve anomalies and also for providing services for security and trust. Anyway indexing and search is organised in distributed manner. We also have mediators, which are Chord network nodes, which mediate messages to interested parties. Anyway, the mediators may change during P2P network evolution—a mediator peer may leave or a peer with better characteristics would be chosen for a new mediator.

Although our system can function without the P2P architecture, we believe that P2P would give some added value to our MAS, especially when it comes to balancing message load between agents. In fact, our empirical/analytical results show that in a system with an increasing number of Semantic Web services and agents, our P2P approach would mean almost the same message load as a system with mediator-based multicast. However, with P2P architecture message load between agents is balanced more evenly compared to a system with multicast, where all messages are routed through a central mediator. Additionally, the usage of P2P would eliminate the central point of failure in the whole system.

12.1.5 Gap detection

We also applied partial deduction for determining possible missing Web services and for identifying possible inconsistencies in Web service descriptions. These issues,

despite of being important for practical systems, have not yet been considered in the literature of automated Web service composition.

The described gap detection heuristics can be applied also in a more generic setting. Namely, for analysing which Web services in a particular *domain* could have large potential for exploitation but have not been implemented yet. This information would be especially useful for Web services provision companies, since it identifies new market opportunities.

12.1.6 Implementation of an automated composition tool

In order to analyse better the limitations and advantages of automated Web service composition, we implemented a tool, which demonstrates partially our vision of automated Web service composition. Given that the semantics of Web services is represented with OWL-S or WSMO-like ontologies, their descriptions are first translated to LL formulae. As the next step the descriptions are forwarded to agents. Then agents employ symbolic negotiation for composing new Web services according to their requirements. This approach leads to distributed composition of Web services.

12.1.7 Applicability of automated composition

In this thesis we analysed the general structure of Web services. The analysis gave a rough estimation of the reusability of existing semantic annotations. Additionally we analysed general differences between commercial and governmental Web services. While our representative set of governmental Web services was selected from X-Road [143] project, commercial Web services were retrieved through Google API.

Furthermore, we extracted and analysed most important semantic data objects in the currently available Web services and provided an overview of the currently available Web services. It turns out that governmental Web services are more data-intensive compared to commercial ones. Having an overview of general characteristics of Web services would greatly improve design of new annotation methods.

We also presented a methodology for analysing Web services' domains through automated Web service composition. By using this methodology we analysed interaction and potential synergy between commercial and governmental Web services. Moreover, applicability of particular Web services was measured by using Web service composition approach. The analysis may suggest which new services should be implemented to take advantage of current Web services, or how to extend the usage of existing services.

The main conclusion is that although most of the Web services were developed independently of eachother there is a great opportunity for their composition. Such a composition is possible not only within commercial or governmental services but also across the borders in a merged domain.

12.2 Answers to research questions

Our main research question was to determine how can automated Web service composition be applied in practice, and to what extent? In order to answer this general question, we tried to answer to the following research questions:

- **How could we automate Web service composition?** In Chapter 3 we defined PD as a computational formalism. In Chapter 5 we described how to represent Web services in LL. If all Web services, both existing and required ones, are described in LL, LL theorem proving or our PD formalism can be applied for automated composition.

  In this thesis we demonstrated that the usage of formal logics with high expressive power (such as LL) together with reasoning methods (PD, automated theorem proving) leads to a practical solution for automated Web service composition. Although there are other ways to automate Web service composition, our approach combines strong formal properties and high expressiveness of LL with efficient problem solving heuristics.

- **How to distribute automated Web service composition?** For distributing automated Web service composition we defined symbolic negotiation in Chapter 4 and presented a MAS in Chapter 6 for encapsulating the proposed symbolic negotiation formalism. While symbolic negotiation determines how to apply PD in distributed environments, the MAS determines supporting entities and protocols for implementing symbolic negotiation. Therefore, if PD can be applied for automated composition, symbolic negotiation together with a MAS can be applied for *distributed* automated composition.

- **How to extend Web services to support automated Web service composition?** As already discussed in Chapter 11 and Chapter 5, industrial standards like WSDL do not directly support automated composition. We also have to have mappings from syntax-oriented WSDL to semantic-oriented description languages in order to apply automated composition. Initial efforts addressing this issues have led to WSDL-S standard, which allows to attach semantic annotations into WSDL documents.

  Another issue, which would significantly enhance the usage of automated composition, is related to modelling effects of Web services to the environment. Currently in WSDL documents only inputs and outputs of Web service operations are described. However, preconditions and effects of Web service operations are not described. Hence, if preconditions and effect would be described, composite Web services with higher quality could be developed. Fortunately, these issues can be solved by using WSMO/WSML for modelling Web services.

- **Which industrial problems automated Web service composition can solve?** The initial goal of automated composition was to provide methods for dynamic

reconfiguration of information systems. However, as demonstrated in Chapter 10, automated composition can be used for generating proposals for new Web services, which have not been implemented yet. Moreover, as it has been demonstrated in Chapter 11, automated composition can be applied for analysing which kind of data and Web services are most relevant in particular domains.

- **What are the limitations of automated Web service composition?** As summarised in Chapter 11, automated composition constructs relatively short and simple solutions. This finding is supported by Estublier and Sanlaville [49], who argue that composite Web services can express only simple business processes. Hence the main disadvantage of automated composition, as we consider it in this thesis, is that it constructs too simple workflows.

12.3 Future work

Future research may include extending our method for service composition to also exploit business models, which specify additional relationships between services. Future work related to partial deduction, symbolic negotiation, P2P networks and Web services are enlisted in the following sections.

12.3.1 Partial deduction and symbolic negotiation

It has been indicated [125] that modal logic S4 has a direct translation to LL. This result leads us to question whether current BDI-theories could be embedded into our symbolic negotiation framework. Embedded BDI theories may help to specify explicitly agent negotiation strategies.

In future work we would also like to study different properties of symbolic negotiation. Moreover, we would like to introduce additional symbolic negotiation rules. Then, by considering symbolic negotiation rules as PD strategies, we are interested in determining efficient strategies for various PD tasks.

In the framework of Contract Net protocol [40] we could apply symbolic negotiation for task decomposition and Contract Net itself for non-symbolic negotiation. Embedding non-symbolic negotiation into symbolic negotiation is appealing since it may combine the advantages of both frameworks while reducing the effect of their disadvantages. This is also one way to implement hybrid negotiation systems.

12.3.2 Symbolic negotiation in P2P networks

Our P2P architecture assumes currently that there exists a function for transforming semantic concepts, like concepts in ontologies, to unique keys such that the concepts with the same meaning have the same key. In the current implementation this is achieved by constraining the agents to use the same ontology. Anyway, we recognise need to

an interaction between different ontologies, since in large P2P networks different communities tend to use different ontologies.

