Czech Technical University in Prague
Faculty of Electrical Engineering

DISSERTATION THESIS

February 2007

Marek Obitko

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

Translations between Ontologies in Multi-Agent Systems

Dissertation Thesis

Marek Obitko

February 2007

Study Programme
Electrical Engineering and Information Technology

Branch of Study
Artificial Intelligence and Biocybernetics

Supervisor: **Prof. Ing. Vladimír Mařík, DrSc.**
Supervisor Specialist: **Ing. Tomáš Vlček, CSc.**

Acknowledgements

I am indebted to my advisor Professor Vladimír Mařík for all the guidance, help and support during my whole Ph.D. study. He suggested me to work in the area of ontologies and multi-agent systems and was always providing the ideas for next direction of the research. I would like to thank my advisor-specialist Tomáš Vlček as well for all the feedback provided during my Ph.D. study.

My special thanks go to Radek Mařík for many discussions on the topic of ontologies, and for inviting me to work on the Industrial Search project that helped me to use and evaluate my earlier proposals relevant to the dissertation. Also, I would like to thank Pavel Vrba for discussions about the applications of ontologies in multi-agent systems as well as for the feedback for applications in transportation domain.

There are many other people who provided help, suggestions and feedback to me in various stages of my work towards this dissertation (even when the topics did not reach the state to be included into this dissertation). My thanks go especially to (in alphabetical order) Petr Aubrecht, Michal Fořtík, Zdeněk Kouba, Lubomír Kurz, Filip Macůrek, Michal Pěchouček, Jan Šmíd, Olga Štěpánková, and Pavel Tichý.

Last, but not least, I would like to thank to my family and especially to my wife Daniela for the continuous flow of joy, for patience and support during the time I worked on this thesis.

The work on this thesis as well as publication of results was partially supported by the grants MSM212300013, MSM6840770013, and LN00B096 of the Ministry of Education, Youth and Sports of the Czech Republic and by the EU Network of Excellence I*PROMS. The work environment was supported by ProTyS, a. s. The work on the Industrial Semantic Search was supported by Rockwell Automation.

Abstract

As information systems change from isolated programs to more powerful interconnected systems, the importance of communication arises. Communicating systems can be modeled using the multi-agent systems paradigm. Each agent represents its view of the world in its knowledge base and exchanges the information about the world with other agents. Information is represented using an ontology, formal explicit specification of conceptualization. Sharing information is an issue in heterogeneous multi-agent systems where different agents may use different ontologies for the same domain.

Semantic Web is an activity towards sharing and reusing data (not only documents) on the web, and the use of multiple ontologies is inherent there. In this dissertation we investigate possibilities of using the results from the semantic web research in multi-agent systems as standardized by the Foundation for Intelligent Physical Agents.

We describe the usage of semantic web languages RDF and OWL and the query language SPARQL in multi-agent systems for information representation and for communication. We summarize the advantages, from which we point out the possibility of work with multiple ontologies and the usage of reasoning. Based on FIPA proposal we design a specialized ontology agent that uses semantic web technologies and that is able to help other agents with ontology related tasks, including translation between ontologies. We propose a way of mapping between XML and OWL, so that semantics expressed in OWL can be employed in a system where some agents use XML. In addition to using ontology agent for translation between ontologies we also propose a transparent translation made by a multi-agent platform, so that agents can be integrated even when they are not aware of ontologies. We also discuss the translation of information between different ontologies using rules, SPARQL and OWL reasoning, illustrate these approaches on ontologies from the transportation domain, and compare the features of these approaches. The translation using rules is used by our ontology agent.

Finally, we show an application of the semantic web enabled agents on an example of semantic search in industrial domain. It is shown how files of products used for assembly line design, operation and maintenance are converted to RDF/OWL, how the information is processed and stored in an agent built on the ontology agent, and how the enriched and connected information from multiple files can be searched. We discuss deduction of new information based on ontology and discuss using translation for user queries.

The contributions of this dissertation can be used for integration of heterogeneous multi-agent systems at the different levels, from agents controlling shop floor to virtual enterprises.

Abstrakt

Během změn informačních systémů z izolovaných programů ve výkonnější propojené systémy roste význam komunikace. Komunikující systémy mohou být modelovány pomocí paradigmatu multiagentních systémů. Každý agent používá svou reprezentaci světa ve své znalostní bázi a vyměňuje si informace o světě s ostatními agenty. Informace je reprezentována pomocí ontologie, definované jako formální explicitní specifikace konceptualizace. Sdílení informací znamená problém v heterogenních multiagentních systémech, kde různí agenti mohou používat různé ontologie popisující stejnou doménu.

Sémantický Web je aktivita vedoucí ke sdílení a využití dat (ne pouze dokumentů) na webu. K tomu patří použití více ontologií. V této práci studujeme možnosti použití výsledků výzkumu sémantického webu v multiagentních systémech standardizovaných nadací FIPA.

Popisujeme použití jazyků sémantického webu RDF a OWL a dotazovacího jazyka SPARQL v multiagentních systémech pro reprezentaci znalostí a pro komunikaci. Shrnujeme výhody, z nichž zdůrazňujeme možnost pracovat s více ontologiemi a použití automatického uvažování. Dle návrhu FIPA jsme vytvořili specializovaného ontologického agenta, který používá technologie sémantického webu a který je schopen pomoci ostatním agentům s úlohami týkající se ontologií, včetně překladu mezi ontologiemi. Navrhujeme způsob mapování mezi XML a OWL, takže sémantika vyjádřená v OWL může být využita i v systému, kde někteří agenti používají pouze XML. Mimo ontologického agenta též navrhujeme možnost transparentního překladu, kde je překlad prováděn multiagentní platformou, takže agenti mají možnost komunikace bez toho, že by si byli vědomi problematiky ontologií. Diskutujeme možnosti překladu informace mezi různými ontologiemi pomocí pravidel, pomocí jazyka SPARQL a pomocí uvažování v OWL, a ilustrujeme tyto přístupy na ontologiích z oblasti dopravy a porovnáváme vlastnosti těchto přístupů. Překlad pomocí pravidel je použit v našem ontologickém agentovi.

Nakonec ukazujeme aplikaci agentů pracujících s technologiemi sémantického webu na příkladu sémantického vyhledávání v doméně průmyslu. Ukazujeme, jak jsou soubory z produktů pro návrh a provoz montážní linky zkonvertovány do RDF a OWL, jak je tato informace zpracována a uložena agentem postaveným na ontologickém agentovi, a jak tato sémanticky obohacená a propojená báze může být prohledávána. Diskutujeme možnosti uvažování pro odvození informací na základě ontologie a využití překladu pro konstrukci dotazů uživatelem.

Přínosy této disertace mohou být využity pro integraci heterogenních multiagentních systémů na různých úrovních, od agentů řídících montážní linku po virtuální podniky.

Contents

I Introduction **7**

1 Introduction and Motivation **8**
1.1 Goals of the Dissertation 10
1.2 Dissertation Structure 10

II State of the Art **12**

2 Ontologies **13**
2.1 Philosophical Roots 13
2.2 What is Ontology? 15
2.2.1 Ontology as a Specification of Conceptualization 15
2.2.2 Ontology as a Body of Knowledge 17
2.2.3 Expressing Ontology 18

3 Formal Representation of Ontologies **19**
3.1 Frame-Based Models 20
3.2 Semantic Networks 22
3.3 Conceptual Graphs 22
3.4 Knowledge Interchange Format (KIF) 24
3.5 Common Logic 25
3.6 Description Logics 25
3.6.1 Syntax and Semantics 26
3.6.2 Translation to First Order Predicate Logic 27
3.6.3 Reasoning 28

4 Working with Ontologies **31**
4.1 Operations on Ontologies 31
4.2 Relationships between Ontologies 33
4.3 Modularization of Ontologies 34

5 Semantic Web **36**
5.1 Resource Description Framework (RDF) 39
5.2 RDF Schema (RDFS) 43
5.3 Web Ontology Language (OWL) 46
5.4 RDF Query Language SPARQL 50
5.5 Reasoning Support 52

6 Communication in Multi-Agent Systems **55**
6.1 FIPA Compliant Multi-Agent Systems 56
6.2 Ontologies in Multi-Agent Systems 58
6.3 FIPA Ontology Service 58
6.3.1 Ontologies 60
6.3.2 FIPA-ontol-service-ontology ontology 60
6.3.3 Implementation 61

III Translations between Ontologies in Multi-Agent Systems **62**

7 Semantic Web Technologies and Multi-Agent Systems **63**
7.1 RDF and OWL versus SL and OKBC 63
7.2 Semantic Web Technologies for Agent Communication 65
7.2.1 Agent Ontology 66
7.2.2 Implementation 68
7.3 Ontology Agent Based on OWL 68
7.3.1 OntoAgent Ontology 68
7.3.2 Usage of OntoAgent Ontology 73
7.3.3 Example Scenario 74
7.4 Adding OWL Semantics to XML 77
7.4.1 Basic Heuristics for Translation 78
7.4.2 Additional Possibilities 80

8 Translation between Ontologies **84**
8.1 Examples of Transportation Domain Ontologies 84
8.1.1 Berlin Transportation Ontology 85
8.1.2 Boeing Transportation Ontology 86
8.1.3 Material Transportation Ontology 87
8.2 Translation 88
8.2.1 Translation Using Rules 89
8.2.2 Example Message Translation 91
8.2.3 Translation Using SPARQL 92

8.2.4 Translation Using OWL Reasoning 94
8.2.5 Comparison 95
8.3 Translation in Ontology Agent 97
8.4 Architectural Considerations 98
8.4.1 Agents Handle all the Translation Themselves 99
8.4.2 Agents Knowingly Use Specialized Services 100
8.4.3 Translation is Transparent 100
8.4.4 Transparent Translation Implementation 101
8.4.5 Discussion 102

9 Application: Semantic Search in Manufacturing Industry 104
9.1 Overview and Motivation 104
9.1.1 RSLogix 5000 104
9.1.2 RSView Enterprise 106
9.1.3 Semantic Search Motivation 107
9.2 Conversion to RDF and OWL 108
9.2.1 RSLogix 5000 108
9.2.2 RSView Enterprise 109
9.2.3 File Metadata 111
9.3 Architecture 112
9.3.1 RDF Storage Agent 112
9.3.2 File Converter Agents 113
9.3.3 User Interface Agent 114
9.4 Semantic Search 114
9.4.1 Translation for User Queries 116
9.5 Performance 117

IV Discussion and Conclusion 122

10 Discussion and Conclusion 123
10.1 Summary and Discussion 123
10.1.1 Semantic Web Technologies in Multi-Agent Systems . . 124
10.1.2 Translation Between Ontologies 125
10.1.3 Application: Semantic Search in Manufacturing Industry 126
10.2 Future Work 127
10.3 Dissertation Contributions and Goals Evaluation 128
10.4 Conclusion 129

List of Figures

2.1 Ontology expressing intended models for description of the domain. 16

3.1 Levels of expressivity in ontology description 19
3.2 Simple conceptual graph in the graphical representation DF . 23

4.1 Modularization of ontologies depending on the scope and partial ordering defined by inheritance. 34

5.1 Semantic web architecture in layers 37
5.2 RDF triple (in graph representation) describing Joe Smith — "Joe has homepage identified by URI" 39
5.3 RDF triple (in graph representation) describing Joe Smith — "Joe has family name *Smith*" 40
5.4 RDF graph describing Joe Smith 41
5.5 Illustration of a bag (unordered container) and a blank node. . 42
5.6 Pizza OWL ontology expressed in RDF triples 54

6.1 FIPA Agent Management Reference Model [46] 56

7.1 Architecture of the FIPA ontology agent implemented in Jade and using Jena API to access OWL ontologies. 69
7.2 A snapshot of the JADE sniffer agent: message flow between agents in the sample FIPA scenario. 75
7.3 Data flow diagram of translating between XML and OWL. . . 78
7.4 XML2OWL/RDF Wizard asking for first properties of the conversion. 81

8.1 Berlin transportation ontology (upper part with main concepts). 85
8.2 Boeing transportation ontology (upper part with main concepts). 86

8.3 Material transportation ontology (upper part with main concepts). 87
8.4 Illustration of translations from source to target ontologies in various situations. 90
8.5 Translation time depending on the number of triples for SPARQL `CONSTRUCT` query. 96

9.1 RSLogix 5000 screenshot — editing ladder code. 105
9.2 RSView Enterprise ME screenshot — editing display. 106
9.3 XML2OWL/RDF Wizard asking for object properties names and whether to use domain and range for them during the conversion of RSView display files. 110
9.4 Semantic search multi-agent system architecture. 112
9.5 Time for XML to RDF conversion, and comparison of offline reasoning methods. 119
9.6 Comparison of online reasoning time. 120

List of Tables

3.1 $\mathcal{AL}$ (attributive language) logic syntax and semantics 26
3.2 Examples of $\mathcal{AL}$ logic basic extensions 27
3.3 Examples of description logics complexity [19, 96] 29

5.1 RDFS classes 44
5.2 RDFS properties (all are instances of `rdf:Property`) 45
5.3 RDFS instance 46
5.4 OWL DL descriptions, data ranges, properties, individuals and data values syntax and semantics [56, 80, 19] 47
5.5 OWL DL axioms and facts [56, 80, 19] 48

6.1 Main elements of the `FIPA-ontol-service-ontology`. 61

7.1 Comparison of SL, OKBC and FIPA specification versus RDF and OWL for the usage in multi-agent systems [73]. 65

8.7 Comparison of translation time depending on the number of triples. 96
8.8 Comparison of the performance of translation architectures. . 103

9.4 Comparison of offline reasoning time — number of triples, conversion to RDF, and reasoning using naive program, Jena reasoner, Pellet and Jena rules. 119
9.5 Comparison of online reasoning time — number of millions of triples, reasoning using Jena reasoner and Pellet reasoner . . . 120

Part I

Introduction

Chapter 1

Introduction and Motivation

Computer systems are not isolated islands any more, and in order to accomplish their work, they need to communicate with other systems. In artificial intelligence, the paradigm of multi-agent and holonic systems has deservedly gained big popularity, as it allows efficient and robust decomposition of a task. In software engineering, the service oriented architecture is directing the development of software systems in the same way.

Since the systems need to communicate (i.e., exchange data, information and knowledge) they need to *understand* each other. This understanding means that when a message is communicated from one system to another one, the message needs

- to be delivered to the other system — i.e., the physical layer of the communication needs to physically transport the message between two systems
- to be decoded by the other system — i.e., at the syntactical layer, both systems have to use the same syntax (encoding) of the message
- to be understood by the other system — i.e., at the semantical layer, systems have to understand each other; they have to use the same words and constructs to express the same meaning.

The compatibility at all of these three layers may mean no problem for systems that were designed to work together from the beginning of their development. However, in real world cases, systems need to be connected after they are developed, and need to work with systems that no one expected to be connected. In these case, the heterogeneity at all these three layers arises — systems from different vendors (which may mean just a different

department within the same company) use different means to represent data, information and knowledge.

Interesting real-life example provided in [60] shows how a transportation chain involving different companies works well for transportation of containers weighing tons around the world, but doesn't work so well for transportation of information saying what is in these containers. This results in the need of manual repacking of huge number of containers when they arrive to see what is in them. The problem is that there are hundreds of different computer systems and databases in airlines, shipping companies, trucking companies and manufacturing companies that were never designed to directly operate with each other. Although each system may be efficient, the interfaces between these systems introduce tremendous disruption and delay and usually require significant human intervention.

From the layers described above, we are primarily interested in the third layer, semantics. The semantics can be captured using so called *ontologies*. Ontologies, defined as "formal explicit specification of conceptualization", capture the structure of the domain, i.e., they define how to model the state of affairs in a domain together with possible restrictions. Ontologies capture knowledge that is not changing (or that is changing very rarely), while a particular knowledge base or a particular message exchanged between agents captures the particular state of affairs. The problem of semantical heterogeneity can be then formulated as the state when systems that use different ontologies describing the same domain need to communicate.

It is unlikely that there would be ever one ontology for even a simple domain that would satisfy everybody and forever [79]. Standardization is a good way to ensure communication and understanding, but even standards are evolving and like ontologies they are also intended for some specific applications.

Multi-agent systems and individual agents need to be connected for a common work in a shared domain even when they do not use the same ontologies. What makes the task even harder is that some of the systems do not have their "ontologies" expressed explicitly.

The topic of ontologies is important for the Semantic Web area. In fact, the usage of multiple ontologies is one of the central ideas of Semantic Web. Semantic Web is an effort to enhance current web so that computers can process information presented on WWW instead of only delivering documents to human. The focus of semantic web is to share data instead of documents. In the Semantic Web area of research ontologies play important role.