In order to facilitate semantic reasoning during the composition process, we would like to design a function, which would map objects with the similar meaning to a similar integer key. This would allow us to be sure that the objects/concepts with the same meaning are in the same neighborhood. One possible solution has been proposed by Tang et al [183] who consider semantics in P2P systems. They adopt Latent Semantic Indexing (LSI) for information retrieval in Content-Addressable Networks (CAN). The semantics of documents (which could represent concepts in ontologies) is described with sets of keywords. Instead of LSI possibly some other information retrieval algorithm [155] could be applied as well.

Alternatively, if objects have been annotated with keywords, Hilbert space filling curves [166] (SFC) could be applied for mapping an n-dimensional keyword space to 1-dimensional hash value space. This approach has been used by Schmidt and Parashar [168] for locating Web services at Chord P2P network. Unfortunately we could not apply their results in our system, since Schmidt and Parashar described Web service classification with keywords, while we need to annotate the inputs and outputs of Web services. Nonetheless, there may be another use for SFC-s within our research.

Finally, we are also considering options for mapping the entire structure of LL formulae to P2P networks instead of literals only. Such a method would obviously allow us to exploit richer structural semantics of Web services already at P2P level.

12.3.3 Web services descriptions

Web services standards are currently far from perfect. For instance, Van der Aalst et al [190] evaluate the expressiveness of Web service composition languages BPEL4WS, XLANG, WSFL, BPML and WSCI using both workflow and communication patterns. An in-depth analysis of BPEL4WS language and its control structures is performed by Wohed et al [197]. Both studies identify that there is room for further progress in Web services orchestration standards. Some tools related to Web services' orchestration are reviewed by Peltz [151].

Kuno and Sahai [107] on the other hand state the agents' inability to use UDDI registries due to the static content of the latter and most importantly, lack of machine-readable semantic information. It is also emphasised that in the Semantic Web context agents should be able to discover services that are appropriate given customer's preferences and requirements. Therefore, our future work could be focused towards embedding semantics into UDDI registries.

12.3.4 Automated Web service annotation

Although the set of Web services' operations and concepts analysed in this thesis is quite representative, we reviewed only a limited number of Web services compared to

what is currently available on the Web. More thorough analysis to determine more advanced automated methods for Web services' annotation and analysis should be undertaken. Therefore our future research focus is biased towards providing such automated methods and tools.

We are looking forward to explore further the research by Lister et al [118] and Ernst et al [48], who aim to extract the semantics of inputs and outputs of Web services from the data produced by particular Web services. By combining descriptions of Web services in WSDL documents with data, which they produce, annotations of higher quality may be constructed automatically.

In the future we would like to implement a method for automated annotation of Web services. The method would take into account general results presented in this thesis and encountered challenges to provide an efficient and user-friendly environment for annotation. Finally, the method would be incorporated into our annotation tool to facilitate higher productivity and efficiency of the analysis process.

Appendix A

Abbreviations

Table A.1: Abbreviations used in the thesis.

Abbreviation	Full name
ACL	Agent Communication Language
AI	Artificial Intelligence
API	Application Program Interface
BPEL	Business Process Execution Language
BPEL4WS	BPEL for Web Services
CPS	Cooperative Problem Solving
DAML	DARPA Agent Markup Language
DHT	Distributed Hashtables
FIPA	Foundation for Intelligent Physical Agents
GUI	Graphical User Interface
LL	Intuitionistic Linear Logic
IS	Information System
KQML	Knowledge Query and Manipulation Language
LL	Linear Logic
MAS	Multi-Agent System
OWL	Web Ontology Language
OWL-S	OWL-based Web service ontology
P2P	Peer to Peer
PD	Partial Deduction
QoS	Quality of Service
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
SOAP	Simple Object Access Protocol
SSP	Structural Synthesis of Programs

SWSO	Semantic Web Services Ontology
UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Identifiers
W3C	World Wide Web Consortium
WSDL	Web Service Description Language
WSDL-S	WSDL Semantics
WSML	Web Service Modeling Language
WSFL	Web Services Flow Language
WSMO	Web Service Modeling Ontology
XML	eXtensible Markup Language

Appendix B

Logic Rules

B.1 Rules of intuitionistic LL

Logical axiom and Cut rule:

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

Multiplicative connectives:

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

$$\frac{}{\vdash 1} \ R1 \qquad \frac{\Gamma \vdash A}{\Gamma, 1 \vdash A} \ L1$$

$$\frac{\Sigma_1 \vdash A \quad B, \Sigma_2 \vdash C}{\Sigma_1, (A \multimap B), \Sigma_2 \vdash C} \ L\multimap \qquad \frac{\Sigma, A \vdash B}{\Sigma \vdash (A \multimap B)} \ R\multimap$$

Additive connectives:

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

$$\textit{No R0 rule} \qquad \frac{}{\Gamma, 0 \vdash A} \ L0$$

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

$$\frac{}{\Gamma \vdash \top}\, R\top \qquad \textit{No } L\top \textit{ rule}$$

Rules for the exponential !:

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

Rules for quantifiers:

$$\frac{\Gamma, A[a/x] \vdash \Delta}{\Gamma, \forall x A \vdash \Delta}\, L\forall \qquad \frac{\Gamma \vdash \Delta, A[t/x]}{\Gamma \vdash \Delta, \forall x A}\, R\forall$$

$$\frac{\Gamma, A[t/x] \vdash \Delta}{\Gamma, \exists x A \vdash \Delta}\, L\exists \qquad \frac{\Gamma \vdash A[a/x], \Delta}{\Gamma \vdash \exists x A, \Delta}\, R\exists$$

where t is not free in Γ and Δ.

Bibliography

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

[2] P. Adjiman, P. Chatalic, F. Goasdoué, M.-C. Rousset, and L. Simon. Distributed reasoning in a peer-to-peer setting. Technical report, LRI, Université Paris Sud, France, 2004.

[3] L. Amgoud, S. Parsons, and N. Maudet. Arguments, dialogue and negotiation. In *Proceedings of 14th European Conference on Artificial Intelligence, Berlin, Germany, August 20–25, 2000*, pages 338–342. IOS Press, 2000.

[4] J.-M. Andreoli, R.Pareschi, and T. Castagnetti. Static analysis of linear logic programming. *New Generation Computing*, 15:449–481, 1997.

[5] A. Andrzejak and Z. Xu. Scalable, efficient range queries for grid information services. In *Proceedings of the Second IEEE International Conference on Peer-to-Peer Computing (P2P 2002), Linköping, Sweden, September 5–7, 2002*, pages 33–40. IEEE Computer Society Press, 2002.

[6] N. Antonopoulos and J. Salter. Efficient resource discovery in grids and P2P networks. *Internet Research*, 14(5):339–346, 2004.

[7] N. Antonopoulos, J. Salter, and R. Peel. A multi-ring method for efficient multi-dimensional data lookup in P2P networks. In *Proceedings of the 2005 International Conference on Foundations of Computer Science (FCS'05), Monte Carlo Resort, Las Vegas, Nevada, USA, June 27–30, 2005*, 2005.

[8] L. Ardissono, A. Goy, and G. Petrone. Enabling conversations with Web services. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2003, July 14–18, 2003, Melbourne, Victoria, Australia*, pages 819–826. ACM Press, 2003.

[9] I. B. Arpinar, B. Aleman-Meza, R. Zhang, and A. Maduko. Ontology-driven Web services composition platform. In *Proceedings of IEEE International Conference on E-Commerce Technology, CEC'04, San Diego, California, USA, July 6–9, 2004*, pages 146–152. IEEE Press, 2004.

[10] A.-L. Barabasi. *Linked: How Everything Is Connected to Everything Else and What It Means*. Plume, 2003.

[11] M. Bawa, G. S. Manku, and P. Raghavan. SETS: Search enhanced by topic segmentation. In *Proceedings of 26th Annual International ACM SIGIR Conference (SIGIR 2003), Toronto, Canada, July 28–August 1, 2003*, pages 306–313. ACM Press, 2003.

[12] G. Bellin and P. J. Scott. On the pi-calculus and linear logic. *Theoretical Computer Science*, 135(1):11–65, 1994.

[13] B. Benatallah, M. Dumas, Q. Z. Sheng, and A. H. H. Ngu. Declarative composition and peer-to-peer provisioning of dynamic Web services. In *Proceedings of the 18th International IEEE Conference on Data Engineering, ICDE'02, San Jose, USA, February 2002*, pages 297–308, 2002.

[14] D. Berardi, D. Calvanese, G. de Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of e-services that export their behavior. In *Proceedings of the First International Conference on Service-Oriented Computing, ICSOC 2003, Trento, Italy, December 15–18, 2003*, volume 2910 of *Lecture Notes in Computer Science*, pages 43–58. Springer-Verlag, 2003.

[15] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[16] W. Bibel. A deductive solution for plan generation. *New Generation Computing*, 4:115–132, 1986.

[17] W. Bibel. Let's plan it deductively. *Artificial Intelligence*, 103:183–208, 1998.

[18] N. Biri and D. Galmiche. A modal linear logic for distribution and mobility (abstract). In *Proceedings of International Workshop on Linear Logic, Copenhagen, Denmark, July 2002*, 2002.

[19] S. Biundo, D. Dengler, and J. Koehler. Deductive planning and plan reuse in a command language environment. In *Proceedings of the 10th European Conference on Artificial Intelligence, Vienna, Austria, August 1992*, pages 628–632, 1992.

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

[21] D. G. Bobrow and M. J. Stefik. Perspectives on Artificial Intelligence programming. *Science*, 231(4741):951–956, 1986.

[22] M. P. Bonacina. A taxonomy of parallel strategies for deduction. *Annals of Mathematics and Artificial Intelligence*, 29(1–4):223–257, 2000.

[23] B. Bonet and H. Geffner. Planning as heuristic search: New results. In S. Biundo and M. Fox, editors, *Recent Advances in AI Planning. Proceedings of the 5th European Conference on Planning (ECP'99), Durham, UK, September 1999*, volume 1809 of *Lecture Notes in Artificial Intelligence*, pages 360–372. Springer-Verlag, 2000.

[24] M. Bozzano, G. Delzanno, M. Martelli, V. Mascardi, and F. Zini. Logic programming & multi-agent systems: a synergic combination for applications and semantics. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 5–32. Springer-Verlag, 1999.

[25] M. E. Bratman. *Intentions, Plans, and Practical Reason*. Harvard University Press, 1987.

[26] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, and J. W. R. Stata, A. Tomkins. Graph structure in the Web. *Computer Networks*, 3:309–320, 2000.

[27] J. Broekstra, M. Ehrig, P. Haase, F. van Harmelen, A. Kampman, M. Sabou, R. Siebes, S. Staab, H. Stuckenschmidt, and C. Tempich. A metadata model for semantics-based peer-to-peer systems. In *Proceedings of the WWW'03 Workshop on Semantics in Peer-to-Peer and Grid Computing. Budapest, Hungary, May 20, 2003*, 2003.

[28] M. Broy and P. Pepper. Program development as a formal activity. *IEEE Transactions on Software Engineering*, 7(1):14–22, 1981.

[29] S. Brüning, S. Hölldobler, J. Schneeberger, U. Sigmund, and M. Thielscher. Disjunction in resource-oriented deductive planning. Technical Report AIDA-93-03, Technische Hochschule Darmstadt, Germany, 1994.

[30] T. Bultan, X. Fu, R. Hull, and J. Su. Conversation specification: A new approach to design and analysis of e-service composition. In *Proceedings of 12th International World Wide Web Conference (WWW'03), Budapest, Hungary, May 20–24, 2003*, pages 403–410, 2003.

[31] M. Burstein. Ontology mapping for dynamic service invocation on the Semantic Web. In *AAAI Spring Symposium on Semantic Web Services, Palo Alto, March, 2004*, 2004.

[32] L. Cabral, J. Domingue, E. Motta, T. Payne, and F. Hakimpour. Approaches to Semantic Web services: An overview and comparisons. In *Proceedings of the First European Semantic Web Symposium (ESWS2004), Heraklion, Crete, Greece, May 10–12, 2004*, volume 3053 of *Lecture Notes in Computer Science*, pages 225–239. Springer-Verlag, 2004.

[33] L. Cardelli and A. D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[34] F. Casati, S. Ilnicki, L.-J. Jin, V. Krishnamoorthy, and M.-C. Shan. Adaptive and dynamic service composition in eFlow. In *Proceeding of 12th Int. Conference on Advanced Information Systems Engineering (CAiSE 2000), Stockholm, Sweden, June 5–9, 2000*, volume 1789 of *Lecture Notes in Computer Science*, pages 13–31. Springer-Verlag, 2000.

[35] P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(3):213–261, 1990.

[36] P. R. Cohen and H. J. Levesque. Teamwork. *Nous*, 25(4):487–512, 1991.

[37] A. Crespo and H. Garcia-Molina. Semantic overlay networks for P2P systems. Technical report, Department of Computer Science, Yale University, 2002.

[38] S. Cresswell, A. Smaill, and J. Richardson. Deductive synthesis of recursive plans in linear logic. In *Proceedings of the Fifth European Conference on Planning (ECP'99), Durham, United Kingdom, September 8–10, 1999*, pages 252–264, 1999.

[39] J. Darlington. An experimental program transformation and synthesis system. *Artificial Intelligence*, 16:1–46, 1981.

[40] R. Davis and R. G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20:63–109, 1983.

[41] L. de Alfaro and T. A. Henzinger. Interface automata. In *Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ESEC 2001, Vienna, Austria, September 10–14, 2001*, pages 109–120. ACM Press, 2001.

[42] D. de Schreye, R. Glück, J. Jørgensen, M. Leuschel, B. Martens, and M. H. Sørensen. Conjunctive partial deduction: Foundations, control, algorithms and experiments. *Journal of Logic Programming*, 41(2–3):231–277, 1999.