We investigate if and how the Semantic Web ideas can be used for the multi-agents systems to help them overcome the problems that we have de-

scribed above. For this, we need to investigate how the semantic web technologies can be used in multi-agent systems. The translation between ontologies is still not very well developed topic in the semantic web area, but such a translation is critical for achieving the goal of communication in heterogeneous multi-agent systems.

Our approach could be well tested on a proposal of a specialized ontology agent that would help other agents with ontological tasks, and that was not fully implemented yet.

Our application and testing area is primarily the field of manufacturing [62]. The difference from the web search applications is that we are interested in exact results during communication and search instead of possibly approximate results as it is usually allowed in web search applications. We evaluate our approach on two main areas — on transportation domain and on search within files of software products used for assembly line design, operation, and maintenance.

1.1 Goals of the Dissertation

To summarize, our goals for this dissertation thesis are as follows.

- Investigate and propose possibilities of using the results of research from the semantic web area in the area of multi-agent systems in order to help multi-agent systems to overcome the problems with ontologies that they face.
- Investigate, propose and evaluate ways of translation between different ontologies, especially for agent communication, using semantic web languages, when the translation information would be expressed in a declarative form.
- Investigate the possibility of implementation of the ontology agent that has been proposed by FIPA using semantic web technologies.
- Show and evaluate a manufacturing domain application of the semantic web technologies enabled agents.

1.2 Dissertation Structure

This dissertation thesis is organized as follows.

After the introduction and motivation we summarize the state of the art in the areas of ontologies, semantic web, and multi-agent systems. In the

part that is discussing ontologies we make a brief overview of the definitions of ontology and formalisms that are currently used for expressing ontologies. The emphasis is put on description logics that have good computational properties and are used in the semantic web. In the semantic web chapter we discuss primarily the Web Ontology Language OWL that will be used in the rest of the thesis. In multi-agent systems overview we are primarily interested in communication matters and in a proposal for a specialized agent that would help other agents with ontology related tasks.

The next part is formed by the description of our work. First we describe the possibilities of usage of semantic web technologies in multi-agent systems, in a FIPA platform that is popular in the manufacturing systems area. We compare the FIPA standards with Web Ontology Language and propose agent communication based on semantic web technologies. Also, we discuss the specialized ontology agent proposal and describe its implementation. We illustrate its functionality on a sample scenario. We also discuss the interoperability of agents that use primarily XML as a syntax for communication language. A mapping wizard is proposed that allows adding OWL semantics to XML so that the XML messages can be used to employ OWL semantics.

In next chapter we show our proposals for translation between ontologies. We illustrate our approach on transportation ontologies and show how the translation can be handled in the ontology agent. We also discuss architectural issues and propose a translation that would be transparent to agents, so that they would not have to be aware of translation during communication with agents that use different ontologies.

In the last chapter describing our work we show an application of the semantic web enabled agents for a semantic search in manufacturing industry. We show how the files of software packages for assembly line design, operation and maintenance can be enriched by semantic information stored by an agent that is extension of the ontology agent and how this information can be used for semantic search.

In the last part we discuss our work, provide directions for future research, summarize the contributions of the thesis, and conclude.

Part II

State of the Art

Chapter 2

Ontologies

The history of artificial intelligence shows that knowledge is critical for intelligent systems. In many cases, better knowledge can be more important for solving a task than better algorithms. To have truly intelligent systems, knowledge needs to be captured, processed, reused, and communicated. Ontologies support all these tasks.

The term "ontology" can be defined as an explicit specification of conceptualization [50]. Ontologies capture the structure of the domain, i.e. conceptualization. This includes the model of the domain with possible restrictions. The conceptualization describes knowledge about the domain, not about the particular state of affairs in the domain. In other words, the conceptualization is not changing, or is changing very rarely. Ontology is then specification of this conceptualization — the conceptualization is specified by using particular modeling language and particular terms. Formal specification is required in order to be able to process ontologies and operate on ontologies automatically.

Ontology describes a domain, while a knowledge base (based on an ontology) describes particular state of affairs. Each knowledge based system or agent has its own knowledge base, and only what can be expressed using an ontology can be stored and used in the knowledge base. When an agent wants to communicate to another agent, he uses the constructs from some ontology. In order to understand in communication, ontologies must be shared between agents.

2.1 Philosophical Roots

The term ontology was taken from philosophy. According to Webster's Dictionary an ontology is

- *a branch of metaphysics relating to the nature and relations of being*
- *a particular theory about the nature of being or the kinds of existence*

Ontology (the "science of being") is a word, like metaphysics, that is used in many different senses. It is sometimes considered to be identical to metaphysics, but we prefer to use it in a more specific sense, as that part of metaphysics that specifies the most fundamental categories of existence, the elementary substances or structures out of which the world is made. Ontology will thus analyze the most general and abstract concepts or distinctions that underlay every more specific description of any phenomenon in the world, e.g. time, space, matter, process, cause and effect, system.

Recently, the term of "ontology" has been up taken by researchers in Artificial Intelligence, who use it to designate the building blocks out of which models of the world are made. An agent (e.g. an autonomous robot) using a particular model will only be able to perceive that part of the world that his ontology is able to represent. In this sense, only the things in his ontology can exist for that agent. In that way, an ontology becomes the basic level of a knowledge representation scheme. An example is set of link types for a semantic network representation which is based on a set of "ontological" distinctions: changing–invariant, and general–specific.

Ontology is originally [53] a branch of philosophy that deals with the nature and the organization of reality. It tries to answer questions like "what is existence", "what properties can explain the existence" etc. According to [53], Aristotle defined ontology as the science of being as such. Unlike the special sciences, each of which investigates a class of beings and their determinations, ontology regards "all the species qua being and the attributes that belong to it qua being". In this sense the philosophical ontology tries to answer the question "what is the being?" or, in a meaningful reformulation "what are the features common to all beings?".

This is what can be called "general ontology" — in contrast with various special ontologies for a particular domain. Such an ontology is not concerned with the existence of certain objects, but rather in the rigorous description of their forms of being, i.e., their structural features. In practice, ontology can be intended as the theory of the distinctions, which can be applied independently of the state of the world. In particular, we are interested in distinctions:

- among the entities of the world (physical objects, events, regions...)
- among the meta-level categories used to model the world (concept, property, quality, state, role, part...)

The research of ontology in this philosophical sense is relevant for knowledge based systems. Ontology forms a base for knowledge used in such systems.

2.2 What is Ontology?

Although it is required from an ontology to be formally defined, there is no common definition of the term "ontology" itself. The definitions can be categorized into roughly three groups (more detailed discussion is available for example in [53, 67, 65]):

1. Ontology is a term in philosophy and its meaning is "theory of existence".
2. Ontology is an explicit specification of conceptualization [50].
3. Ontology is a body of knowledge describing some domain, typically common sense knowledge domain

The definition 1 is the meaning in philosophy as we have discussed above, however it has many implications for the AI purposes. The second definition is generally accepted as a definition of what an ontology is for the AI community. The last third definition views an ontology as an inner body of knowledge, not as the way to describe the knowledge.

2.2.1 Ontology as a Specification of Conceptualization

The second definition of ontology mentioned above, "explicit specification of conceptualization", comes from Thomas Gruber [51, 49]. The exact meaning depends on the understanding of the terms "specification" and "conceptualization". According to [51], explicit specification of conceptualization means that an ontology is a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents. This definition is consistent with the usage of ontology as set of concept definitions, but more general.

A conceptualization, as explained in [53], can be defined as an intensional semantic structure that encodes implicit knowledge constraining the structure of a piece of a domain. Ontology is a (partial) specification of this structure, i.e., it is usually a logical theory that expresses the conceptualization explicitly in some language. Conceptualization is language independent, while ontology is language dependent. The use can be illustrated in the figure 2.1 — it shows how an ontology restricts (i.e., defines) possible use of

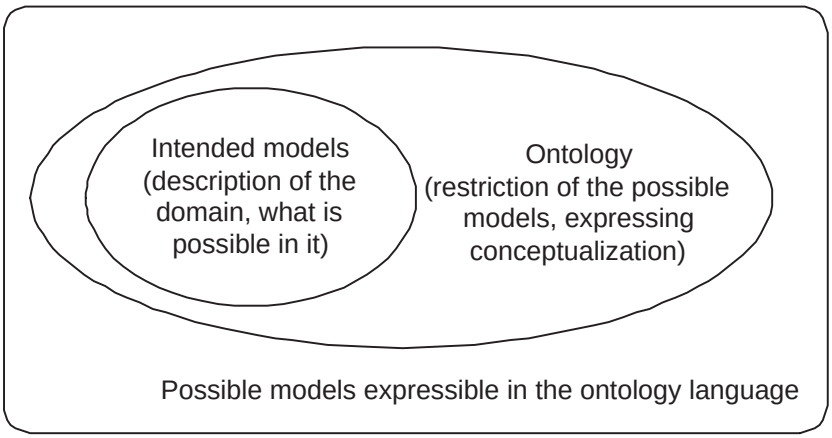

Figure 2.1: Ontology expressing intended models for description of the domain.

constructs used in the description of the domain [52]. Notice that ontology does not have to express all the possible constraints — the level of details in conceptualization depends on the requirements of the intended application and expressing conceptualization in ontology in addition depends on the used ontology language.

In this sense, ontology is important for the purpose of enabling knowledge sharing and reuse. An ontology is in this context a specification used for making ontological commitments. Practically, an ontological commitment is an agreement to use a vocabulary (i.e., ask queries and make assertions) in a way that is consistent (but not complete) with respect to the theory specified by an ontology. Agents then commit to ontologies and ontologies are designed so that the knowledge can be shared among these agents.

The representation of a body of knowledge (knowledge base) is based on the specification of conceptualization. A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. Every knowledge base, knowledge-based system or agent is committed to some conceptualization, explicitly or implicitly. For these systems, what "exists" is what can be represented. When the knowledge of a domain is represented in a declarative formalism, the set of objects that can be represented is called the universe of discourse. This set of objects and the describable relationships among them are reflected in the representational vocabulary with which a knowledge-based program represents knowledge. Thus, in the context of AI, we can describe the ontology of a program by defining a set of representational terms. In such an ontology, definitions associate the names

of entities in the universe of discourse (e.g. classes, relations, functions, or other objects) with descriptions of what the names mean, and formal axioms that constrain the interpretation and well-formed use of these terms. Formally it can be said that an ontology is a statement of a logical theory [50, 51].

The backbone of ontology is often a taxonomy. Taxonomy is a classification of things in a hierarchical form. It is usually a tree or a lattice that express subsumption relation — i.e., A subsumes B meaning that everything that is in A is also in B. An example is classification of living organisms. The taxonomy usually restricts the intended usage of classes — where classes are subsets of the set of all possible individuals in the domain. A taxonomy of properties can be defined as well.

However, ontologies need not to be limited to taxonomic hierarchies of classes and need not to be limited to definitions that only introduce terminology and do not add any knowledge about the world. To specify a conceptualization, axioms that constrain the possible interpretations for the defined terms may be also needed. Pragmatically, an ontology defines the vocabulary with which queries and assertions are exchanged among agents. The ontological commitment is then a guarantee of consistency for communications.

2.2.2 Ontology as a Body of Knowledge

Sometimes, ontology is defined as a body of knowledge describing some domain, typically a common sense knowledge domain, using a representation vocabulary as described above. In this case, an ontology is not only the vocabulary, but the whole "upper" knowledge base (including the vocabulary that is used to describe this knowledge base).

The typical example is the project CYC [36, 37, 93] that defines its knowledge base as an ontology for any other knowledge based system. CYC is the name of a very large, multi-contextual knowledge base and inference engine. CYC is an early attempt to do symbolic AI on a massive scale by capturing common knowledge that is required to do tasks that are trivial for people, but very hard for computers. All of the knowledge in CYC is represented declaratively in the form of logical assertions. CYC contains over $400,000$ significant assertions [93], which include simple statements of facts, rules about what conclusions to draw if certain statements of facts are satisfied, and rules about how to reason with certain types of facts and rules. New conclusions are derived by the inference engine using deductive reasoning. The CYC common sense knowledge can be used as a foundation of a knowledge base for any knowledge intensive system. In this sense, this body of knowledge

can be viewed as an ontology of the knowledge base of the system.

Generally speaking, it is not possible to objectively distinguish what should be in ontology and what should be in knowledge base. It depends on the requirements of intended usage of the ontology. That is why the CYC view of ontology is not incompatible with the Gruber's definition, even when it may contain facts that one would classify as representing state of affairs.

2.2.3 Expressing Ontology

In this thesis, we will use the meaning of ontology as an "explicit specification of conceptualization". The ontology in this sense constrains the intended usage of the terms in ontology (as illustrated in the figure 2.1) so that the ontology forms a vocabulary and axioms that can be used to express knowledge base and that can be used for sharing knowledge between different systems.

In addition, we require that the specification is formal (like a program is formally written in a programming language), so that the ontology can be processed by a computer. Some of the formal languages that can be used for expressing ontologies are described in the next chapter.

Unless said otherwise, in the rest of the thesis the ontology is used in the sense of "formal explicit specification of conceptualization".

Chapter 3

Formal Representation of Ontologies

Ontologies, if they are to be used for automatic processing in computers, need to be specified formally. There are several languages that are used for expressing ontologies. In this chapter we summarize some of them. These languages provide means for expressing particular ontology, and often they can be also used for expressing knowledge base based on the ontology.

The formality of the description of ontologies is summarized in the figure 3.1. On the right end of the scale there is a catalog of terms used for expressing knowledge or information. These terms may have no description at all, and they are understood only because they are chosen from the natural language, and their meaning can be only estimated. The description of each term in natural language is better, especially if there are also relations expressed between the terms, such as is-a, part-of, related-to, etc. However, until this description is in natural language, which is not formally defined, we usually do not call such specification ontology.

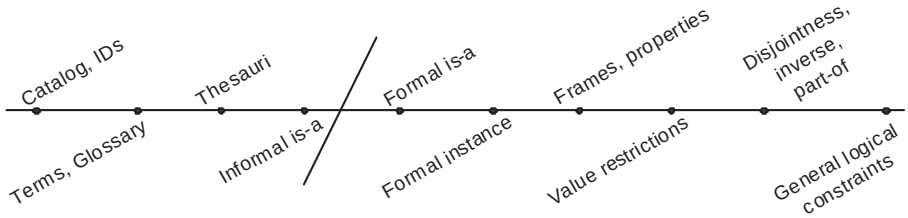

Figure 3.1: Levels of expressivity in ontology description

The point, from which we start to talk about ontology, about formal spec-

ification of conceptualization, is when at least one formal relation is defined and used between terms. Typically this is formal is-a relation, taxonomies of classes and properties, class-instance relationship, etc. Formal relation means that it is expressed in a formal language and that it is possible to conclude implications that follow from the fact that two terms are connected with the relation. One level higher there are languages that provide set of constructs to describe ontology, such as frames or simplified logics such as description logics. These languages allow to describe classes, instances, and relations between them, and to constrain their intended usage. On the right end of the scale there is a general logical theory, where we can use full logics (such as first or higher order logics, modal logics) to express intended usage of terms that are used in ontology.

The scale in the figure 3.1 also expresses the usability of ontologies. The part of informal description is not very usable for automatic processing, at least not when natural language understanding is not available. Of course, the less formal description is used the easier it is to develop an ontology. On the other hand, if we have more formal and expressive description, then such description can be automatically processed, we can better capture the intended meaning, and sharing of ontology is easier. When working with an ontology, we usually want to maintain consistency. However, it is not always possible to maintain completeness and decidability. For example, when using description in first order predicate logic, we cannot guarantee decidability, which can be guaranteed in less expressive description logics. Thus in practical cases one needs to find balance between expressivity and computational properties of the description.

A good general recommendation [86] for choosing ontology modeling language is to choose a formalism that will not limit us during the development of ontology and that will allow us to express all the distinctions that we need. When this description is finished, and the computational properties are not acceptable, then we can use only a subset of the description for some levels of automatic processing. We will see this approach later in this thesis during the description of layered languages in the semantic web (chapter 5).

In the rest of this chapter we will briefly describe selected formalisms for ontology modeling and for expressing information and knowledge.

3.1 Frame-Based Models

Frame based systems use entities like frames and their properties as a modeling primitive. The central modeling primitive is a frame together with slots. These slots are applicable only to the frames they are defined for.

Value restriction (facets) can be defined for each attribute. A frame provides a context for modeling one aspect of a domain. An important part of frame-based languages is the possibility of inheritance between frames. The inheritance allows inheriting attributes together with restrictions on them. Knowledge base then consists from instances (objects) of these frames.