[43] M. de Weerdt, A. Bos, H. Tonino, and C. Witteveen. A resource logic for multi-agent plan merging. *Annals of Mathematics and Artificial Intelligence*, 37(1–2):93–130, 2003.

[44] I. Dickinson and M. Wooldridge. Towards practical reasoning agents for the Semantic Web. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2003, July 14–18, 2003, Melbourne, Victoria, Australia*, pages 827–834. ACM Press, 2003.

[45] M. B. Do and S. Kambhampati. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artificial Intelligence*, 132:151–182, 2001.

[46] U. Engberg and G. Winskel. Completeness results for linear logic on Petri nets. *Annals of Pure and Applied Logic*, 86:101–135, 1997.

[47] V. Ermolayev, N. Keberle, O. Kononenko, S. Plaksin, and V. Terziyan. Towards a framework for agent-enabled Semantic Web service composition. *International Journal of Web Services Research*, 1(3):63–87, 2004.

[48] M. D. Ernst, R. Lencevicius, and J. H. Perkins. Detection of Web service substitutability and composability. In *Proceedings of International Workshop on Web Services Modeling and Testing (WS-MaTe2006), Palermo, Sicily, Italy, June 9, 2006*, pages 123–135, 2006.

[49] J. Estublier and S. Sanlaville. Business processes and workflow coordination of web services. In *Proceedings of 2005 IEEE International Conference on e-Technology, e-Commerce, and e-Services, EEE'2005, Hong Kong, China, 29 March–1 April, 2005*, pages 85–88. IEEE Computer Society, 2005.

[50] P. Fabiani and Y. Meiller. Planning with tokens: An approach between satisfaction and optimisation. In *Proceedings of the 14th Workshop "New Results in Planning, Scheduling and Design" (PuK2000), Berlin, August 21–22, 2000, in conjunction with 14th European Conference on Artificial Intelligence (ECAI 2000), August 20–25, 2000, Berlin, Germany*, pages 26–35, 2000.

[51] M. Fisher. Characterising simple negotiation as distributed agent-based theorem-proving—a preliminary report. In *Proceedings of the Fourth International Conference on Multi-Agent Systems, Boston, MA, July 10-12, 2000*, pages 127–134. IEEE Press, 2000.

[52] M. Fisher and M. Wooldridge. Distributed problem-solving as concurrent theorem proving. In *Proceedings of 8th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Ronneby, Sweden, May 13-16, 1997*, volume 1237 of *Lecture Notes in Computer Science*, pages 128–140. Springer-Verlag, 1997.

[53] D. Fuchs. Requirement-based cooperative theorem proving. In *Proceedings of JELIA-1998, Dagstuhl, Germany, October 12–15, 1998*, volume 1489 of *Lecture Notes in Artificial Intelligence*, pages 139–153. Springer-Verlag, 1998.

[54] J. A. Giampapa and K. Sycara. Conversational case-based planning for agent team coordination. In *Proceedings of the Fourth International Conference on Case-Based Reasoning, ICCBR 2001, Vancouver, BC, Canada, 30 July–2 August, 2001*, volume 2080 of *Lecture Notes in Artificial Intelligence*, pages 189–203. Springer-Verlag, 2001.

[55] N. Gibbins, S. Harris, and N. Shadbolt. Agent-based Semantic Web services. In *Proceedings of the Twelfth International World Wide Web Conference, WWW2003, Budapest, Hungary, May 20–24, 2003*, pages 710–717. ACM Press, 2003.

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

[57] A. Gómez-Pérez, R. González-Cabero, and M. Lama. A framework for design and composition of Semantic Web services. In *Proceedings of the First International Semantic Web Services Symposium, AAAI 2004 Spring Symposium Series, March 22–24, 2004*, pages 113–120. AAAI Press, 2004.

[58] G. Governatori, A. H. M. ter Hofstede, and P. Oaks. Defeasible logic for automated negotiation. In *Proceedings of the 5th CollECTeR Conference on Electronic Commerce, Brisbane, Australia, December 13–14, 2000*. Deakin University, 2000. On CD-ROM.

[59] C. Green. Application of theorem proving to problem solving. In *Proceedings of First International Joint Conference on Artificial Intelligence (IJCAI-69), Washington, DC, May 7–9, 1969*, pages 219–239, 1969.

[60] G. Grosse, S. Hölldobler, and J. Schneeberger. Linear deductive planning. *Journal of Logic and Computation*, 6:232–262, 1996.

[61] B. Grosz and S. Kraus. Collaborative plans for complex group actions. *Artificial Intelligence*, 86:269–357, 1996.

[62] R. Guha. Semantic negotiation: Co-identifying objects across data sources. In *Proceedings of the First International Semantic Web Services Symposium, AAAI 2004 Spring Symposium Series, March 22–24, 2004*, pages 7–12. AAAI Press, 2004.

[63] M. Harf and E. Tyugu. Algorithms of structured synthesis of programs. *Programming and Computer Software*, 6:165–175, 1980.

[64] J. Harland, D. Pym, and M. Winikoff. Forward and backward chaining in linear logic. In *Proceedings of the CADE-17 Workshop on Proof-Search in Type-Theoretic Systems, Pittsburgh, June 20–21, 2000*, volume 37 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2000.

[65] J. Harland and M. Winikoff. Agent negotiation as proof search in linear logic. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002), July 15–19, 2002, Bologna, Italy*, pages 938–939, 2002.

[66] J. Harland and M. Winikoff. Language design issues for agents based on linear logic (extended abstract). *Electronic Notes in Theoretical Computer Science*, 70(5), 2002. Proceedings of the Workshop on Computational Logic in Multi-Agent Systems (CLIMA'02), Copenhagen, Denmark, August 1, 2002.

[67] S. V. Hashemian and F. Mavaddat. A graph-based approach to Web services composition. In *Proceedings of 2005 IEEE/IPSJ International Symposium on Applications and the Internet, SAINT 2005, Trento, Italy, January 31–February 4, 2005*, pages 183–189. IEEE Computer Society, 2005.

[68] P. Haslum and H. Geffner. Heuristic planning with time and resources. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01) Workshop on Planning with Resources*, 2001.

[69] J. Hendler. Agents and the Semantic Web. *IEEE Intelligent Systems*, 16(2):30–37, 2001.

[70] A. Heß, E. Johnston, and N. Kushmerick. Assam: A tool for semi-automatically annotating semantic web services. In *Proceedings of the 3rd International Semantic Web Conference (ISWC 2004)*, Hiroshima, Japan, 2004.

[71] A. Heß and N. Kushmerick. Learning to attach semantic metadata to web services. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *Proceedings of the 2nd International Semantic Web Conference*, number 2870 in Lecture Notes in Computer Science, pages 258–273, Sanibel Island, Florida, USA, 2003. Springer-Verlag.

[72] T. Hirai. Propositional temporal linear logic and its application to concurrent systems. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E83-A(11):2219–2227, 2000. Special Section on Concurrent Systems Technology.