An example of the usage of the frame-based model is Open Knowledge Base Connectivity (OKBC) [33, 42] that defines API for accessing knowledge representation systems. It defines most of the concepts found in frame-based systems, object databases and relational databases. The OKBC API is defined in language independent fashion, and implementations exist for Common Lisp, Java, and C [33]. The OKBC API provides operations for manipulating knowledge expressed in an implicit representation formalism called the OKBC Knowledge Model. The conceptualization in OKBC Knowledge Model is based on frames, slots, facets, instances, types, and constants. This knowledge model supports an object-oriented representation of knowledge and provides a set of representational constructs and thus can serve as an *interlingua* for knowledge sharing and translation. The OKBC Knowledge Model includes constants, frames, slots, facets, classes, individuals, and knowledge bases. For precise description of the model, the KIF language ([48], also see section 3.4) is used.

The OKBC knowledge model assumes a universe of discourse consisting of all entities about which knowledge is to be expressed. In every domain of discourse it is assumed that all *constants* of the following basic types are always defined: integers, floating point numbers, strings, symbols, lists, classes. It is also assumed that the logical constants true and false are included in every domain of discourse. Classes are sets of entities, and all sets of entities are considered to be classes.

A *frame* is a primitive object that represents an entity in the domain of discourse. A frame is called *class frame* when it represents a class, and is called *individual frame* when it represents an individual. A frame has associated with it a set of *slots* that have associated a set of *slot values*. A slot has associated a set of facets that put some restrictions on slot values. Slots and slot values can be again any entities in the domain of discourse, including frames. A *class* is a set of entities, that are *instances* of that class (one entity can be instance of multiple classes). A class is a *type* for those entities. Entities that are not classes are referred to as *individuals*. Class frames may have associated a *template slots* and *template facets* that are considered to be used in instances of subclasses of that class. Default values can be also defined. Each slot or facet may contain multiple values. There are three collection types: *set*, *bag* (unordered, multiple occurrences permitted), and *list* (ordered bag). A *knowledge base* (KB) is a collection

of classes, individuals, frames, slots, slot values, facets, facet values, frame-slot associations, and frame-slot-facet associations. KBs are considered to be entities of the universe of discourse and are represented by frames. There are defined standard classes, facets, and slots with specified names and semantics expressing frequently used entities [33, 42].

3.2 Semantic Networks

Semantic network (also called concept network) is a graph, where vertices represent concepts and where edges represent relations between concepts. Semantic network at the level of ontology expresses vocabulary that is helpful especially for human, but that still can be usable for machine processing. The relations between concepts that are used in semantic networks are as follows:

- synonym — concept A expresses the same thing as concept B
- antonym — concept A expresses the opposite of concept B
- meronym, holonym — part-of and has-part relation between concepts
- hyponym, hypernym — inclusion of semantic range between concepts in both directions

Semantic networks were created as an attempt to express interlingua, a common language that would be used for translation between various natural languages. A typical example is WordNet [13] that describes relations between English words and defines the words using natural language. Parts of WordNet were translated to other languages and the links between various languages exist and can be used as the base for translation.

Topic Maps [81] are (syntactically) standardized form of semantic networks. They allow using topics (concepts), associations (relations) between concepts (including specifying role of topic in the association), and occurrences (resources relevant to topic, in fact instances of topic). Topics, associations and occurrences are used to create ontology of a domain, and a particular topic map then uses them to expresses state of affairs in the domain.

3.3 Conceptual Graphs

Conceptual Graphs (CG) [86] is a logical formalism that includes classes, relations, individuals and quantifiers. This formalism is based on semantic

networks, but it has direct translation to the language of first order predicate logic, from which it takes its semantics. The main feature is standardized graphical representation that like in the case of semantic networks allows human to get quick overview of what the graph means. Conceptual graph is a bipartite orientated graph where instances of concepts are displayed as rectangle and conceptual relations are displayed as ellipse. Oriented edges then link these vertices and denote the existence and orientation of relation. A relation can have more than one edges, in which case edges are numbered. An example of a graphical representation, so called Display Form (DF), of a sentence "a cat is on a mat" is shown in the figure 3.2.

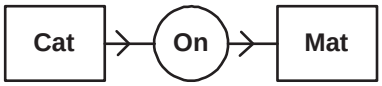

Figure 3.2: Simple conceptual graph in the graphical representation DF

Using textual notation Linear Form (LF) this sentence would be written as

$$[Cat] - (On) - [Mat]$$

DF and LF are intended as representation (and presentation) formats for human. There is also a formal language CG Interchange Form (CGIF) defined. In this language the sentence would be expressed as

$$[Cat : *x]\,[Mat : *y]\,(On\ ?x\ ?y)$$

where $*x$ is a variable definition and $?x$ is a reference to the defined variable. Using syntactical shortcuts, the same sentence could be also written in the same language as

$$(On\,[Cat]\,[Mat])$$

The conversion between the three languages is defined as well as direct conversion between CGIF and KIF (Knowledge Interchange Format, see next section). In the KIF language this example would be expressed as

$$(exists\,((?x\,Cat)\,(?y\,Mat))\,(On\,?x\,?y))$$

All these forms have the same semantics in the predicate logic:

$$\exists x, y : Cat(x) \wedge Mat(x) \wedge on(x, y)$$

According to [86] conceptual graphs have the same expressing power as predicate logic. As we have seen there is again a possibility to define concepts (i.e., ontology) and then use them to express particular state of affairs.

3.4 Knowledge Interchange Format (KIF)

Knowledge Interchange Format (KIF) [48] is a language designed to be used for exchange of knowledge between different systems. It is based semantically on predicate logic and syntactically on LISP. It allows representing arbitrary sentences in the first order predicate logic. This language was defined within the Ontolingua [50, 51] project that provides a cooperative ontology builder that allows exporting ontologies to various formalisms.

When KIF is used, one usually implements a representation formalism in KIF and uses this implementation for representation of particular ontology or knowledge. This is also the case of Ontolingua - Frame Ontology defining classes, slots, facets etc. was defined in KIF, and the KIF together with the frame ontology forms the language of Ontolingua, that allows write ontologies in a canonical form. These ontologies can then be exported to other formalisms, such as Prolog [50]. Even when KIF was primarily intended as interlingua, it is currently used for encoding knowledge directly. Other formats may be exported from KIF definition automatically (see for example [8]).

For example, KIF definition expressing that a rail vehicle is a vehicle designed to move on railways is written as:

```
(subclass RailVehicle LandVehicle)
(documentation RailVehicle
         "A Vehicle designed to move on &%Railways.")
(=>  (instance ?X RailVehicle)
     (hasPurpose ?X
      (exists (?EV ?SURF)
        (and  (instance ?RAIL Railway)
              (instance ?EV Transportation)
              (holdsDuring (WhenFn ?EV)
                      (meetsSpatially ?X ?RAIL))))))
```

Since KIF is highly expressive to be able to serve as an interchange format between various knowledge representation formalisms, it is not surprising that no reasoning support has ever been provided within Ontolingua.

3.5 Common Logic

Common Logic (CL) [1, 15] is a framework for a family of logic-based languages with the purpose of standardizing syntax and semantics for information interchange. It is a work in progress towards an ISO standard (currently ISO Draft [15]). There are three syntaxes standardized:

- CLIF — Common Logic Interchange Format, based on KIF
- CGIF — Conceptual Graph Interchange Format
- XCL — eXtended Common Logic Markup Language, based on XML

Each CL feature has a concrete expression in these three syntaxes; in other words, any statements in any of these languages can be translated to any other language while preserving the original semantics. Common Logic model theory supports first order predicate logic, and so can be used for exchanging first order formulas in a standardized form. It also supports some features of higher order logics, such as quantifying over predicates specified in the domain of discourse (not over any possible predicates).

This project is similar to the Ontolingua project [51] in a sense that it aims to propose interchange format, however it is more focused on logic-based languages and builds its semantics on the first order predicate logic. It also proposes multiple syntactic forms to be used for expressing the same knowledge. Also, it should become official ISO standard.

3.6 Description Logics

Description logics (DL) [19, 47] are logics serving primarily for formal description of concepts and roles (relations). These logics were created from the attempts to formalize semantic networks and frame based systems. Semantically they are found on predicate logic, but their language is formed so that it would be enough for practical modeling purposes and also so that the logic would have good computational properties such as decidability. The focus of research in DLs is how the various DL constructs are usable for real world applications and what is the impact of them against the complexity of reasoning.

Knowledge representation system based on DLs consists of two components — TBox and ABox. The TBox describes *terminology*, i.e., the ontology in the form of concepts and roles definitions, while the ABox contains *assertions* about individuals using the terms from the ontology. Concepts describe sets of individuals, roles describe relations between individuals.

3.6.1 Syntax and Semantics

The description of ontologies and knowledge in description logics uses constructs that have semantics given in predicate logic. However, due to historical reasons, different notation is used, that is closer to semantic networks and frame based systems. Let us have a look at the $\mathcal{AL}$ (attribute language) logic that is a minimal logic with a practically usable vocabulary. In the table 3.1 there is the syntax and semantics of the $\mathcal{AL}$ logic including a short comment. In the table as well as in the following description A and B are atomic concepts, C and D are concept descriptions, and R is atomic role. The semantics is defined using interpretation $\mathcal{I}$ that consists of non-empty set $\Delta^{\mathcal{I}}$ (the domain of interpretation) and an interpretation function, which assigns a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ to every atomic concept A and that assigns a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to every atomic role R. The interpretation function is then extended by inductive definitions summarized in the table 3.1. Two concepts C and D are equivalent, written $C \equiv D$, if $C^{\mathcal{I}} = D^{\mathcal{I}}$ for all interpretations $\mathcal{I}$.

Syntax	Semantics	Comment
A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$	atomic concept
R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	atomic role
$\top$	$\Delta^{\mathcal{I}}$	top (most general) concept
$\bot$	$\emptyset$	bottom (most specific) concept
$\neg A$	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$	atomic negation
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	intersection
$\forall R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in R^{\mathcal{I}} \Rightarrow b \in C^{\mathcal{I}}\}$	value restriction
$\exists R.\top$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in R^{\mathcal{I}}\}$	limited existential quantification

Table 3.1: $\mathcal{AL}$ (attributive language) logic syntax and semantics

Let us illustrate the syntax and expressivity on a simple example. Let us suppose that Person and Female are atomic concepts. Then Person$\sqcap$Female is an $\mathcal{AL}$ concept describing persons that are female, i.e., women. In a similar way, Person $\sqcap \neg$Female would describe man. In addition, let us suppose that hasChild is an atomic role. Then Person $\sqcap \exists$hasChild.$\top$ describes persons that have a child, and Person $\sqcap \forall$hasChild.Female describes persons all of whose children are female.

The $\mathcal{AL}$ logic can be further extended by adding new constructs, see table 3.2 for examples. The name of the logic is then formed from the string

Name	Syntax	Semantics	Comment
$\mathcal{U}$	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	union of two concepts
$\mathcal{E}$	$\exists R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$	full quantification
$\mathcal{N}$	$\geqslant nR$ $\leqslant nR$	$\{a \in \Delta^{\mathcal{I}} \mid \ \mid\{b \mid (a,b) \in R^{\mathcal{I}}\}\mid \geq n\}$ $\{a \in \Delta^{\mathcal{I}} \mid \ \mid\{b \mid (a,b) \in R^{\mathcal{I}}\}\mid \leq n\}$	number restriction
$\mathcal{C}$	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	negation of arbitrary concept

Table 3.2: Examples of $\mathcal{AL}$ logic basic extensions

$\mathcal{AL}[\mathcal{U}][\mathcal{E}][\mathcal{N}][\mathcal{C}]$, so for example the logic $\mathcal{ALEN}$ is the attributive language logic extended with full existential quantification and number restrictions. Some of the combinations are not unique from the semantic point of view — for example, union and existential quantification can be expressed using negation. Letter $\mathcal{C}$ is preferred to combination $\mathcal{UE}$ in logic name.

Some further extensions of $\mathcal{ALC}$ logic that will be of interest for us are as follows.

- $\mathcal{S}$ — role transitivity $Trans(R)$ (asserting that role is transitive)
- $\mathcal{H}$ — role hierarchy $R \subseteq S$ (asserting hierarchy of roles)
- $\mathcal{I}$ — role inverse R^{-} (creating inverse role)
- $\mathcal{F}$ — functionality $\leqslant 1R$ (functional role in concept creation)
- $\mathcal{O}$ — nominals $\{a_1, ..., a_n\}$ (concept declared by enumeration)

These constructors or axioms all extend the $\mathcal{ALC}$ logic and so it is enough to specify the extensions in the logic name — for example, logics $\mathcal{SHIF}$ and $\mathcal{SHOIN}$ will be interesting for us later in this thesis.

3.6.2 Translation to First Order Predicate Logic

As we have seen, the notation is different from the predicate logic one, but semantically, (some of) the description logics are subsets of first order predicate logic. A concept corresponds to an unary predicate while a role corresponds to a binary predicate. More formally, a concept C corresponds to a predicate logic formula $\phi_C(x)$ with one free variable x such that for every interpretation $\mathcal{I}$ the set of elements of $\Delta^{\mathcal{I}}$ satisfying $\phi_C(x)$ is exactly $C^{\mathcal{I}}$. An atomic

concept A is translated into the formula $A(x)$, the constructors intersection, union, and negation are translated into logical conjunction, disjunction, and negation. If C is translated to $\phi_C(x)$ and R is an atomic role, then $\exists R.C$ and $\forall R.C$ are translated as

$$\phi_{\exists R.C}(y) = \exists x.R(y,x) \vee \phi_C(x)$$

$$\phi_{\forall R.C}(y) = \forall x.R(y,x) \rightarrow \phi_C(x)$$

where y is a new variable. For further details of relationship between first order predicate logic and description logics see [19].

3.6.3 Reasoning

Reasoning in ontologies and knowledge bases is one of the reasons why a specification needs to be formal one. By reasoning we mean deriving facts that are not expressed in ontology or in knowledge base explicitly. All of the formalisms that were discussed in this chapter were created with the outlook of automatic processing, but due their properties such as decidability or computational complexity or even due to the level of formality it is not always possible. In this section we will discuss reasoning for description logics only.

Description logics are created with the focus on tractable reasoning. A few examples of tasks required from reasoner are as follows.

- Satisfiability of a concept — determine whether a description of the concept is not contradictory, i.e., whether an individual can exist that would be instance of the concept.
- Subsumption of concepts — determine whether concept C subsumes concept D, i.e., whether description of C is more general than the description of D.
- Consistency of ABox with respect to TBox — determine whether individuals in ABox do not violate descriptions and axioms described by TBox.
- Check an individual — check whether the individual is an instance of a concept
- Retrieval of individuals — find all individuals that are instances of a concept

- Realization of an individual — find all concepts which the individual belongs to, especially the most specific ones

These tasks are not semantically very different. For example, satisfiability can be tested as subsumption of $\bot$ — concept is unsatisfiable if no individual can exist that would be instance of the concept. For all tasks, it is enough to be able to check deductive consequence or derive all deductive consequences of a theory. However, there may be special optimized algorithms for different tasks in a reasoner.

The complexity of selected DLs is shown in the table 3.3 — all of the logics in the table are decidable. Even when the theoretical complexity seems to be intractable, there are optimized reasoners available that are usable for practical real world cases.

Description logic	Subsumption computation complexity
$\mathcal{AL}$	PTime
$\mathcal{ALC}$	PSpace
$\mathcal{SHIF}$	ExpTime
$\mathcal{SHOIN}$	NExpTime

Table 3.3: Examples of description logics complexity [19, 96]

The algorithms that are used in implemented reasoners to compute subsumption can be divided into two groups — structural and logical. Structural algorithms compare normalized syntactic structure of the two concepts. It can be shown that these algorithms are sound; however, they have problems with completeness, especially when more expressive logics are used. Because of this problem, logical algorithms are used almost exclusively today. Logical algorithms verify subsumption $C \sqsubseteq D$ by verifying that $C \sqcap \neg D$ is not satisfiable. There are different approaches how to verify it, including translation of the query to existing predicate logic reasoner. The most dominant approach is tableau algorithm.

Tableau algorithm tries to prove satisfiability of a concept C by constructing a model, an interpretation $\mathcal{I}$ in which $\mathcal{D}^{\mathcal{I}}$ is not empty. A tableau is a graph representing such a model. In the tableau graph nodes correspond to individuals (elements of $\Delta^{\mathcal{I}}$) and edges correspond to relationships between individuals (elements of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$). The algorithm starts with an individual satisfying D and applies expansion rules until either no further inferences are possible or until a contradiction has been found. The expansion rules are specific for a particular description logic and consist of two parts. The head

of the rule states conditions for applying the rule, and the body of the rule states how to expand the tableau. For details as well as for other techniques for optimizing performance see [19].

Chapter 4

Working with Ontologies

After an ontology is developed, it is used, reused, and related to other ontologies, and also needs to be maintained. These tasks may be easier when an ontology is designed with these tasks in mind. For example, building ontology on an shared upper ontology and using a modular design usually means easier use and maintenance. In this chapter we describe operations on ontologies, relations between ontologies, and a classification of ontologies.

4.1 Operations on Ontologies

It is possible that one application uses multiple ontologies, especially when using modular design of ontologies or when we need to integrate with systems that use other ontologies. In this case, some operations on ontologies may be needed in order to work with all of them. We will summarize some of these operations [86, 66, 58, 63]. The terminology in this areas is still not stable and different authors may use these terms in a bit shifted meaning, and so the terms may overlap, however, all of these operations are important for maintenance and integration of ontologies.

Merge of ontologies means creation of a new ontology by linking up the existing ones. Conventional requirement is that the new ontology contains all the knowledge from the original ontologies, however, this requirement does not have to be fully satisfied, since the original ontologies may not be together totally consistent. In that case the new ontology imports selected knowledge from the original ontologies so that the result is consistent. The merged ontology may introduce new concepts and relations that serve as a bridge between terms from the original ontologies.

Mapping from one ontology to another one is expressing of the way how to translate statements from ontology to the other one. Often it means

translation between concepts and relations. In the simplest case it is mapping from one concept of the first ontology to one concept of the second ontology. It is not always possible to do such one to one mapping. Some information can be lost in the mapping. This is permissible, however mapping may not introduce any inconsistencies.

Alignment is a process of mapping between ontologies in both directions whereas it is possible to modify original ontologies so that suitable translation exists (i.e., without losing information during mapping). Thus it is possible to add new concepts and relations to ontologies that would form suitable equivalents for mapping. The specification of alignment is called *articulation*. Alignment, as well as mapping, may be partial only.

Refinement is mapping from ontology A to another ontology B so that every concept of ontology A has equivalent in ontology B, however primitive concepts from ontology A may correspond to non-primitive (defined) concepts of ontology B. Refinement defines partial ordering of ontologies.

Unification is aligning all of the concepts and relations in ontologies so that inference in one ontology can be mapped to inference in other ontology and vice versa. Unification is usually made as refinement of ontologies in both directions.

Integration is a process of looking for the same parts of two different ontologies A and B while developing new ontology C that allows to translate between ontologies A and B and so allows interoperability between two systems where one uses ontology A and the other uses ontology B. The new ontology C can replace ontologies A and B or can be used as an interlingua for translation between these two ontologies. Depending on the differences between A and B, new ontology C may not be needed and only translation between A and B is the result of integration. In other words, depending on the number of changes between ontologies A and B during development of ontology C the level of integration can range from alignment to unification.

Inheritance means that ontology A inherits everything from ontology B. It inherits all concepts, relations and restrictions or axioms and there is no inconsistency introduced by additional knowledge contained in ontology A. This term is important for modular design of ontologies (see later) where an upper ontology describes general knowledge and a lower application ontology adds knowledge needed only for the particular application. Inheritance defines partial ordering between ontologies.

Not all of these operations can be made for all ontologies. In general, these are very difficult tasks that are in general not solvable automatically — for example because of undecidability when using very expressive logical languages or because of insufficient specification of an ontology that is not enough to find similarities with another ontology. Because of these rea-

sons these tasks are usually made manually or semi-automatically, where a machine helps to find possible relations between elements from different ontologies, but the final confirmation of the relation is left on human. Human then decides based on natural language description of the ontology elements or decides only based on the natural language names of the ontology elements and common sense.

An example of such a tool is Chiamera [63] that helps with merging ontologies. It provides suggestions for subsumption, disjointness or instance relationship. These suggestions are generated heuristically and are provided for an operator, so that he may choose which one will be actually used. PROMT or SMART [66, 58] system is a similar system that provides suggestions based on linguistic similarity, ontology structure and user actions. It points the user to possible effects of these changes.

4.2 Relationships between Ontologies

In the previous section, we have shown operations that relate ontologies. A classification of the degree of translatability between two ontologies is provided in [42]:

- Extension — ontology O_1 extends ontology O_2. The ontology O_1 extends or includes the ontology O_2. Informally this means that all the symbols that are defined within the O_2 are found in the O_1 together with the restrictions, meanings and other axiomatic relations of these symbols from O_2.

- Identical — ontologies O_1 and O_2 are identical. Vocabulary, axiomatization and the language are physically identical, but the name can be different.

- Equivalent — ontologies O_1 and O_2 are equivalent. Logical vocabulary and logical axiomatization are the same, but the language (syntax) is different. When O_1 and O_2 are equivalent then they are strongly translatable in both ways.

- Strongly-translatable — source ontology O_1 is strongly translatable to the target ontology O_2. The vocabulary of O_1 can be totally translated to the vocabulary of O_2, axiomatization from O_1 holds in O_2, there is no loss of information from O_1 to O_2 and there is no introduction of inconsistency. Note that the representation languages can still be different.

- Weakly-translatable — source ontology O_1 is weakly translatable to the target ontology O_2. The translation permits some loss of information (e.g. the terms are simplified in O_2), but doesn't permit introduction of inconsistency.
- Approx-translatable — source ontology O_1 is approximately translatable to the target ontology O_2. The translation permits even introduction of inconsistencies, i.e. some of the relations become no more valid and some constraints do not apply anymore.

Like with the operations in the previous section 4.1, deciding the relationship between two ontologies often requires manual intervention. Also, one is usually interested also in the way of translating between translatable ontologies. Once the relationship is known, it can be used for deciding what ontologies to select for particular purposes.

4.3 Modularization of Ontologies

The purpose of authoring ontologies is also reusing of knowledge. Once ontology is created for a domain, it should be (at least to some degree) reusable for other applications in the same domain. To simplify both ontology development and reuse, modular design is beneficial. The modular design uses inheritance of ontologies — upper ontologies describe general knowledge, and application ontologies describe knowledge for a particular application, as illustrated in the figure 4.1.

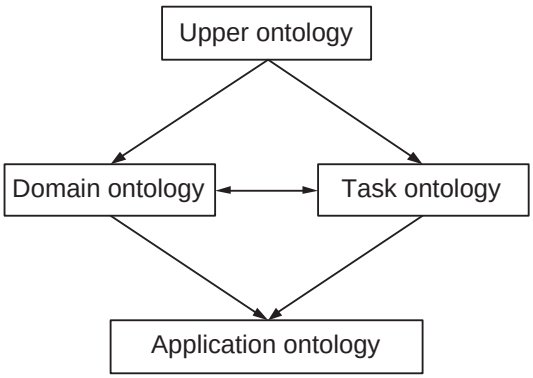

Figure 4.1: Modularization of ontologies depending on the scope and partial ordering defined by inheritance.

Depending on the scope of the ontology, ontology may be classified as follows (see also figure 4.1):

- upper, generic, top-level ontology — describing general knowledge, such as what is time and what is space
- domain ontology — describing a domain, such as medical domain or electrical engineering domain, or narrower domains, such as personal computers domain
- task — ontology suitable for a specific task, such as assembling parts together
- application — ontology developed for a specific application, such as assembling personal computers

At each level modularization can be used as well — for example, upper ontology may consist of modules for real numbers, topology, time, and space (these parts of the upper ontology are usually called generic ontologies). Ontologies at lower levels import ontologies from upper levels and add additional specific knowledge. In this way, ontologies form a lattice of ontologies [86] defined by partial ordering of inheritance of ontologies. Task and domain ontologies may be independent and are merged for application ontology, or it is possible that for example task ontology imports domain ontology. The upper ontologies are the most reused ones while application ontologies may be suitable for one application only.

When developing new ontology it is desirable to reuse existing ontologies as much as possible. The new ontology should be started by importing upper level ontologies when appropriate ontologies exist. This will simplify the development since one can focus at the domain or application specific knowledge only. It will also simplify integration between applications in the future since defined parts of ontologies will be shared.

Chapter 5

Semantic Web

Current World Wide Web (WWW) is a huge library of interlinked documents that are transferred by computers and presented to people. It has grown from hypertext systems, but the difference is that anyone can contribute to it. This also means that the quality of information or even the persistence of documents cannot be generally guaranteed. Current WWW contains a lot of information and knowledge, but machines usually serve only to deliver and present the content of documents describing the knowledge. People have to connect all the sources of relevant information and interpret them themselves.

Semantic web [11] is an effort to enhance current web so that computers can process the information presented on WWW, interpret and connect it, to help humans to find required knowledge. In the same way as WWW is a huge distributed hypertext system, semantic web is intended to form a huge distributed knowledge based system. The focus of semantic web is to share data instead of documents. In other words, it is a project that should provide [11] *a common framework that allows data to be shared and reused across application, enterprise, and community boundaries.* It is a collaborative effort led by World Wide Web Consortium (W3C).

The architecture of semantic web is illustrated in the figure 5.1. The first layer, URI and Unicode, follows the important features of the existing WWW. Unicode [10] is a standard of encoding international character sets and it allows that all human languages can be used (written and read) on the web using one standardized form. Uniform Resource Identifier [25] (URI) is a string of a standardized form that allows to uniquely identify resources (e.g., documents). A subset of URI is Uniform Resource Locator (URL), which contains access mechanism and a (network) location of a document — such as `http://www.example.org/`. Another subset of URI is URN that allows to identify a resource without implying its location and means of dereferencing it — an example is `urn:isbn:0-123-45678-9`. The usage of URI is important

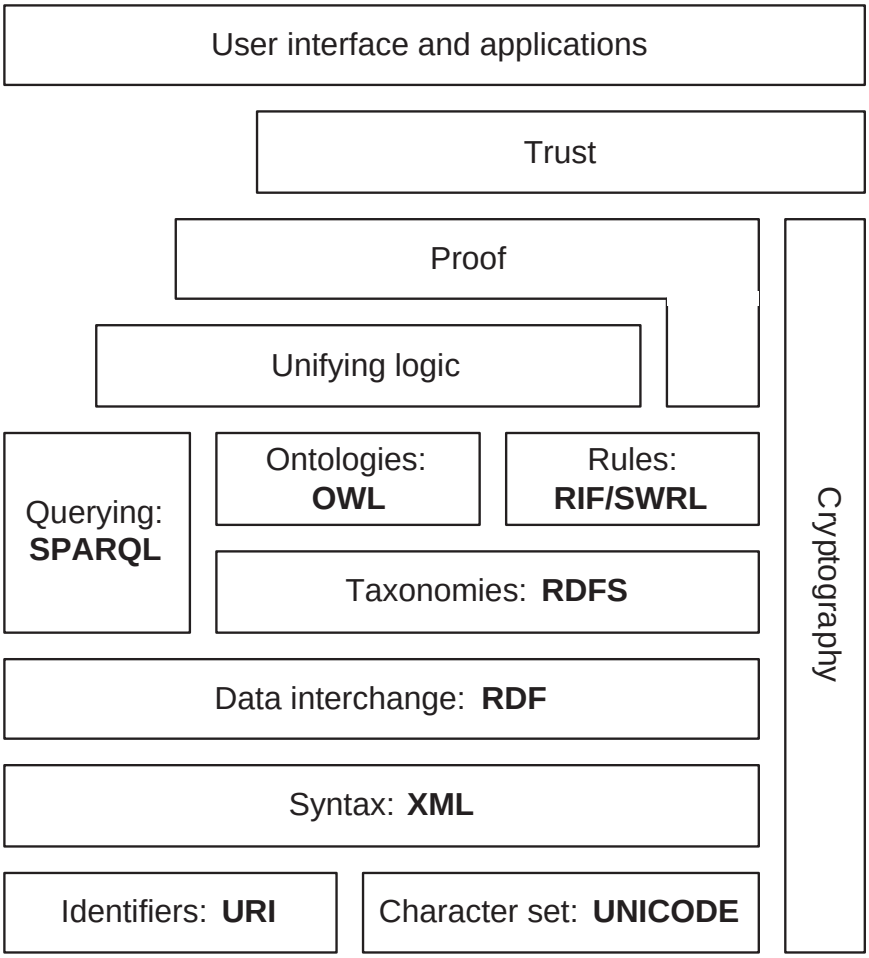

Figure 5.1: Semantic web architecture in layers

for a distributed internet system as it provides understandable identification of all resources. An international variant to URI is Internationalized Resource Identifier (IRI) that allows usage of Unicode characters in identifier and for which a mapping to URI is defined. In the rest of this text, whenever URI is used, IRI can be used as well as a more general concept.

Extensible Markup Language [28] (XML) layer with XML namespace [27] and XML schema [12] definitions makes sure that there is a common syntax used in the semantic web. XML is a general purpose markup language for documents containing structured information. A XML document contains elements that can be nested and that may have attributes and content. XML namespaces allow to specify different markup vocabularies in one XML document. XML schema serves for expressing schema of a particular set of

XML documents.

A core data representation format for semantic web is Resource Description Framework (RDF) [61]. RDF is a framework for representing information about resources in a graph form. It was primarily intended for representing metadata about WWW resources, such as the title, author, and modification date of a Web page, but it can be used for storing any other data. It is based on triples *subject–predicate–object* that form graph of data. All data in the semantic web use RDF as the primary representation language. The normative syntax for serializing RDF is XML in the RDF/XML [20] form. Formal semantics for RDF is defined as well [54].

RDF itself serves as a description of a graph formed by triples. Anyone can define vocabulary of terms used for more detailed description. To allow standardized description of taxonomies and other ontological constructs, a RDF Schema [29] (RDFS) was created together with its formal semantics [54] within RDF. RDFS can be used to describe taxonomies of classes and properties and use them to create lightweight ontologies.

More detailed ontologies can be created with Web Ontology Language OWL [14]. The OWL is a language derived from description logics, and offers more constructs over RDFS. It is syntactically embedded into RDF, so like RDFS, it provides additional standardized vocabulary. OWL comes in three species — OWL Lite for taxonomies and simple constrains, OWL DL for full description logic support, and OWL Full for maximum expressiveness and syntactic freedom of RDF. Since OWL is based on description logic, it is not surprising that a formal semantics is defined [80] for this language.

RDFS and OWL have semantics defined and this semantics can be used for reasoning within ontologies and knowledge bases described using these languages. To provide rules beyond the constructs available from these languages, rule languages are being standardized for the semantic web as well. Two standards are emerging — RIF [17] and SWRL [55].

For querying RDF data as well as RDFS and OWL ontologies with knowledge bases, a Simple Protocol and RDF Query Language (SPARQL) [82] is available. SPARQL is SQL-like language, but uses RDF triples and resources for both matching part of the query and for returning results of the query. Since both RDFS and OWL are built on RDF, SPARQL can be used for querying ontologies and knowledge bases directly as well. Note that SPARQL is not only query language, it is also a protocol for accessing RDF data.

It is expected that all the semantics and rules will be executed at the layers below Proof and the result will be used to prove deductions. Formal proof together with trusted inputs for the proof will mean that the results can be trusted, which is shown in the top layer of the figure 5.1. For reliable inputs, cryptography means are to be used, such as digital signatures for

verification of the origin of the sources. On top of these layers, application with user interface can be built.

In the rest of this chapter, we will describe some of these technologies — RDF, RDFS, OWL, and SPARQL.

5.1 Resource Description Framework (RDF)

Resource Description Framework (RDF) [61, 20, 54] is a framework for representing information about resources in a graph form. Since it was primarily intended for representing metadata about WWW resources, it is built around resources with URI.

Information is represented by triples *subject–predicate–object*. An example of a triple is shown in the figure 5.2. It says that "Joe Smith has homepage `http://www.example.org/~joe/`". All elements of this triple are resources defined by URI. The first resource `http://www.example.org/~joe/contact.rdf#joesmith` (*subject*) is intended to identify Joe Smith. Note that it precisely defines how to get to a RDF document as well as how to get the `joesmith` RDF node in it. The second resource `http://xmlns.com/foaf/0.1/homepage` (*predicate*) is the predicate homepage from a FOAF (Friend-of-a-friend) vocabulary [30]. The last resource (*object*) is Joe's homepage `http://www.example.org/~joe/`.

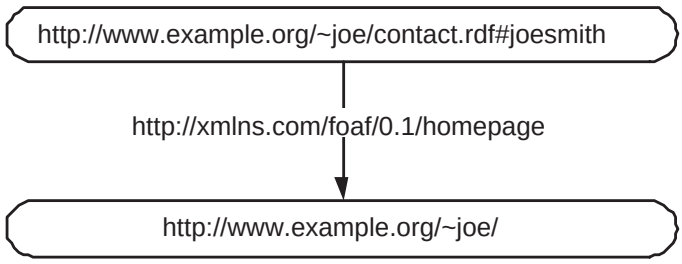

Figure 5.2: RDF triple (in graph representation) describing Joe Smith — "Joe has homepage identified by URI"

All of the elements of the triple are resources with the exception of the last element, object, that can be also a literal. Literal in the RDF sense is a constant string value such as string or number. Literals can be either plain literals (without type) or typed literals typed using XML Datatypes [26]. An example of literal usage is illustrated in the triple shown in the figure 5.3.