[73] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[74] K. Y. K. Hui, J. C. S. Lui, and D. K. Y. Yau. Small world overlay P2P networks. In *Proceedings of the Twelfth IEEE International Workshop on Quality of Service (IWQoS 2004), Montreal, Canada, June 7–9, 2004*, pages 201–210. IEEE Communications Society, 2004.

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

[76] É. Jacopin. Classical AI planning as theorem proving: The case of a fragment of linear logic. In *Proceedings of AAAI Fall Symposium on Automated Deduction in Nonstandard Logics*, pages 62–66. AAAI Press, 1993.

[77] N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge. Automated negotiation: Prospects, methods and challenges. *International Journal of Group Decision and Negotiation*, 10(2):199–215, 2001.

[78] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1:275–306, 1998.

[79] S. Kaffille, K. Loesing, and G. Wirtz. Distributed service discovery with guarantees in peer-to-peer networks using distributed hashtables. In *Proceedings of the 2005 International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA 2005, Las Vegas, Nevada, USA, June 27–30, 2005*, 2005.

[80] A. C. Kakas, P. Torroni, and N. Demetriou. Agent planning, negotiation and control of operation. In *Proceedings of the 16th European Conference on Artificial Intelligence, ECAI 2004, Valencia, Spain, August 22–27, 2004*, pages 28–32. IOS Press, 2004.

[81] M. Kanovich and T. Ito. Temporal linear logic specifications for concurrent processes (extended abstract). In *Proceedings of the Twelfth Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29–July 2, 1997*, pages 48–57. IEEE Computer Society Press, 1997.

[82] M. I. Kanovich. Linear logic as a logic of computations. *Annals of Pure and Applied Logic*, 67:183–212, 1994.

[83] M. I. Kanovich, M. Okada, and A. Scedrov. Specifying real-time finite-state systems in linear logic. In *Proceedings of the 2nd International Workshop on Constraint Programming for Time-Critical Applications and Multi-Agent Systems (COTIC'98), Nice, France, September 7, 1998*, volume 16 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1998.

[84] M. I. Kanovich and J. Vauzeilles. The classical AI planning problems in the mirror of Horn linear logic: Semantics, expressibility, complexity. *Mathematical Structures in Computer Science*, 11(6):689–716, 2001.

[85] R. M. Karp and R. E. Miller. Parallel program schemata. *Journal of Computer and Systems Sciences*, 3(2):147–195, 1969.

[86] H. Kautz and B. Selman. Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI-92)*, pages 359–363, 1992.

[87] H. Kautz and B. Selman. BLACKBOX: A new approach to the application of theorem proving to problem solving. In *Working notes of the Workshop on Planning as Combinatorial Search, held in conjunction with AIPS-98, Pittsburgh, PA, June 1998*, pages 58–60, 1998.

[88] S. M. Kim and M. C. Rosu. A survey of public Web services. In *Proceedings of 5th International Conference on E-Commerce and Web Technologies, EC-Web 2004, Zaragoza, Spain, August 31–September 3, 2004*, volume 3182 of *Lecture Notes in Computer Science*, pages 96–105. Springer-Verlag, 2004.

[89] J. Komorowski. *A Specification of An Abstract Prolog Machine and Its Application to Partial Evaluation*. PhD thesis, Department of Computer and Information Science, Linkoping University, Linkoping, Sweden, 1981.

[90] S. Kraus, K. Sycara, and A. Evenchik. Reaching agreements through argumentation: A logical model and implementation. *Artificial Intelligence*, 104(1–2):1–69, 1998.

[91] P. Küngas. Abstraction within partial deduction for linear logic. In *Proceedings of 7th International Conference on Artificial Intelligence and Symbolic Computation, AISC 2004, RISC, Castle of Hagenberg, Austria, September 22–24, 2004*, volume 3249 of *Lecture Notes in Artificial Intelligence*, pages 52–65. Springer-Verlag, 2004.

[92] P. Küngas. Analysing ai planning problems in linear logic—a partial deduction approach. In *Proceedings of XVII Brazilian Symposium on Artificial Intelligence, SBIA 2004, Sao Luis, Maranhao, Brazil, September 29–October 1, 2004*, volume 3171 of *Lecture Notes in Artificial Intelligence*, pages 52–61. Springer-Verlag, 2004.

[93] P. Küngas. Dynamic web service discovery and exploitation through symbolic agent negotiation. In *Proceedings of the 2nd European Starting AI Researcher Symposium, STAIRS 2004, Valencia, Spain, August 23–24, 2004, collocated with 16th European Conference on Artificial Intelligence, ECAI'04, Valencia, Spain, August 22–27, 2004*, pages 247–252. IOS Press, 2004.

[94] P. Küngas. Temporal linear logic for symbolic agent negotiation. In *Proceedings of the 8th Pacific Rim International Conference on Artificial Intelligence, PRICAI 2004, Auckland, New Zealand, August 9–13, 2004*, volume 3157 of *Lecture Notes in Artificial Intelligence*, pages 23–32. Springer-Verlag, 2004.

[95] P. Küngas and M. Matskin. Semantic web service composition through a p2p-based multi-agent environment. In *Proceedings of the Fourth International Workshop on Agents and Peer-to-Peer Computing (in conjunction with AAMAS 2005), AP2PC 2005, Utrecht, Netherlands, July 26, 2005*, Lecture Notes in Computer Science. Springer-Verlag. To appear.

[96] 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*, volume 2990 of *Lecture Notes in Artificial Intelligence*, pages 263–279. Springer-Verlag, 2004.

[97] P. Küngas and M. Matskin. Symbolic negotiation with linear logic. In *Proceedings of the Fourth International Workshop on Computational Logic in Multi-Agent Systems, CLIMA IV, Fort Lauderdale, FL, USA, January 6-7, 2004. Revised Selected and Invited Papers*, volume 3259 of *Lecture Notes in Computer Science*, pages 71–88. Springer-Verlag, 2004.

[98] P. Küngas and M. Matskin. Combining symbolic and non-symbolic negotiation for agent-based web service composition. In *Proceedings of the 2005 International Conference on Artificial Intelligence, ICAI'05, Las Vegas, Nevada, USA, June 27–30, 2005*, pages 513–519. CSREA Press, 2005.

[99] P. Küngas and M. Matskin. Detection of missing web services: The partial deduction approach. In *Proceedings of International Conference on Next Generation Web Services Practices, NWeSP'05, Seoul, Korea, August 22–26, 2005*, pages 339–344. IEEE Computer Society Press, 2005.

[100] P. Küngas and M. Matskin. Detection of missing web services: The partial deduction approach. *International Journal of Web Services Practices*, 1(1–2):133–141, 2005.