These triples together form RDF graph. A graph with the triples from figures 5.2 and 5.3 and with some additional triples are shown in the figure

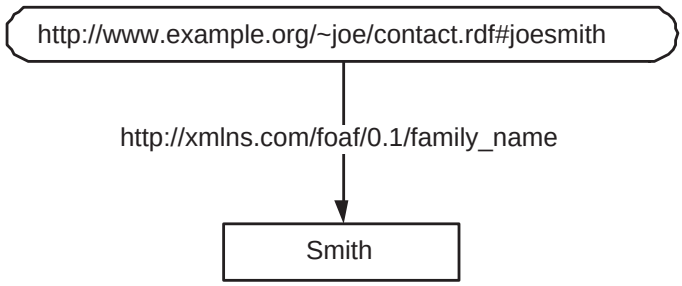

Figure 5.3: RDF triple (in graph representation) describing Joe Smith — "Joe has family name *Smith*"

5.4. The top triple uses `type` as a predicate from RDF vocabulary to express that `joesmith` is of type `Person`.

A normative syntax for serializing RDF is RDF/XML [20]. The RDF graph from the figure 5.4 is written in RDF/XML as follows. Note that it uses XML namespaces with prefixes defined in the beginning of the XML document.

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns="http://www.example.org/~joe/contact.rdf#">
  <foaf:Person rdf:about=
        "http://www.example.org/~joe/contact.rdf#joesmith">
    <foaf:mbox rdf:resource="mailto:joe.smith@example.org"/>
    <foaf:homepage
            rdf:resource="http://www.example.org/~joe/"/>
    <foaf:family_name>Smith</foaf:family_name>
    <foaf:givenname>Joe</foaf:givenname>
  </foaf:Person>
</rdf:RDF>
```

RDF/XML is a normative syntax, however, other serialization formats are used as well. The TURTLE and N3 syntax is less verbose than RDF/XML and so is quite popular. The *Notation 3* [24] (N3) is designed as a readable language for data on the Web that goes beyond RDF (it contains logical extensions and rules). The *Terse RDF Triple Language* [21] (TURTLE) is a RDF-only subset of N3. For the purposes of this thesis these two languages are interchangeable. An example of N3 serialization of the graph from the figure 5.4 follows.

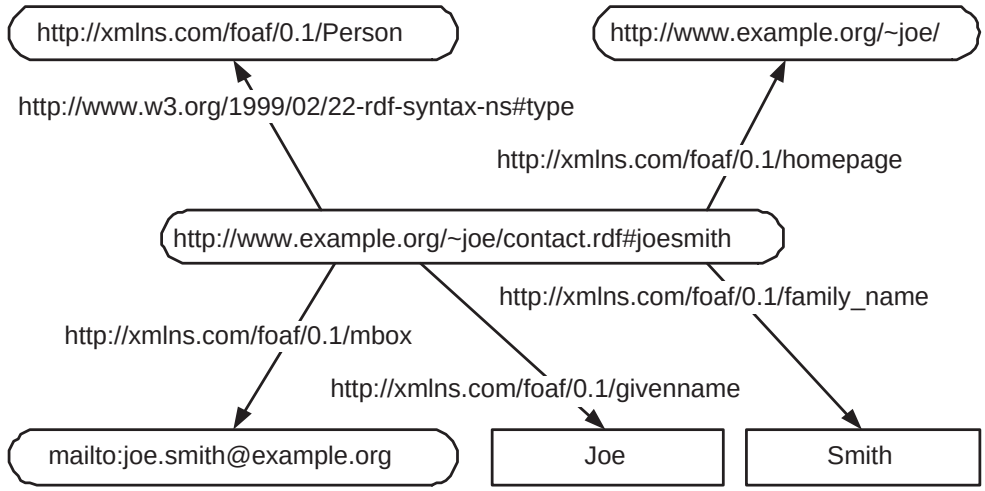

Figure 5.4: RDF graph describing Joe Smith

```
@prefix :     <http://www.example.org/~joe/contact.rdf#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

:joesmith a foaf:Person ;
      foaf:givenname "Joe" ;
      foaf:family_name "Smith" ;
      foaf:homepage <http://www.example.org/~joe/> ;
      foaf:mbox <mailto:joe.smith@example.org> .
```

The `a` in the first triple is a syntactic shortcut for `rdf:type`. Note that it N3 is similar to the linear form of conceptual graphs serialization. In the rest of this thesis we will use primarily the N3/TURTLE notation when expressing RDF. The same syntax is used for SPARQL [82] RDF querying language.

We may not need to specify URI for every RDF node in a graph. A node without URI called *blank node* (abbreviated *b-node*) and can be viewed as a graph scoped identifier that cannot be directly referenced from outside. A blank node can be used in any triple only as a subject or an object and cannot be used as a predicate. Blank node is useful for example when specifying lists (see figure 5.5).

The RDF language namespace prefix is usually `rdf:` and is (syntactically) defined at `http://www.w3.org/1999/02/22-rdf-syntax-ns#`. RDF vocabulary includes the following elements:

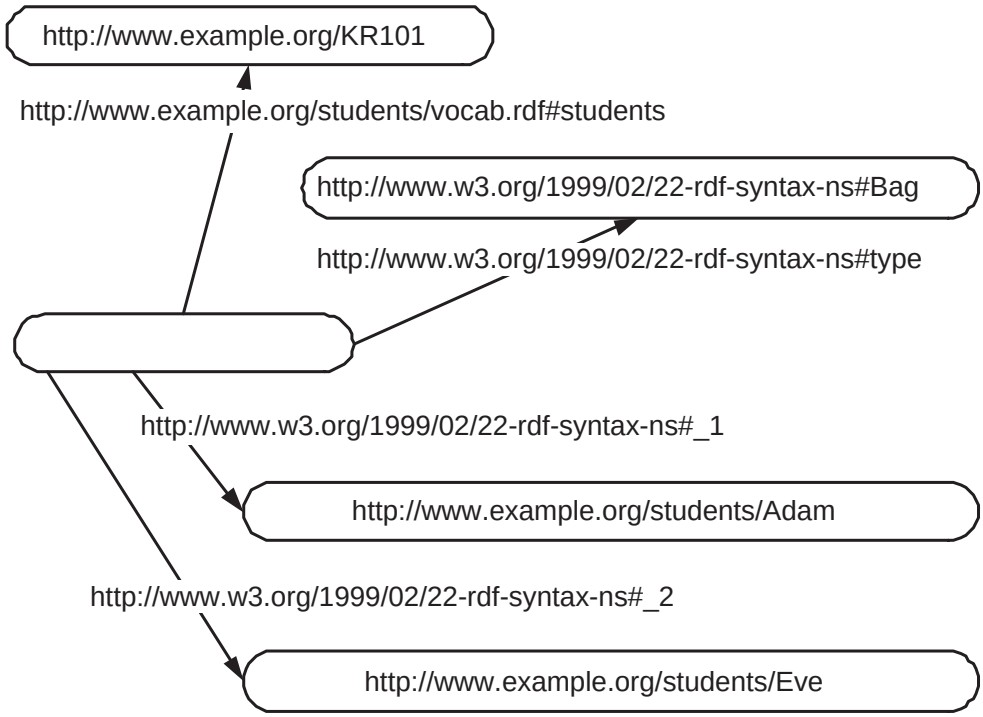

Figure 5.5: Illustration of a bag (unordered container) and a blank node.

- `rdf:type` — it is a predicate used to state that a resource is an instance of a class
- `rdf:XMLLiteral` — the class of typed literals (i.e., of XML literal values)
- `rdf:Property` — the class of properties (i.e. binary relations that are used as predicates in triples)
- `rdf:Alt`, `rdf:Bag`, `rdf:Seq` – containers of alternatives, unordered containers, and ordered containers (see figure 5.5 for an example)
- `rdf:List` — the class of RDF Lists
- `rdf:nil` — an instance of `rdf:List` representing the empty list
- `rdf:Statement`, `rdf:subject`, `rdf:predicate`, `rdf:object` — used for reification (described below)

These elements are further discussed in the next section about RDFS, since RDFS adds further constrains on them to specify their meaning more precisely.

Let us only further explain RDF reification. RDF reification allows to disassemble a statement (triple) to its parts and to use the whole statement or parts of the statement as a part of other triples. The whole triple can then be treated as a resource which allows to make assertions about the statement. For example, for the statement

```
:john  :has  :cat
```

the RDF reification is as follows (note that the result is resource that can for example participate as a subject in another triple):

```
[ a rdf:Statement;
    rdf:subject :john;
    rdf:predicate :has;
    rdf:object :cat ].
```

To summarize, RDF triple is a triple $< subject, predicate, object >$ where *subject* can be URI or b-node, *predicate* can be URI, and *object* can be URI, b-node, or literal. RDF graph is a set of RDF triples. Formal semantics for RDF is defined using model theory and is available in [54].

5.2 RDF Schema (RDFS)

RDF Schema [29] (RDFS) is extending RDF vocabulary to allow describing taxonomies of classes and properties. It also extends definitions for some of the elements of RDF, for example it sets the domain and range of properties and relates the RDF classes and properties into taxonomies using the RDFS vocabulary.

Let us first illustrate the use of RDFS vocabulary on an example showing taxonomy of classes and properties and usage of range and domain of properties:

```
@prefix :     <http://www.example.org/sample.rdfs#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

:Dog      rdfs:subClassOf :Animal.
:Person   rdfs:subClassOf :Animal.
```

Element	Class of	rdfs:subClassOf	rdf:type
rdfs:Resource	all resources	rdfs:Resource	rdfs:Class
rdfs:Class	all classes	rdfs:Resource	rdfs:Class
rdfs:Literal	literal values	rdfs:Resource	rdfs:Class
rdfs:Datatype	datatypes	rdfs:Class	rdfs:Class
rdf:XMLLiteral	XML literal values	rdfs:Literal	rdfs:Datatype
rdf:Property	properties	rdfs:Resource	rdfs:Class
rdf:Statement	statements	rdfs:Resource	rdfs:Class
rdf:List	lists	rdfs:Resource	rdfs:Class
rdfs:Container	containers	rdfs:Resource	rdfs:Class
rdf:Bag	unordered containers	rdfs:Container	rdfs:Class
rdf:Seq	ordered containers	rdfs:Container	rdfs:Class
rdf:Alt	containers of alternatives	rdfs:Container	rdfs:Class
rdfs:Container MembershipProperty	`rdf:_1`... properties expressing membership	rdf:Property	rdfs:Class

Table 5.1: RDFS classes

```
:hasChild rdfs:range :Animal;
          rdfs:domain :Animal.
:hasSon   rdfs:subPropertyOf :hasChild.

:Max      a :Dog.
:Abel     a :Person.
:Adam     a :Person;
          :hasSon :Abel.
```

The list of classes defined by RDFS is shown in the table 5.1. All resources can be divided into groups called classes. Classes are also resources, so they are identified by URIs and can be described using properties. The members of a class are instances of classes, which is stated using the `rdf:type` property. Note that class and a set of instances does not have to be the same. The set of instances is the extension of the class, and two different classes may contain the same set of instances. For example, looking at the graph 5.4, class of people having mailbox `mailto:joe.smith@example.org` is different from the class of people having homepage `http://www.example.org/~joe/`, but the extension (i.e., the set of instances) of these classes is the same — it is the resource `http://www.example.org/~joe/contact.rdf#joesmith`.

In RDFS a class may be an instance of a class. All resources are instances of the class `rdfs:Resource`. All classes are instances of `rdfs:Class` and subclasses of `rdfs:Resource`. All literals are instances of `rdfs:Literal`. All

Element	Relates	rdfs:domain	rdfs:range
rdfs:range	restricts subjects	rdf:Property	rdfs:Class
rdfs:domain	restricts objects	rdf:Property	rdfs:Class
rdf:type	instance of	rdfs:Resource	rdfs:Class
rdfs:subClassOf	subclass of	rdfs:Class	rdfs:Class
rdfs:subPropertyOf	subproperty of	rdf:Property	rdf:Property
rdfs:label	human readable label	rdfs:Resource	rdfs:Literal
rdfs:comment	human readable comment	rdfs:Resource	rdfs:Literal
rdfs:member	container membership	rdfs:Resource	rdfs:Resource
rdf:first	first element	rdf:List	rdfs:Resource
rdf:rest	rest of list	rdf:List	rdf:List
rdf:_1, rdf:_2, ...	container membership	rdfs:Container	rdfs:Resource
rdfs:seeAlso	further information	rdfs:Resource	rdfs:Resource
rdfs:isDefinedBy	definition	rdfs:Resource	rdfs:Resource
rdf:value	for structured values	rdfs:Resource	rdfs:Resource
rdf:object	object of statement	rdf:Statement	rdfs:Resource
rdf:predicate	predicate of statement	rdf:Statement	rdfs:Resource
rdf:subject	subject of of statement	rdf:Statement	rdfs:Resource

Table 5.2: RDFS properties (all are instances of `rdf:Property`)

properties are instances of `rdf:Property`. The `rdfs:subClassOf` (subclass-of relation) and `rdf:type` (instance-of relation) for all RDFS classes are shown in the the table 5.1.

Properties in RDFS are relations between subjects and objects in RDF triples, i.e., predicates. The properties as defined by RDFS are listed in the figure 5.2. All properties may have defined domain and range. Domain of a property states that any resource that has given property is an instance of the class. Range of a property states that the values of a property are instances of the class. If multiple classes are defined as the domain and range then the intersection of these classes is used. Ranges and domains for RDFS properties are summarized in the figure 5.2. An example stating that the domain of `hasSon` property is `Person` and that the domain of the same property is `Man` follows:

```
@prefix :        <http://www.example.org/sample.rdfs#> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#>.

:hasSon rdfs:domain :Person;
        rdfs:range  :Man.
```

The taxonomy of classes is formed by property `rdfs:subClassOf`, taxonomy of properties is formed by property `rdfs:subPropertyOf`.

Element	Meaning	rdfs:type
rdfs:nil	empty list	rdf:List

Table 5.3: RDFS instance

RDFS also defines one instance that is neither class nor property. This instance denotes empty list: `rdfs:nil`, see the table 5.3.

5.3 Web Ontology Language (OWL)

The Web Ontology Language OWL [14] extends RDF and RDFS. Its primary aim is to bring the expressive and reasoning power of description logic to the semantic web. Unfortunately, not everything from RDF can be expressed in DL. For example, the classes of classes are not permitted in the (chosen) DL, and some of the triple expressions would have no sense in DL. That is why OWL can be only syntactic extension of RDF/RDFS (note that RDFS is both syntactic and semantic extension of RDF). To partially overcome this problem, and also to allow layering within OWL, three species of OWL are defined.

OWL Lite can be used to express taxonomy and simple constraints, such as 0 and 1 cardinality. It is the simplest OWL language and corresponds to description logic $\mathcal{SHIF}$. *OWL DL* supports maximum expressiveness while retaining computational completeness and decidability. The DL in the name shows that it is intended to support description logic capabilities. OWL DL corresponds to description logic $\mathcal{SHOIN}$. *OWL Full* has no expressiveness constraints, but also does not guarantee any computational properties. It is formed by the full OWL vocabulary, but does not no impose any syntactic constrains, so that the full syntactic freedom of RDF can be used.

These three languages are layered in a sense that every legal OWL Lite ontology is a legal OWL DL ontology, every legal OWL DL ontology is a legal OWL Full ontology, every valid OWL Lite conclusion is a valid OWL DL conclusion, and every valid OWL DL conclusion a valid OWL Full conclusion. The inverses of these relations generally do not hold. Also, every OWL ontology is a valid RDF document (i.e., DL expressions are mapped to triples), but not all RDF documents are valid OWL Lite or OWL DL documents. In this thesis, we are interested primarily in OWL DL. If we will not indicate otherwise, we mean OWL DL by OWL in the rest of the text.

The OWL DL descriptions, data ranges, properties, individuals and data values syntax and semantics are summarized in the table 5.4, OWL DL axioms and facts are summarized in the table 5.5. As we can see, OWL DL is

Abstract Syntax	DL Syntax	Semantics
Descriptions (C)		
A (URI Reference)	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
`owl:Thing`	$\top$	$\texttt{owl:Thing}^{\mathcal{I}} = \Delta^{\mathcal{I}}$
`owl:Nothing`	$\bot$	$\texttt{owl:Nothing}^{\mathcal{I}} = \emptyset$
`intersectionOf`($C_1\ C_2 \ldots$)	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
`unionOf`($C_1\ C_2\ \ldots$)	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
`complementOf`(C)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
`oneOf`($o_1 \ldots$)	$\{o_1, \ldots\}$	$\{o_1^{\mathcal{I}}, \ldots\}$
`restriction`(R `someValuesFrom`(C))	$\exists R.C$	$\{x \mid \exists y\ \ (x,y) \in R^{\mathcal{I}} \cup y \in C^{\mathcal{I}}\}$
`restriction`(R `allValuesFrom`(C))	$\forall R.C$	$\{x \mid \forall y\ \ (x,y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
`restriction`(R `hasValue`(o))	$R : o$	$\{x \mid (x, o^{\mathcal{I}}) \in R^{\mathcal{I}}\}$
`restriction`(R `minCardinality`(n))	$\geqslant nR$	$\{a \in \Delta^{\mathcal{I}} \mid\ \mid\{b \mid (a,b) \in R^{\mathcal{I}}\}\mid \geq n\}$
`restriction`(R `maxCardinality`(n))	$\leqslant nR$	$\{a \in \Delta^{\mathcal{I}} \mid\ \mid\{b \mid (a,b) \in R^{\mathcal{I}}\}\mid \leq n\}$
`restriction`(U `someValuesFrom`(D))	$\exists U.D$	$\{x \mid \exists y\ \ (x,y) \in U^{\mathcal{I}} \cup y \in D^{\mathcal{D}}\}$
`restriction`(U `allValuesFrom`(D))	$\forall U.D$	$\{x \mid \forall y\ \ (x,y) \in U^{\mathcal{I}} \rightarrow y \in D^{\mathcal{D}}\}$
`restriction`(U `hasValue`(v))	$U : v$	$\{x \mid (x, v^{\mathcal{I}}) \in U^{\mathcal{I}}\}$
`restriction`(U `minCardinality`(n))	$\geqslant nU$	$\{a \in \Delta^{\mathcal{I}} \mid\ \mid\{b \mid (a,b) \in U^{\mathcal{I}}\}\mid \geq n\}$
`restriction`(U `maxCardinality`(n))	$\leqslant nU$	$\{a \in \Delta^{\mathcal{I}} \mid\ \mid\{b \mid (a,b) \in U^{\mathcal{I}}\}\mid \leq n\}$
Data Ranges (D)		
D (URI reference)	D	$D^{\mathcal{D}} \subseteq \Delta_{\mathcal{D}}^{\mathcal{I}}$
`oneOf`($v_1 \ldots,$)	$\{v_1 \ldots,\}$	$\{v_1^{\mathcal{I}} \ldots,\}$
Object Properties (R)		
R (URI reference)	R	$\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
	R^-	$(R^{\mathcal{I}})^-$
Datatype Properties (U)		
U (URI reference)	U	$U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta_{\mathcal{D}}^{\mathcal{I}}$
Individuals (o)		
o (URI reference)	o	$o^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
Data Values (v)		
v (RDF literal)	v	$v^{\mathcal{D}}$

Table 5.4: OWL DL descriptions, data ranges, properties, individuals and data values syntax and semantics [56, 80, 19]

the description logic $\mathcal{SHOIN}$ with support of data values, data types and datatype properties, i.e., $\mathcal{SHOIN}(\mathcal{D})$, but since OWL is based on RDF(S), the terminology slightly differs. A concept from DL is referred to as a class in OWL and a role from DL is referred to as a property in OWL. For description of OWL ontology or knowledge base, the DL syntax can be used. There is an "abstract" LISP-like syntax defined [80] that is easier to write in ASCII character set (see also tables 5.4 and 5.5). Since OWL is syntactically embedded into RDF, all of the RDF serializations can be used. RDF/XML

Abstract Syntax	DL Syntax	Semantics
Classes		
`Class`(A `partial` $C_1 \dots C_n$)	$A \sqsubseteq C_1 \sqcap \dots \sqcap C_n$	$A^{\mathcal{I}} \subseteq C_1^{\mathcal{I}} \cap \dots \cap C_n^{\mathcal{I}}$
`Class`(A `complete` $C_1 \dots C_n$)	$A \equiv C_1 \sqcap \dots \sqcap C_n$	$A^{\mathcal{I}} = C_1^{\mathcal{I}} \cap \dots \cap C_n^{\mathcal{I}}$
`EnuneratedClass`($A\ o_1 \dots o_n$)	$A \equiv \{o_1, \dots, o_n\}$	$A^{\mathcal{I}} = \{o_1^{\mathcal{I}}, \dots, o_n^{\mathcal{I}}\}$
`SubClassOf`($C_1\ C_2$)	$C_1 \sqsubseteq C_2$	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
`EquivalentClasses`($C_1 \dots C_n$)	$C_1 \equiv \dots \equiv C_n$	$C_1^{\mathcal{I}} = \dots = C_n^{\mathcal{I}}$
`DisjointClasses`($C_1 \dots C_n$)	$C_i \sqcap C_j = \bot, i \neq j$	$C_i^{\mathcal{I}} \cap C_j^{\mathcal{I}} = \emptyset, i \neq j$
`Datatype`(D)		$D^{\subset} \Delta_{\mathcal{D}}^{\mathcal{I}}$
Datatype Properties		
`DatatypeProperty`(		
U `super`(U_1) $\dots$ `super`(U_n)	$U \sqsubseteq U_i$	$U^{\mathcal{I}} \subseteq U_i^{\mathcal{I}}$
`domain`(C_1) $\dots$ `domain`(C_m)	$\geqslant 1\ U \sqsubseteq C_i$	$U^{\mathcal{I}} \subseteq C_i^{\mathcal{I}} \times \Delta_{\mathcal{D}}^{\mathcal{I}}$
`range`(D_1) $\dots$ `range`(D_l)	$\top \sqsubseteq \forall U.D_i$	$U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times D_i^{\mathcal{I}}$
`[Functional])`	$\top \sqsubseteq \leqslant 1U$	U_i is functional
`SubPropertyOf`($U_1\ U_2$)	$U_1 \sqsubseteq U_2$	$U_1^{\mathcal{I}} \subseteq U_2^{\mathcal{I}}$
`EquivalentProperties`($U_1 \dots U_n$)	$U_1 \equiv \dots \equiv U_n$	$U_1^{\mathcal{I}} = \dots = U_n^{\mathcal{I}}$
Object Properties		
`ObjectProperty`(		
R `super`(R_1) $\dots$ `super`(R_n)	$R \sqsubseteq R_i$	$R^{\mathcal{I}} \subseteq R_i^{\mathcal{I}}$
`domain`(C_1) $\dots$ `domain`(C_m)	$\geqslant 1\ R \sqsubseteq C_i$	$R^{\mathcal{I}} \subseteq C_i^{\mathcal{I}} \times \Delta_{\mathcal{D}}^{\mathcal{I}}$
`range`(C_1) $\dots$ `range`(C_l)	$\top \sqsubseteq \forall R.C_i$	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times C_i^{\mathcal{I}}$
`[inverseOf(`R_0`)]`	$R \equiv (R_0^-)$	$R^{\mathcal{I}} = (R_0^{\mathcal{I}})^-$
`[Symmetric]`	$R \equiv (R^-)$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^-$
`[Functional]`	$\top \sqsubseteq \leqslant 1R$	$R^{\mathcal{I}}$ is functional
`[InverseFunctional]`	$\top \sqsubseteq \leqslant 1R^-$	$(R^{\mathcal{I}})^-$ is functional
`[Transitive])`	$Tr(R)$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^+$
`SubPropertyOf`($R_1\ R_2$)	$R_1 \sqsubseteq R_2$	$R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$
`EquivalentProperties`($R_1 \dots R_n$)	$R_1 \equiv \dots \equiv R_n$	$R_1^{\mathcal{I}} = \dots = R_n^{\mathcal{I}}$
Annotation		
`AnnotationProperty`(S)		
Individuals		
`Individual`(		
o `type`(C_1) $\dots$ `type`(C_n)	$o \in C_i$	$o^{\mathcal{I}} \in C_i^{\mathcal{I}}$
`value`($R_1\ o_1$) $\dots$ `value`($R_n\ o_n$)	$\{o, o_i\} \in R_i$	$\{o^{\mathcal{I}}, o_i^{\mathcal{I}}\} \in R_i^{\mathcal{I}}$
`value`($U_1\ v_1$) $\dots$ `value`($U_n\ v_n$))	$\{o, v_i\} \in U_i$	$\{o^{\mathcal{I}}, v_i^{\mathcal{I}}\} \in U_i^{\mathcal{I}}$
`SameIndividual`($o_1 \dots o_n$)	$o_1 = \dots = o_n$	$o_1^{\mathcal{I}} = \dots = o_n^{\mathcal{I}}$
`DifferentIndividual`($o_1 \dots o_n$)	$o_i \neq o_j, i \neq j$	$o_i^{\mathcal{I}} \neq o_j^{\mathcal{I}}, i \neq j$

Table 5.5: OWL DL axioms and facts [56, 80, 19]

is the normative syntax and should be used to exchange information between systems.

Let us illustrate the use of OWL vocabulary on an example ontology

(inspired by [83]): "Pizza has PizzaBase as its base; Pizza is disjoint with PizzaBase; NonVegetarianPizza is exactly Pizza that is not VegetarianPizza; isIngredientOf is a transitive property; isIngredientOf is inverse of hasIngredient". The example expressed in the description logic syntax follows:

$$\textsc{Pizza} \sqsubseteq \exists\, \mathsf{hasBase}.\textsc{PizzaBase}$$
$$\textsc{Pizza} \sqcap \textsc{PizzaBase} \equiv \bot$$
$$\textsc{NonVegetarianPizza} \equiv \textsc{Pizza} \sqcap \neg \textsc{VegetarianPizza}$$
$$Tr(\mathsf{isIngredientOf})$$
$$\mathsf{isIngredientOf} \equiv \mathsf{hasIngredient}^{-}$$

The same example expressed using OWL Abstract Syntax formulates the same information using LISP-like notation, and in addition uses URI for identification of all classes and properties:

```
Namespace(p = <http://example.com/pizzas.owl#>)
Ontology( <http://example.com/pizzas.owl#>
  Class(p:Pizza partial
    restriction(p:hasBase someValuesFrom(p:PizzaBase)))
  DisjointClasses(p:Pizza p:PizzaBase)
  Class(p:NonVegetarianPizza complete
    intersectionOf(p:Pizza complementOf(p:VegetarianPizza)))
  ObjectProperty(p:isIngredientOf Transitive
    inverseOf(p:hasIngredient))
)
```

When embedding the example OWL ontology to RDF, every statement must be converted to triples — see figure 5.6. For example, the $\exists R.C$ restriction is formed by anonymous resource of type `owl:Restriction`. This anonymous resource (blank node) is a subject for two properties `owl:onProperty` and `owl:someValuesFrom` that relate the restriction relation (property) and concept (class). The anonymous resource is then used to be related to the constrained class (by `rdfs:subClassOf` in our case). The example expressed in triples and serialized in N3 follows:

```
@prefix :     <http://example.com/pizzas.owl#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl:  <http://www.w3.org/2002/07/owl#> .

:Pizza rdfs:subClassOf
        [ a owl:Restriction ;
```

```
          owl:onProperty :hasBase ;
          owl:someValuesFrom :PizzaBase ] ;
       owl:disjointWith :PizzaBase .

:NonVegetarianPizza  owl:equivalentClass
        [  owl:intersectionOf
            (  [owl:complementOf :VegetarianPizza]
               :Pizza  )  ] .

:isIngredientOf
      a owl:TransitiveProperty , owl:ObjectProperty ;
      owl:inverseOf :hasIngredient .
```

The OWL DL uses all the $\mathcal{SHOIN}(\mathcal{D})$ features. The overview of the possible descriptions, data ranges, properties, individuals and data values is shown in the table 5.4. The description of the semantics was introduced in the section 3.6.1. The domain of individuals in the model is $\Delta^{\mathcal{I}}$, the domain of data values $\Delta^{\mathcal{I}}_{\mathcal{D}}$ was added to specify semantics of data ranges.

The ontology is formed by constraints on a model. The axioms that can be used to constrain a model are summarized in the table 5.5.

In addition to the standard description logic features there are so called annotation properties added. In addition to RDFS annotation properties (such as `rdfs:comment` and `rdfs:label`) there are properties that allow to state for example version information, state compatibility or incompatibility between ontologies. There is also a construct `owl:imports` that allows to state that an ontology imports another ontology.

5.4 RDF Query Language SPARQL

The Simple Protocol and RDF Query Language (SPARQL) [82] is a SQL-like language for querying RDF data. For expressing RDF graphs in the matching part of the query, TURTLE syntax is used.

An example of a SELECT query follows.

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name .
        ?x foaf:mbox ?mbox . }
```

The first line defines namespace prefix, the last two lines use the prefix to express a RDF graph to be matched. Identifiers beginning with question

mark `?` identify variables. In this query, we are looking for resource `?x` participating in triples with predicates `foaf:name` and `foaf:mbox` and want the subjects of these triples. Syntactic shortcuts of TURTLE can be used in the matching part.

In addition to specifying graph to be matched, constraints can be added for values using `FILTER` construct. An example of string value restriction is `FILTER regex(?mbox, "company")` that specifies regular expression query. An example of number value restriction is `FILTER (?price < 20)` that specifies that `?price` must be less than 20. A few special operators are defined for the `FILTER` construct. They include `isIRI` for testing whether variable is IRI/URI, `isLiteral` for testing whether variable is literal, `bound` to test whether variable was bound and others [82].

The matching part of the query may include `OPTIONAL` triples. If the triple to be matched is optional, it is evaluated when it is present, but the matching does not fail when it is not present. Optional sections may be nested. It is possible to make `UNION` of multiple matching graphs — if any of the graphs matches, the match will be returned as a result. The `FROM` part of the query is optional and may specify the RDF dataset on which query is performed.

The sequence of result may be modified using the following keywords with the meaning similar to SQL:

- `ORDER BY` — ordering by variable value
- `DISTINCT` — unique results only
- `OFFSET` — offset from which to show results
- `LIMIT` — the maximum number of results.

There are four query result forms. In addition to the possibility of getting the list of values found it is also possible to construct RDF graph or to confirm whether a match was found or not.

- `SELECT` — returns the list of values of variables bound in a query pattern
- `CONSTRUCT` — returns an RDF graph constructed by substituting variables in the query pattern
- `DESCRIBE` — returns an RDF graph describing the resources that were found

- `ASK` — returns a boolean value indicating whether the query pattern matches or not

The `CONSTRUCT` form specifies a graph to be returned with variables to be substituted from the query pattern, such as in the following example that will return graph saying that Alice knows last two people when ordered by alphabet from the given URI (the result in the RDF graph is not ordered, it is a graph and so the order of triples is not important).