[101] P. Küngas and M. Matskin. Partial deduction for assisting automated semantic web service composition. In *Proceedings of the Workshop on Exploring Planning and Scheduling for Web Services, Grid and Autonomic Computing held in conjunction with The Twentieth National Conference on Artificial Intelligence, (AAAI 2005), Pittsburgh, Pennsylvania, USA, July 9–10, 2005*, volume WS-05-03 of *AAAI Technical Report*, pages 43–45. AAAI Press, 2005.

[102] P. Küngas and M. Matskin. Partial deduction for linear logic—the symbolic negotiation perspective. In *Proceedings of the Second International Workshop on Declarative Agent Languages and Technologies (in conjunction with AAMAS 2004), DALT'2004, New York, USA, July 19, 2004*, volume 3476 of *Lecture Notes in Artificial Intelligence*, pages 35–52. Springer-Verlag, 2005.

[103] P. Küngas and M. Matskin. Symbolic negotiation revisited. In *Proceedings of Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems, AAMAS'06, Future University-Hakodate, Japan, May 8–12, 2006*. ACM Press, 2006.

[104] P. Küngas and M. Matskin. Web services analysis: Making use of Web service composition and annotation. In *Proceedings of the First Asian Semantic Web*

Conference, ASWC'06, Beijing, China, September 3–7, 2006, Lecture Notes in Computer Science. Springer-Verlag, 2006. To appear.

[105] P. Küngas and M. Matskin. Web services roadmap: The Semantic Web perspective. In *Proceedings of International Conference on Internet and Web Applications and Services, ICIW'06, Guadeloupe, French Caribbean, February 23–25, 2006*. IEEE Computer Society Press, 2006.

[106] P. Küngas, J. Rao, and M. Matskin. 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 15–17, 2004*, volume 3129 of *Lecture Notes in Computer Science*, pages 458–467. Springer-Verlag, 2004.

[107] H. Kuno and A. Sahai. My agent wants to talk to your service: Personalizing Web services through agents. In B. Burg, J. Dale, T. Finin, H. Nakashima, L. Padgham, C. Sierra, and S. Willmott, editors, *Agentcities: Challenges in Open Agent Environments*. Springer-Verlag, 2003.

[108] Y. Labrou, T. Finin, and Y. Peng. Agent communication languages: The current landscape. *IEEE Intelligent Systems*, 14(2):45–52, 1999.

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

[110] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.

[111] H. Lehmann and M. Leuschel. Solving planning problems by partial deduction. In *Proceedings of the 7th International Conference on Logic for Programming and Automated Reasoning, LPAR'2000, Reunion Island, France, November 11–12, 2000*, volume 1955 of *Lecture Notes in Artificial Intelligence*, pages 451–467. Springer-Verlag, 2000.

[112] D. B. Lenat. Beings: Knowledge as interacting experts. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence, Tbilisi, Georgia, USSR, September 3–8, 1975*, pages 126–133, 1975.

[113] M. Leuschel and H. Lehmann. Solving coverability problems of Petri nets by partial deduction. In *Proceedings of the 2nd International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, PPDP'2000, Montreal, Canada, September 20–23, 2000*, pages 268–279. ACM Press, 2000.

[114] H. J. Levesque, P. R. Cohen, and J. H. T. Nunes. On acting together. In *Proceedings of the Eighth National Conference on Artificial Intelligence, AAAI-90, Boston, Massachusetts, July 29–August 3, 1990*, pages 94–99. AAAI Press, 1990.

[115] H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1–3):59–83, 1997.

[116] P. Lincoln. Linear logic. *ACM SIGACT Notices*, 23(2):29–37, 1992.

[117] P. Lincoln. Deciding provability of linear logic formulas. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, volume 222 of *London Mathematical Society Lecture Note Series*, pages 109–122. Cambridge University Press, 1995.

[118] K. Lister, L. Sterling, and K. Taveter. Reconciling ontological differences by assistant agents. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-06), Future University, Hakodate, Japan, May 8–12, 2006*, pages 943–945. ACM Press, 2006.

[119] S. Liu, P. Küngas, and M. Matskin. Agent-based web service composition with jade and jxta. In *Proceedings of the 2006 International Conference on Semantic Web and Web Services, SWWS'06, Las Vegas, Nevada, USA, June 26-29, 2006*. CSREA Press, 2006. To appear.

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

[121] S. Majithia, M. S. Shields, I. J. Taylor, and I. Wang. Triana: A graphical Web service composition and execution toolkit. In *Proceedings of the Second IEEE International Conference on Web Services (ICWS'04), San Diego, California, USA, June 6–9, 2004*, pages 514–521. IEEE Computer Society, 2004.

[122] S. Majithia, D. W. Walker, and W. A. Gray. A framework for automated service composition in service-oriented architectures. In *Proceedings of the First European Semantic Web Symposium, ESWS 2004, Heraklion, Crete, Greece, May 10–12, 2004*, volume 3053 of *Lecture Notes in Computer Science*, pages 269–283. Springer-Verlag, 2004.

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

[124] Z. M. Mao, R. H. Katz, and E. A. Brewer. Fault-tolerant, scalable, wide-area internet service composition. Technical Report CSD-01-1129, Computer Science Division, University of California, Berkeley, California, USA, 2001.

[125] S. Martini and A. Masini. A modal view of linear logic. *Journal of Symbolic Logic*, 59(3):888–899, 1994.

[126] M. Masseron, C. Tollu, and J. Vauzeilles. Generating plans in linear logic I–II. *Theoretical Computer Science*, 113:349–375, 1993.

[127] M. Matskin, O. J. Kirkeluten, S. B. Krossnes, and Ø. Sæle. Agora: An infrastructure for cooperative work support in multi-agent systems. In T. Wagner and O. F. Rana, editors, *International Workshop on Infrastructure for Multi-Agent Systems, Barcelona, Spain, June 3–7, 2000, Revised Papers*, volume 1887 of *Lecture Notes in Computer Science*, pages 28–40. Springer-Verlag, 2001.

[128] M. Matskin and J. Komorowski. Partial structural synthesis of programs. *Fundamenta Informaticae*, 30:23–41, 1997.

[129] M. Matskin, P. Küngas, J. Rao, J. Sampson, and S. A. Petersen. Enabling web services composition with software agents. In *Proceedings of the Ninth IASTED International Conference on Internet and Multimedia Systems and Applications, IMSA 2005, Honolulu, Hawaii, USA, August 15–17, 2005*, pages 93–98. ACTA Press, 2005.

[130] M. Matskin and E. Tyugu. Strategies of structural synthesis of programs and its extensions. *Computing and Informatics*, 20:1–25, 2001.

[131] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of Artificial Intelligence. In B. Meltzer, D. Michie, and M. Swann, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, 1969.

[132] D. McDermott. Estimated-regression planning for interaction with Web services. In *Proceedings of the 6th International Conference on AI Planning and Scheduling, Toulouse, France, April 23–27, 2002*. AAAI Press, 2002.

[133] D. McDermott. The formal semantics of processes in PDDL. In *Proceedings of ICAPS Workshop on PDDL*, 2004.

[134] S. McIlraith and T. C. Son. Adapting Golog for composition of Semantic Web services. In *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002), Toulouse, France, April 22–25, 2002*, pages 482–493. Morgan Kaufmann, 2002.

[135] D. Miller. The pi-calculus as a theory in linear logic: Preliminary results. In *Proceedings of the Third International Workshop on Extensions of Logic Programming, ELP'92, Bologna, Italy, February 26–28, 1992*, volume 660 of *Lecture Notes in Computer Science*, pages 242–264. Springer-Verlag, 1993.

[136] R. Milner. The polyadic pi-calculus: A tutorial. Technical Report ECS-LFCS-91-180, Computer Science Department, University of Edinburgh, 1991.

[137] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. Technical Report HPL-2002-57, Hewlett-Packard, 2002.

[138] K. E. Moore and S. M. Gupta. Petri net models of flexible and automated manufacturing systems: A survey. *International Journal of Production Research*, 34:3001–3035, 1996.

[139] S. Muggleton and L. de Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.

[140] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of IEEE*, 77(4):541–580, 1989.

[141] T. Murata, P. C. Nelson, and J. Yim. A predicate-transition net model for multiple agent planning. *Information Sciences*, 57–58:361–384, 1991.

[142] B. Nebel and J. Koehler. Plan reuse versus plan generation: A theoretical and empirical analysis. *Artificial Intelligence*, 76:427–454, 1995.

[143] I. Odrats, editor. *Information Technology in Public Administration of Estonia, yearbook 2004*. OÜ Piltkiri, 2005.

[144] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Importing the Semantic Web in UDDI. In *Proceedings of the CAiSE 2002 International Workshop on Web Services, E-Business, and the Semantic Web, WES 2002, Toronto, Canada, May 27-28, 2002, Revised Papers*, volume 2512 of *Lecture Notes in Computer Science*, pages 225–236. Springer-Verlag, 2002.

[145] M. Paolucci, J. Soudry, N. Srinivasan, and K. Sycara. A broker for OWL-S Web services. In *Proceedings of the First International Semantic Web Services Symposium, AAAI 2004 Spring Symposium Series, March 22–24, 2004*, pages 92–99. AAAI Press, 2004.

[146] M. Paolucci and K. Sycara. Autonomous Semantic Web services. *IEEE Internet Computing*, 7(5):34–41, 2003.

[147] M. Paolucci, K. Sycara, T. Nishimura, and N. Srinivasan. Using DAML-S for P2P discovery. In *Proceedings of the First International Conference on Web Services, ICWS'03, Las Vegas, Nevada, USA, June 23–26, 2003*, pages 203–207. CSREA Press, 2003.

[148] M. P. Papazoglou, B. J. Krämer, and J. Yang. Leveraging Web-services and peer-to-peer networks. In J. Eder and M. Missikoff, editors, *15th International Conference on Advanced Information Systems Engineering, CAiSE 2003, June 16–18, 2003, Klagenfurt, Austria*, volume 2681 of *Lecture Notes in Computer Science*, pages 485–501. Springer-Verlag, 2003.

[149] S. Parsons, C. Sierra, and N. Jennings. Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292, 1998.

[150] A. Patil, S. Oundhakar, A. Sheth, and K. Verma. METEOR-S Web service annotation framework. In *Proceedings of the 13th International Conference on World Wide Web (WWW '04), New York, NY, USA, May 17–22, 2004*, pages 553–562. ACM Press, 2004.

[151] C. Peltz. Web services orchestration: A review of emerging technologies, tools, and standards. Technical report, Hewlett Packard, Co., 2003.

[152] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, and P. Traverso. Planning and monitoring web service composition. In *Proceedings of the 11th International Conference on Artificial Intelligence, Methodologies, Systems, and Applications (AIMSA04), Varna, Bulgaria, September 2–4, 2004*, volume 3192 of *Lecture Notes in Computer Science*, pages 106–115. Springer-Verlag, 2004.

[153] M. Pistore, P. Traverso, P. Bertoli, and A. Marconi. Automated synthesis of composite BPEL4WS Web services. In *Proceedings of 2005 IEEE International Conference on Web Services (ICWS05), Orlando, Florida, USA, July 11–15, 2005*, pages 293–301, 2005.

[154] S. R. Ponnekanti and A. Fox. SWORD: A developer toolkit for Web service composition. In *Proceedings of The Eleventh World Wide Web Conference (Web Engineering Track), Honolulu, Hawaii, USA, May 7–11, 2002*, pages 83–107, 2002.

[155] P. Raghavan. Information retrieval algorithms: A survey. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, Louisiana, United States, January 5–7, 1997*, pages 11–18. SIAM, 1997.

[156] I. Rahwan, S. D. Ramchurn, N. R. Jennings, P. McBurney, S. Parsons, and L. Sonenberg. Argumentation-based negotiation. *The Knowledge Engineering Review*, 18(4):343–375, 2004.

[157] A. Rao. BDI agents speak out in a logical computable language. In *Proceedings of Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96)*, pages 42–55. Springer-Verlag, 1996.

[158] A. S. Rao and M. P. Georgeff. Modeling rational agents within a bdi-architecture. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91), Cambridge, Massachusetts, USA, April 22–25, 1991*, pages 473–484, 1991.

[159] J. Rao, P. Küngas, and M. Matskin. Application of linear logic to Web service composition. In *Proceedings of the First International Conference on Web Services, ICWS 2003, Las Vegas, Nevada, USA, June 23–26, 2003*, pages 3–9. CSREA Press, 2003.

[160] J. Rao, P. Küngas, and M. Matskin. Logic-based Web services composition: From service description to process model. In *Proceedings of the Second International Conference on Web Services (ICWS 2004), San Diego, California, USA, July 6–9, 2004*, pages 446–453, 2004.

[161] J. Rao, P. Küngas, and M. Matskin. Composition of semantic web services using linear logic theorem proving. *Information Systems*, 31(4–5):340–360, 2006.

[162] J. A. Robinson. Computational logic: Memories of the past and challenges for the future. In *Proceedings of the 1st International Conference on Computational Logic, CL 2000, Imperial College, London, UK, July 24–28, 2000*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pages 1–24. Springer-Verlag, 2000.

[163] M. Sabou. From software APIs to Web service ontologies: a semi-automatic extraction method. In *Proceedings of the Third International Semantic Web Conference (ISWC2004), Hiroshima, Japan, November 7–11, 2004*, 2004.

[164] M. Sabou, D. Richards, and S. van Splunter. An experience report using DAML-S. In *Proceedings of the Twelfth International World Wide Web Conference Workshop on E-Services and the Semantic Web*, 2003.

[165] F. Sadri, F. Toni, and P. Torroni. Logic agents, dialogues and negotiation: An abductive approach. In *Proceedings of the Symposium on Information Agents for E-Commerce, Artificial Intelligence and the Simulation of Behaviour Convention (AISB-2001), York, UK, March 21–24, 2001*, 2001.