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
CONSTRUCT { <http://example.org/person#Alice> foaf:knows ?x }
FROM <http://example.org/foaf/people>
WHERE { ?x foaf:name ?name }
ORDER BY desc(?name)
LIMIT 2
```

The `DESCRIBE` form will return information about matched resources in a form of an RDF graph. The exact form of this information is not standardized yet, but usually a blank node closure like for example Concise Bounded Description (CBD) [88] is expected. In short, all the triples that have the matched resource in the object are returned; when a blank node is in the subject, then the triples in which this node participates as object are recursively added as well.

The `ASK` form is intended for asking yes/no questions about matching — no information about matched variables is returned, the result is only indicating whether matching exists or not.

The SPARQL specification [82] is in the state of working draft, but it is already implemented in some software packages and it seems that it will become the main RDF querying language for the semantic web. The specification of protocol for a SPARQL web service is available as well — SPARQL then serves as a RDF data access protocol.

5.5 Reasoning Support

Tasks expected from a reasoner were already mentioned (in the Description Logic introduction, section 3.6.3). Let us recall that according to the table 3.3 the reasoning in OWL reasoning seems to be intractable for general cases (e.g., NExpTime for OWL DL). However, reasoner engines exist that are able to handle practical cases. In this section we only briefly list available reasoners. For general implementation details and problems we refer to [19].

One group of reasoners is the group originally written as description logic reasoners. Examples are Pellet [6], FACT++ [3] and Racer [7]. They can be accessed using XML interface developed by the DL Implementation Group (DIG) [2]. DIG is a simple protocol based on HTTP PUT/GET along with a XML schema for expressing description logics language statements and queries. Unfortunately, the current DIG interface is not sufficient to capture general OWL DL ontologies. For example, datatype support is lacking in DIG 1.1. Even when not all OWL DL elements can be used as an input for reasoning, the general description logics reasoners can be very useful for supporting reasoning on the semantic web.

Several semantic web packages or frameworks exist that implement their own reasoning support to some degree (for example, RDFS or subset of OWL DL is supported). We will focus on Jena, a Semantic Web Framework for Java [16], that provides its own specialized inference engine to support reasoning in RDFS, OWL Lite, OWL DL and OWL Full. The inference in Jena is not claimed to be complete, but is claimed to be sound. Jena also provides a general purpose rule engine where one can write rules for custom inference. Rules for reasoning within subsets of species of OWL are provided as an example. It is possible to connect the description logic reasoner Pellet [6] directly to Jena to overcome DIG limitations, in which case full OWL DL reasoning should be possible in Jena.

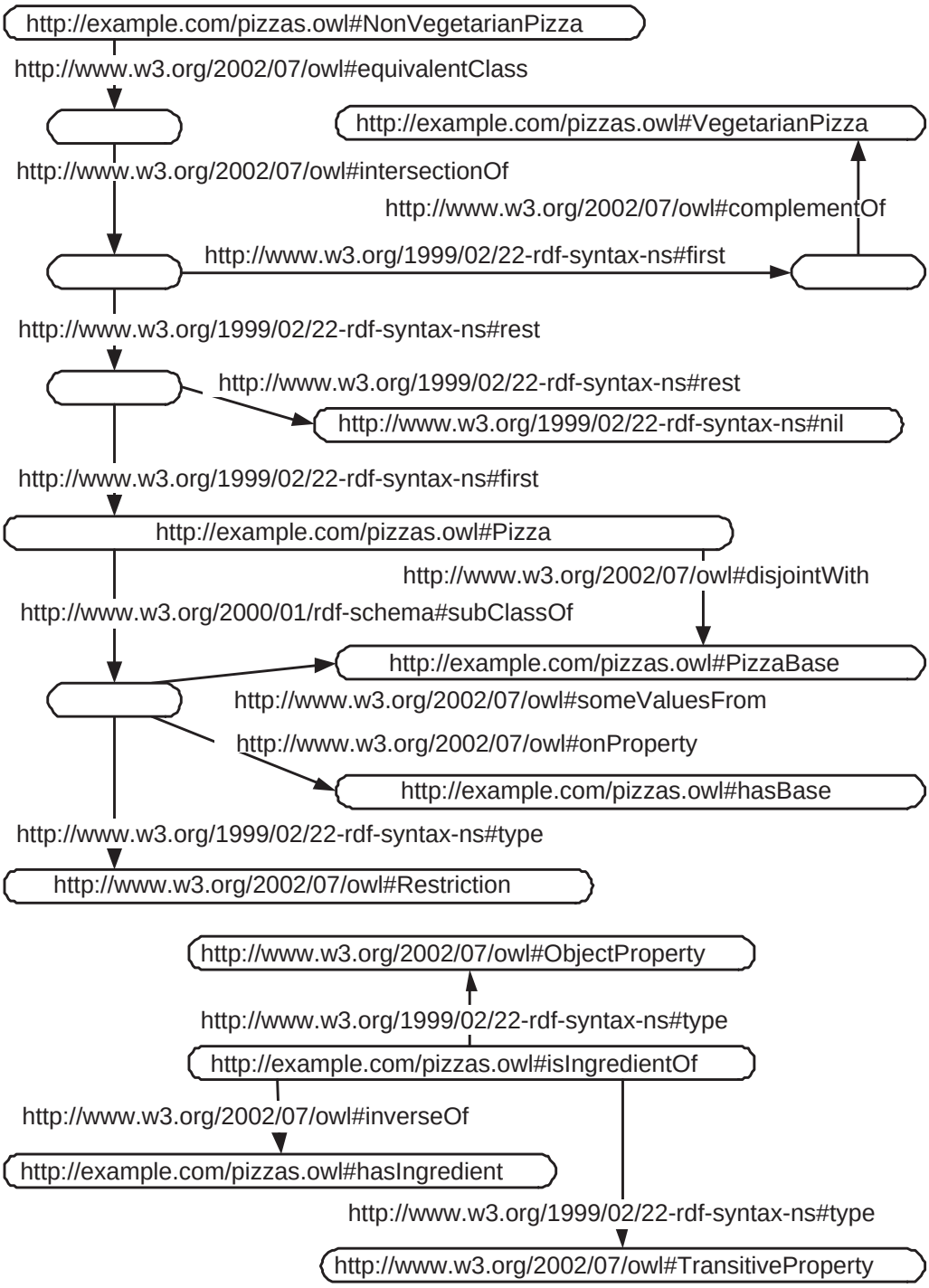

Figure 5.6: Pizza OWL ontology expressed in RDF triples

Chapter 6

Communication in Multi-Agent Systems

Knowledge sharing and exchange is particularly important in multi-agent systems (MAS). An agent is usually described as a persistent entity with some degree of independence or autonomy that carries out some set of operations depending on what he perceives. An agent usually contains some level of intelligence, so it has to have some knowledge about its goals and desires. The whole multi-agent system is created to be capable of reaching goals that are difficult to achieve by an individual agent or a monolithic system. In multi-agent systems, an agent usually cooperates with other agents, so it should have some social and communicative abilities.

In order to communicate, agents must be able to:

- deliver and receive messages — at this physical level, agents must communicate over agreed physical and network layers to be able to deliver and receive strings or objects that represent messages
- parse the messages — at the syntactic level, agents must be able to parse messages to correctly decode the message to its parts, such as message content, language, sender, and also must be able to parse the content of the message
- understand the messages — at the semantic level, the parsed symbols must be understood in the same way, i.e., the ontology describing the symbols must be shared or explicitly expressed and accessible to be able to decode the information contained in the message

For multi-agent systems the first physical level as well as the second syntactic level is well standardized by the Foundation for Intelligent Physical

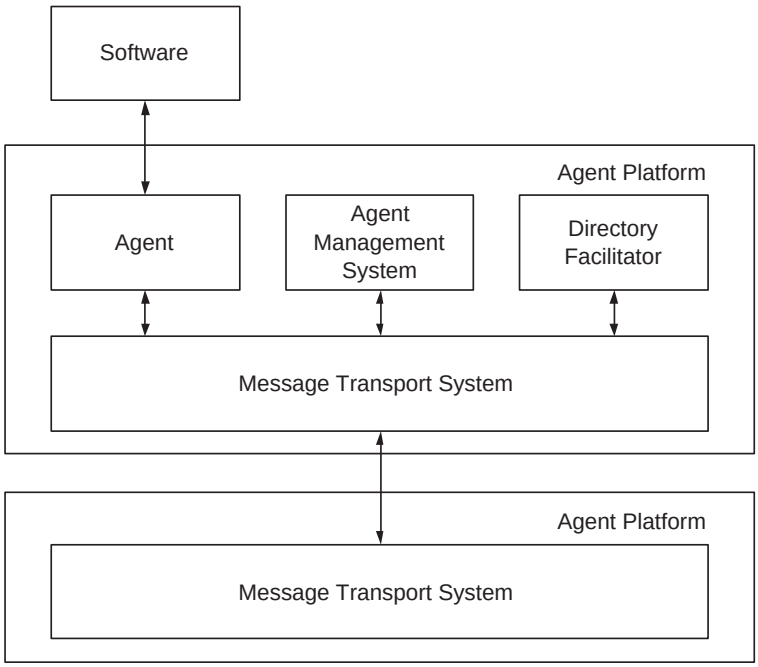

Figure 6.1: FIPA Agent Management Reference Model [46]

Agents (FIPA) [4], for example by agent management specification [46] and agent communication language specification [44]. As for the third level, semantics, standard exists that describe the content languages [45] and that describe usage of ontologies [42].

6.1 FIPA Compliant Multi-Agent Systems

Basic structure of the multi-agent system compliant to FIPA [46] is shown in figure 6.1. An agent is a fundamental actor on an agent platform. Agent platform (AP) provides the physical infrastructure in which agents can be deployed. The AP consists of the machine(s), operating system(s), agent support software, FIPA agent management components and agents. The FIPA agent management components are as follows.

- Agent Management System (AMS) controls access and use of the agent platform and provides services like maintaining a directory of agent names. It provides white page services to other agents. Each agent must be registered with an AMS.

- Message Transport Service (MTS) supports the transportation of FIPA ACL messages between agents on any given AP and between agents on different APs.
- Directory Facilitator (DF) is optional and provides yellow pages services to other agents. Agents may register their services with the DF or query the DF to find out what services are offered by other agents, including the discovery of agents and their offered services in ad hoc networks.

For communication between agents the FIPA Agent Communication Language (ACL) is standardized [44] that is similar to KQML (Knowledge Query and Manipulation Language [9, 41]). An example message in this language follows.