[166] H. Sagan. *Space-Filling Curves*. Springer-Verlag, 1994.

[167] T. W. Sandholm and V. R. Lesser. Coalitions among computationally bounded agents. *Artificial Intelligence*, 94:99–137, 1997.

[168] C. Schmidt and M. Parashar. A peer-to-peer approach to Web service discovery. *World Wide Web Journal*, 7(2):211–229, June 2004.

[169] 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 Int. Conference on Advanced Information Systems*

Engineering (CAiSE 2000), Stockholm, Sweden, June 5–9, 2000, volume 1789 of *Lecture Notes in Computer Science*, pages 247–263. Springer-Verlag, 2000.

[170] O. Shehory and S. Kraus. Task allocation via coalition formation among autonomous agents. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95), Montréal, Québec, Canada, August 20–25, 1995*, pages 655–661, 1995.

[171] O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1–2):165–200, 1998.

[172] M. Sheshagiri, M. desJardins, and T. Finin. A planner for composing services described in DAML-S. In *Proceedings of the AAMAS Workshop on Web Services and Agent-based Engineering*, 2003.

[173] F. Silva, M. Castilho, and L. A. Künzle. Petriplan: A new algorithm for plan generation (preliminary report). In M. C. Monard and J. S. Sichman, editors, *Advances in Artificial Intelligence. Proceedings of International Joint Conference 7th Ibero-American Conference on AI 15th Brazilian Symposium on AI, IBERAMIA-SBIA 2000, Atibaia, SP, Brazil, November 19–22, 2000*, volume 1952 of *Lecture Notes in Computer Science*, pages 86–95. Springer-Verlag, 2000.

[174] E. Sirin, B. Parsia, and J. Hendler. Composition-driven filtering and selection of Semantic Web services. In *Proceedings of the First International Semantic Web Services Symposium, AAAI 2004 Spring Symposium Series, March 22–24, 2004*, pages 129–136. AAAI Press, 2004.

[175] M. Stickel, R. Waldinger, M. Lowry, T. Pressburger, and I. Underwood. Deductive composition of astronomical software from subroutine libraries. In A. Bundy, editor, *Proc. of CADE'94, Nancy, France, 1994*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 341–355. Springer-Verlag, 1994.

[176] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM 2001, San Diego, California, USA, August 27–31, 2001*, pages 149–160. ACM Press, 2001.

[177] P. D. Summers. A methodology for LISP program construction from examples. *Journal of the ACM*, 24(1):161–175, 1977.

[178] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan. Automated discovery, interaction and composition of Semantic Web services. *Journal of Web Semantics*, 1(1):27–46, September 2003.

[179] K. Sycara and D. Zeng. Coordination of multiple intelligent software agents. *International Journal of Intelligent and Cooperative Information Systems*, 5(2–3):181–211, 1996.

[180] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[181] T. Tammet. Proof strategies in linear logic. *Journal of Automated Reasoning*, 12:273–304, 1994.

[182] M. Tanabe. Timed petri nets and temporal linear logic. In *Proceedings of 18th International Conference on Application and Theory of Petri Nets (ICATPN'97), Toulouse, France, June 23–27, 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 156–174. Springer-Verlag, 1997.

[183] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proceedings of ACM SIGCOMM'03, Karlsruhe, Germany, August 25–29, 2003*, pages 175–186. ACM Press, 2003.

[184] S. Thakkar, C. A. Knoblock, J. L. Ambite, and C. Shahabi. Dynamically composing Web services from on-line sources. In *Proceeding of 2002 AAAI Workshop on Intelligent Service Integration, Edmonton, Alberta, Canada*, 2002.

[185] G. Tidhar, A. S. Rao, and E. A. Sonenberg. Guided team selection. In *Proceedings of Second International Conference on Multi-Agent Systems, ICMAS-96, Kyoto, Japan*, pages 369–376, 1996.

[186] P. Traverso and M. Pistore. Automated composition of semantic web services into executable processes. In *Proceedings of 3rd International Semantic Web Conference, ISWC 2004, Hiroshima, Japan, November 7–11, 2004*, volume 3298 of *Lecture Notes in Computer Science*, pages 380–394. Springer-Verlag, 2004.

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

[188] E. Tyugu. The structural synthesis of programs. In *Algorithms in Modern Mathematics and Computer Science*, volume 122 of *Lecture Notes in Computer Science*, pages 290–303. Springer-Verlag, 1981.

[189] T. Uustalu, U. Kopra, V. Kotkas, M. Matskin, and E. Tyugu. The NUT language report. Technical Report TRITA-IT R 94:14, Department of Teleinformatic, Royal Institute of Technology, Stockholm, Sweden, 1994.

[190] W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede. Web service composition languages: Old wine in new bottles? In *29th Euromicro Conference (EUROMICRO'03), Belek-Antalya, Turkey, September 1–6, 2003*, pages 298–307. IEEE Press, 2003.

[191] R. van der Krogt, M. de Weerdt, and C. Witteveen. A resource based framework for planning and replanning. In *Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology, IAT-03, Halifax, Canada, October 13–16, 2003*, pages 247–253. IEEE Computer Society, 2003.

[192] S. A. Vere. Relational production systems. *Artificial Intelligence*, 8:47–68, 1977.

[193] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller. METEORS WSDI: A scalable P2P infrastructure of registries for semantic publication and discovery of Web services. *Journal of Information Technology and Management*, 6(1):17–39, 2005.

[194] R. Waldinger. Web agents cooperating deductively. In *Proceedings of FAABS 2000, Greenbelt, MD, USA, April 5–7, 2000*, volume 1871 of *Lecture Notes in Computer Science*, pages 250–262. Springer-Verlag, 2001.

[195] Q. Wang, Y. Yuan, J. Zhou, and A. Zhou. Peer-Serv: A framework of Web services in peer-to-peer environment. In *Proceedings of 4th International Conference on Advances in Web-Age Information Management, WAIM 2003, Chengdu, China, August 17–19, 2003*, volume 2762 of *Lecture Notes in Computer Science*, pages 298–305. Springer-Verlag, 2003.

[196] M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative and procedural goals in intelligent agent systems. In *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002), April 22–25, 2002, Toulouse, France*, pages 470–481. Morgan Kaufmann, 2002.

[197] P. Wohed, W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede. Analysis of web services composition languages: The case of bpel4ws. In *Proceedings of 22nd International Conference on Conceptual Modeling (ER 2003), Chicago IL, USA, October 13–16, 2003*, volume 2813 of *Lecture Notes in Computer Science*, pages 200–215. Springer-Verlag, 2003.

[198] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[199] M. Wooldridge and N. R. Jennings. The cooperative problem-solving process. *Journal of Logic and Computation*, 9(4):563–592, 1999.

[200] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S Web services composition using SHOP2. In *Proceedings of the 2nd International Semantic Web Conference, ISWC 2003, Sanibel Island, Florida, USA, October 20–23, 2003*, 2003.