```
(inform
    :sender agent1
    :receiver hpl-auction-server
    :content (price (bid good02) 150)
    :in-reply-to round-4
    :reply-with bid04
    :language sl
    :ontology hpl-auction
)
```

The ACL message contains several parameters. The only mandatory parameter is performative (communicative act — such as `inform` in the previous example), but most ACL messages will also contain `sender`, `receiver` and `content` parameters. As we can see, the ACL message is content independent. The usual language used for expressing content is a LISP-like Semantic Language SL [45] that is based on KIF. This language allows expressing propositions and actions. Also, we can see that the content of the message is ontology independent. The ontology for the content can be specified explicitly in a message parameter.

There are several implementations of the core FIPA standards available. The list and comparison of the Java based frameworks for the development of FIPA compliant multi-agent systems is available in [90]. Based on the comparison, the framework that has many advantages over other ones is the Java Agent DEvelopment Framework (Jade) [5, 22].

6.2 Ontologies in Multi-Agent Systems

Each agent has to know something about a domain he is working in and also has to communicate with other agents. An agent is able to communicate only about facts that can be expressed in some ontology. This ontology must be agreed and understood among the agent community (or at least among its part) in order to enable each agent to understand messages from other agents.

Unfortunately, the "ontology" used for communication between agents is not always expressed explicitly — the constructs used to exchange information may be hardcoded in agents, and no explicit form describing the ontology may be available. The assumptions on the meaning of the vocabulary are implicitly embedded in agents, i.e., in software programs representing agents. In this case it is harder to integrate such agents with other agents that were not programmed to communicate together.

In open multi-agents systems, where agents designed by different programmers or organizations may enter into communication, the ontology must be expressed explicitly in order to enable integration. In such environment, it is also necessary to have standard mechanisms to access and refer to explicitly defined ontologies.

When multiple ontologies are used in a system, translation between ontologies is needed for agents to be able to communicate. For relating ontologies, see the section 4.1.

6.3 FIPA Ontology Service

The FIPA specifications include also Ontology Service Specification [42]. This specification is still in "experimental" state. It proposes a dedicated *ontology agent* (OA) to be available in FIPA agent platform for ontology related services. The role of such an agent is to provide some or all of the following services [42]:

- discovery of public ontologies in order to access them
- help in selecting a shared ontology for communication
- maintain (e.g. register with the DF, upload, download, or modify) a set of public ontologies
- translate expressions between different ontologies and/or different content languages

- respond to queries for relationships between terms or between ontologies
- facilitate the identification of a shared ontology for communication between two agents

It is not mandatory that an ontology agent must provide all of these services, but every OA must be able to participate in a communication about these tasks. Also, it is not mandatory that every agent platform must contain an ontology agent, but when an ontology agent is present, it must be compliant with the FIPA specifications.

Example scenarios [42] of using OA particular services are:

- Querying the OA for definitions of terms
  A user interface agent A wants to receive pictures from picture-archiver agent B to show them to a user. It asks agent B for "citrus". However the agent B discovers that it doesn't have any picture with that description. So it asks the appropriate OA to obtain sub-species of "citrus" within the given ontology. OA answers B that "orange" and "lemon" are sub-species of "citrus", so the agent B can send pictures with these descriptions to agent A and so satisfy his requirements.

- Finding equivalent ontology
  An ontology designer declares the ontology "car-product" to the ontology agent OA2 in U.S. in English terms and translates the same ontology to French for the ontology agent OA1 in France. Agent A2 uses the ontology from OA2 and wants to communicate with agent A1 about cars in ontology maintained by OA2. Because agent A1 doesn't know ontology of agent A2, it queries OA1 for ontology equivalent to that one used by A2 (and maintained by OA2[1]). OA1 returns its French ontology about cars and so A1 can inform A2 that these two ontologies are equivalent and that OA1 can be used as a translator. After that, a dialogue between A1 and A2 can start.

- Translations of terms
  An agent A1 wants to translate a given term from an ontology #1 into the corresponding term in an ontology #2 (for example the concept "the name of a part" can be called *name* in ontology #1 and *nomenclature* in ontology #2). A1 queries DF for an OA which supports the

[1]There is an ontology naming scheme described in [42] that allows to identify these ontologies — for example the English car ontology of agent OA2 can be named as `OA2@http://makers.ford.com/car-product`

translation between these ontologies. DF returns the name of an OA that knows the format of these ontologies (e.g. XML) and has capabilities to make translations between them. A1 can then query this OA and request translation of a term from ontology #1 to ontology #2.

6.3.1 Ontologies

To allow agents to talk about knowledge and about ontologies, for instance to query for the definition of a concept or to define a new concept, a standard meta-ontology and knowledge model is necessary. This meta-ontology and knowledge model must be able to describe the primitives like concepts, attributes or relations. The ontology proposed for FIPA agents (and thus also for the ontology agent) is the OKBC Knowledge Model [33], see the section 3.1.

The ontology `FIPA-meta-ontology` based on OKBC Knowledge Model is defined [42] to describe ontologies. This ontology must be used by an agent when it talks about ontologies. Ontology `FIPA-ontol-service-ontology` must be used when requesting services of an ontology agent. This ontology extends the basic `FIPA-meta-ontology` by symbols enabling manipulation with ontologies. These ontologies are described in a tabular form in the natural language explanation in [42].

In an open environment agents may benefit from knowing the existence of some relationships between ontologies, for example to decide if and how to communicate with other agents. In the agent community, the ontology agent has the most adequate role to know that. It can be then queried for the information about such relationships and it can use that for translation or for facilitating the selection of a shared ontology for agent communication. In the FIPA specification [42] the relations as described in the section 4.2 are proposed. The ontology agent is not required to determine these relationships, but should be able to able to maintain database of these relationships.

6.3.2 FIPA-ontol-service-ontology ontology

The `FIPA-ontol-service-ontology` ontology is an ontology that an ontology agent must understand and that is used to express actions and predicates to be handled by ontology agent. It extends the base `FIPA-meta-ontology` and its main elements are summarized in the table 6.1. An action can be `REQUEST`ed or `CANCEL`ed using FIPA ACL communicative act. For querying, the `QUERY-IF` or `QUERY-REF` communicative acts are used.

Type	Name	Meaning
action	assert	Assert a predicate to ontology specified in the ontology slot of the ACL message. The predicate to be asserted is described using the FIPA-Meta-Ontology.
action	retract	Retract a predicate from ontology (inverse to assert).
action	translate	Translate an expression between ontologies. The expression to be translated is specified together with the source and the requested target ontology.
predicate	ontol-relationship	Expresses a relationship between ontologies. Possible relations are as described in section 4.2.

Table 6.1: Main elements of the `FIPA-ontol-service-ontology`.

6.3.3 Implementation

The FIPA Ontology Service specification is not mandatory to be implemented in a FIPA compliant platform, and is still in the "experimental" state. Nevertheless, two partial implementations are known to us.

The implementation using OKBC knowledge model is described in [89]. The agent serves as an interface to an OKBC server, and supports the assert and retract actions. The translation is not supported.

The ACOS system [59] is not directly implementation of the FIPA proposal, but is an implementation inspired by the ontology service idea. It is a service for agents that provides means for ontology storing. An emphasis is put on consistency checking capability for which a custom algorithm is provided.

Part III

Translations between Ontologies in Multi-Agent Systems

Chapter 7

Semantic Web Technologies and Multi-Agent Systems

From the overview of the state of the art in ontologies and in multi-agent systems (MAS) we can see that the tasks solved in the area of ontologies and particularly in the semantic web are very similar to the tasks that need to be solved in multi-agent systems. In fact, the semantic web depends on distributed knowledge management, which is well modeled using the MAS paradigm. On the other hand, multi-agent systems are starting to need knowledge representation and reasoning on the level offered by the state of the art in the area of ontologies — especially in the open systems, where new agents can join the MAS community even when they were not pre-programmed to work together.

However, the multi-agent systems used today have rarely explicitly and formally defined ontology, especially the systems that do not directly work in the knowledge processing area — such as multi-agent systems supporting manufacturing. The "ontology" is usually defined in the software code only or is defined as a list of symbols with a natural language description.

7.1 RDF and OWL versus SL and OKBC

The FIPA specifications [42] propose to use the KIF-like SL language for expressing content of messages and to use OKBC as a base for all ontologies. However, the mandatory FIPA specifications also leave the freedom to use any language for expressing message content in an ACL message.

The advantage of OKBC is that it is a frame-based language and so the part of OKBC constructs can be mapped to the object oriented languages, so that classes in programming languages can be built on the underlying

ontology and be used for exchanging information. This can be illustrated on the Jade package [5] — the recommended way of handling ontologies is to create Java classes describing concepts and predicates (relations) and then register them as a part of the application ontology. This is a practical, fast way of creating an ontology with an immediate underlying implementation.

However, no inference engine that would enable to use this information for instance for ontology integration is provided due to the problems inherent in OKBC (for example, the maximum cardinality restrictions are inherited, while the minimum cardinality restrictions are not).

All specifications of ontologies[1] in the FIPA proposals are provided in a tabular form. The tables contain a term name with its informal description in a natural language together with fields such as presence, type, values etc., where applicable. A programmatic description of an ontology can be made using the FIPA-Meta-Ontology ontology that provides a set of predicates that enable to state facts about ontology (such as defining classes and slots).

Ontologies in the FIPA proposals and related ontologies for practical applications (see [91] for an example) are motivated mainly by the need to have something that would work immediately, because more attention is paid to the functional behavior of agents. There is nothing wrong with this approach, if we want to have a working solution in a short time where we do not care about possibility of reasoning about ontologies as well as about further interoperability. However, the need for reasoning about ontologies can easily arise for example when requiring interoperability in open multi-agent systems, i.e. systems where new agents with possibly different ontologies can join the community.

On the other hand, in the semantic web area, much more attention is paid to ontologies. The language for expressing knowledge bases is RDF and the language for expressing ontologies is OWL. As we have proposed[2] in [69] and [70], these languages can be used for agent communication and have many advantages when doing so.

The disadvantages of SL and OKBC when compared to RDF and OWL as well as disadvantages of the FIPA ontology service specification [42] are summarized in the table 7.1.

[1] As we have mentioned in the section 2.2.3 and also shown in the figure 3.1, we want an ontology to be "formal explicit specification of conceptualization". This is not entirely satisfied in this case; however the term ontology is used in this way in multi-agent systems. Our goal is to turn the ontologies to be really *formal* and *explicit*.

[2] Originally using DAML+OIL language that is the predecessor of OWL.

SL and OKBC in FIPA	RDF and OWL
It is not possible to specify multiple ontologies in SL language.	RDF uses XML namespaces (and prefixes) to use many ontologies in one graph.
The usage of OKBC for reasoning is problematic and not supported	OWL is created to have reasoning support; reasoners exist.
There are no existing standardized constructs for importing and versioning ontologies and for maintaining information about them.	OWL includes construct for specifying import of ontologies.
Ontologies are defined only informally in FIPA specifications, so no reasoning can be used directly on them (only syntax is defined precisely).	Ontologies expressed in OWL can be used for reasoning; OWL DL reasoners exist.
There are possible values defined for the ontol-relationship predicate but there is no explicitly stated relationship between these values.	In OWL, the properties of predicates can be stated formally and can be used for reasoning.
It is not defined how to express relations between individual ontology elements to specify possible translations	OWL can be used to state the relations between ontology elements, for example using `owl:sameAs`.

Table 7.1: Comparison of SL, OKBC and FIPA specification versus RDF and OWL for the usage in multi-agent systems [73].

7.2 Semantic Web Technologies for Agent Communication

To overcome the problems described above and to use OWL advantages, we have proposed to use semantic web languages for communication between FIPA agents [69, 74]. To use RDF and OWL as a language in FIPA compliant agents, we have first to specify how to express FIPA primitives in OWL. The models of RDF, RDFS, and OWL are represented in the graph of triples of the form *object - property - subject*. According to FIPA specifications [44], a content language (that is used for content of messages exchanged between

agents) has to be able to express objects, propositions, and actions. As we have described in [74, 73, 78], an object can be mapped directly to the RDF resource. A proposition can be mapped to RDF statement. An action does not have its direct counterpart in RDF, however an instance of an action can be described by its act, actor, and parameters. The natural mapping is to model an action by an OWL class, and differentiate between acts by subclassing the action class. Moreover, we get a possibility of reasoning about actions, which is not directly possible with the forms as in the standard FIPA content languages. An actor is typically a FIPA agent. A FIPA agent is identified according to FIPA specification [46] — at least its name is required. An action parameter has a name and a value (object or datatype).

For identifying ontologies, OWL uses URIs, which can be adopted directly. The proposal described above can be specified in a general ontology that would be shared between all agents like the SL semantics should be shared between FIPA agents that use SL. We distinguish between three basic ontologies corresponding to the description in the FIPA specification [42]:

- OWL (meta)ontology itself — plays a similar role as the FIPA-Meta-Ontology, and allows to describe ontologies and their parts, together with providing means of describing objects and various statements
- `agent` ontology — uses the OWL ontology and defines basic constructs used in the agent communication, such as agents and actions; it is based on the description provided in [46]
- `ontoagent` ontology — extends the `agent` ontology as needed for the ontology agent and plays a similar role as the FIPA-Ontol-Service-Ontology ontology; it includes ontology agent actions together with their arguments, and also relationships between ontologies (see section 7.3 for details)

7.2.1 Agent Ontology

Let us describe the `agent` ontology. We have chosen `http://fipa.org/ns/agent.owl` as the namespace of this ontology. The primary classes[3] of this ontology are Action and Agent.

An Agent is uniquely identified by its AgentID. One of the reasons for creating a special class AgentID instead of using plain string name is that the OWL DL does not permit datatype properties to be inverse functional properties (i.e., of `rdf:type owl:InverseFunctionalProperty`) to specify the

[3]Let us recall from the section 5.3 that a class in OWL is a concept in DL and that a property in OWL is a role in DL.

uniqueness. Another reason is that FIPA agent may have multiple properties assigned for its identification [46] — name (unique symbolic name), address (transport addresses) and resolver (addresses of name resolution services). The definition for Agent and AgentID is then as follows (for brevity and clarity, defined inverse relations and the usual disjoint axioms[4] are omitted):

$$
\begin{aligned}
\textsc{Agent} &\equiv \exists \mathsf{hasAgentID}.\textsc{AgentID} \\
\textsc{AgentID} &\sqsubseteq \top \\
\textsc{AgentID} &\sqsubseteq = 1\ \mathsf{hasName}.\top \\
\top &\sqsubseteq \forall \mathsf{hasAgentID}.\textsc{AgentID} \\
\top &\sqsubseteq \forall \mathsf{hasAgentID}^{-1}.\textsc{Agent} \\
\top &\sqsubseteq \leq 1\ \mathsf{hasAgentID}.\top \\
\top &\sqsubseteq \leq 1\ \mathsf{hasAgentID}^{-1}.\top \\
\top &\sqsubseteq \forall \mathsf{hasResolver}^{-1}.\textsc{AgentID} \\
\top &\sqsubseteq \forall \mathsf{hasAddress}^{-1}.\textsc{AgentID} \\
\top &\sqsubseteq \leq 1\mathsf{hasName}.\top \\
\top &\sqsubseteq \forall \mathsf{hasName.xsd{:}string} \\
\top &\sqsubseteq \forall \mathsf{hasAddress.xsd{:}string} \\
\top &\sqsubseteq \forall \mathsf{hasResolver.xsd{:}string}
\end{aligned}
$$

An Action has its actor, which is an Agent, and may have multiple action parameters. Each of the action parameters has to have a name and may specify object or datatype parameter. The definition is again quite simple:

$$
\begin{aligned}
\textsc{Action} &\sqsubseteq \top \\
\textsc{Action} &\sqsubseteq \forall \mathsf{hasParameter}.\textsc{ActionParameter} \\
\textsc{ActionParameter} &\sqsubseteq \top \\
\textsc{ActionParameter} &\sqsubseteq = 1\ \mathsf{hasName}.\top \\
\top &\sqsubseteq \forall \mathsf{hasParameter}^{-1}.\textsc{Action} \\
\top &\sqsubseteq \forall \mathsf{hasParameter}.\textsc{ActionParameter} \\
\top &\sqsubseteq \forall \mathsf{hasActor}^{-1}.\textsc{Action} \\
\top &\sqsubseteq \forall \mathsf{hasActor}.\textsc{Agent} \\
\top &\sqsubseteq \forall \mathsf{hasObjectValue}^{-1}.\textsc{ActionParameter} \\
\top &\sqsubseteq \forall \mathsf{hasDatatypeValue}^{-1}.\textsc{ActionParameter}
\end{aligned}
$$

[4] In OWL, the unique name assumption and does not hold, so together with open world assumption this means that the fact that two classes are disjoint has to be stated explicitly for all such classes — for example, $\bot \sqsubseteq \textsc{Agent} \sqcap \textsc{Action}$. Similar disjointness must be explicitly stated for individuals as well.

7.2.2 Implementation

For implementation of FIPA compliant agents that use the `agent` ontology described above, we have used the Jade package [5] for agents together with the Jena framework [16] for semantic web technologies. Agents use Jena API to serialize and deserialize RDF/OWL. Any serialization language supported by Jena can be used, i.e., RDF/XML, TRIPLE or N3/TURTLE, and the language is specified in the language parameter of the ACL message. An architecture is illustrated on the ontology agent example in the figure 7.1.

Thanks to the RDF features the ontology used in the message is expressed directly in the message content (all imported namespaces are specified), so the ontology parameter is in fact redundant in the ACL message. Thus from our point of view, it is not mandatory to fill the parameter in order to communicate using multiple ontologies. This is compatible with the FIPA ACL specification — if the parameter is used, it has to specify ontology, but the parameter does not have to be used.

Example messages that use this ontology in their content are shown in a sample ontology agent scenario in the section 7.3.2.

7.3 Ontology Agent Based on OWL

Using the RDF for content of messages and OWL for expressing ontologies, we have proposed and implemented [74, 73, 78][5] the FIPA Ontology Service specification.

The architecture of the ontology agent based on OWL is shown in the figure 7.1. The agent uses Jena API to access and process ontologies. For reasoning, primarily the Pellet reasoner [6] is used.

7.3.1 OntoAgent Ontology

Let us discuss mapping of the FIPA-Ontol-Service-Ontology to RDF/OWL. We have created new `ontoagent` ontology with the namespace `http://fipa.org/ns/ontoagent.owl`. The ontology describes the vocabulary from the table 6.1 and adds other constructs. First, the class of ontology agents is defined.

$$\textsc{OntologyAgent} \sqsubseteq \textsc{Agent}$$

For communication about assertions and retractions with ontology agent,

[5] The original implementation was using older RDF query language RDQL; also ontologies and reasoning support were improved in the version described in this thesis.

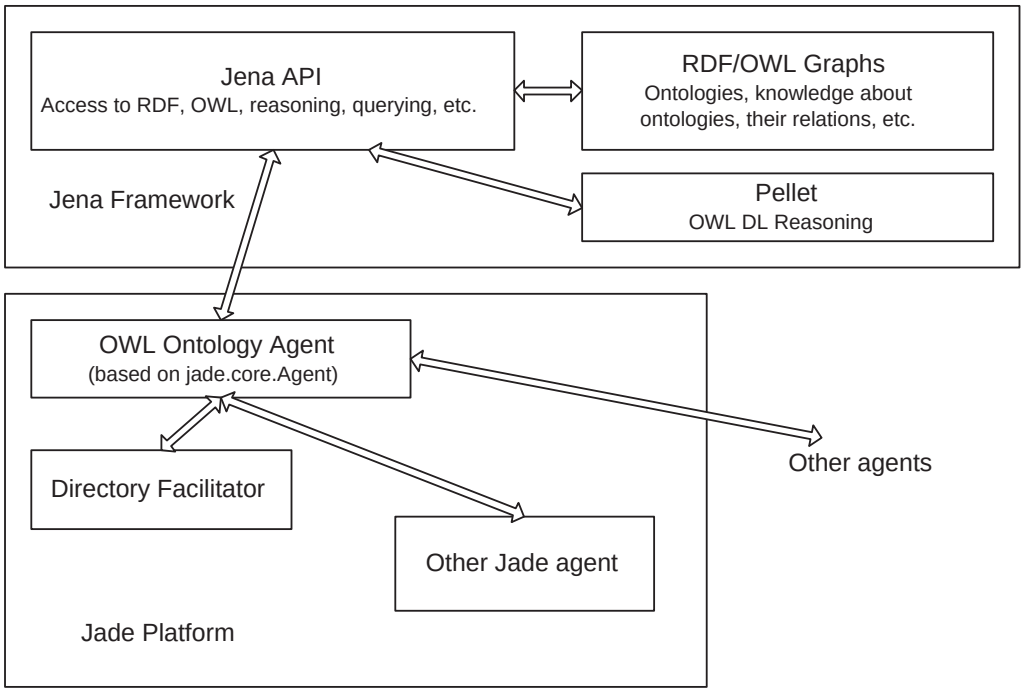

Figure 7.1: Architecture of the FIPA ontology agent implemented in Jade and using Jena API to access OWL ontologies.

we need to express a set of RDF triples (RDF graph) that describe the facts to be added. Note that for example OWL class restrictions are formed by multiple triples that are related together and that may not make any sense alone. The RDF graph (rather than individual triple) needs to be specified as a parameter of an action, such as assert action. It has to be specified as an action parameter in a message to ontology agent.

The possibilities include using a string with graph serialization, usage of reification, and named graphs. The first possibility means that a string would be related to the action instance via datatype property. This string would contain the serialization of the RDF graph, such as in RDF/XML. An example of a string specification follows (the example shows the content of a message requesting to assert that `:john :has :cat`):

```
[] a ontoagent:Assert;
   agent:hasParameter
     [ a ontoagent:StringGraphParam;
       agent:hasDatatypeValue """
           @prefix : <http://example.org/ex.owl#> .
```

```
 :john :has :cat .
""" ].
```

The second possibility is to use RDF reification (see section 5.1). In this way, we can use a statement as an action parameter. Multiple triples can be specified as multiple parameters. However, there is no straightforward way to specify blank nodes that would be used in multiple triples — such as for describing constraints in OWL. Naming blank nodes could be an option, but one would have to be careful to use unique names across the messages to ontology agent, and also for RDF the RDF graph equality as defined for blank nodes would be lost (this may be no problem for OWL constructs). A way to reify blank nodes described in [23] is to make them named resources of a special type (`rei:BNode`[6]) and assume that during dereification the receiver will treat them as anonymous nodes. Unfortunately, the semantics of reification has some problems, for example that it may entail statements that were not intended to be stated [31]. An example of specifying a triple (without blank node) via reification follows:

```
[] a ontoagent:Assert;
   agent:hasParameter
     [ a ontoagent:StatementParam;
       agent:hasObjectValue
           [ a rdf:Statement;
               rdf:subject :john;
               rdf:predicate :has;
               rdf:object :cat ]  ].
```

The last possibility that we consider is to use RDF named graphs [31, 32]. This (non-standardized) RDF extension allows specifying RDF graphs as resources. An additional parameter, a name of the graph, is added to each triple. To enforce the RDF blank node scoping rules blank nodes cannot be shared between different named graphs. However, they can be used without any limitation in one graph — this is the clear advantage over the standard RDF reification. Also, the semantics of named graphs is clearly defined [31], and SPARQL extension for named graphs is available as well. An example that uses named[7] graph extension follows:

[6] Where `rei` namespace is defined at `http://www.w3.org/2004/06/rei` and declares additional (not standardized) reification vocabulary, including `BNode` for reified blank node specification.

[7] Although in this example the graph does not have any name.

```
[] a ontoagent:Assert;
   agent:hasParameter
     [ a ontoagent:NamedGraphParam;
       agent:hasObjectValue
           { :john :has :cat } ].
```

All of these approaches can be used. The first approach, string with serialized graph, does not bring the RDF message to ontology agent out of OWL DL, since the parameter is not part of the message RDF triples (it is hidden in a string). This approach is the easiest one for implementation. The second approach, reification, brings the message to OWL Full, since it is not possible to make statements about statements in OWL DL. The triples are parsed at once within the message, but additional processing is required for dereification. The disadvantage is that processing of blank nodes is not straightforward and requires at least one class with a special semantics that must be taken into account during processing. The last approach, named graphs, is using non standard RDF extension, and is also out the scope of OWL DL, but otherwise it has only advantages: everything is parsed at once, processing of blank nodes is straightforward, and semantics is defined within the RDF graph.

The issue of getting out of the OWL DL matters whenever we want to make any reasoning over messages. If the processing is only to read required action and its parameter from RDF triples then getting out of OWL DL is not any problem.

Let us note that if we relax the requirement to embed all the information in the message, it would be possible to specify the triples to be added as a RDF graphs accessible via URI, for example from web or from network file system. Then the action parameter would be named resource, and by dereferencing URI the RDF document would be obtained.

An ontology for all the options above and the actions in the table 6.1 is as follows. Note that in addition to retract, assert and translate actions as defined by FIPA [42], actions to add and delete ontology are added. The parameters of these actions are defined as described above — there is no limitation on what approach is used.

$$
\begin{aligned}
\textsc{OntoManipAction} &\equiv \textsc{Action} \sqcap \\
&\quad\ \exists \mathsf{hasParameter}.\textsc{OntoNameParam} \\
\textsc{Create} &\sqsubseteq \textsc{OntoManipAction} \\
\textsc{Delete} &\sqsubseteq \textsc{OntoManipAction} \\
\textsc{Assert} &\sqsubseteq \textsc{OntoManipAction} \\
\textsc{Retract} &\sqsubseteq \textsc{OntoManipAction}
\end{aligned}
$$

$$
\begin{aligned}
\textsc{Translate} \equiv\ & \textsc{Action} \sqcap \\
& \exists \mathsf{hasParameter}.\textsc{OntoNameFromParam} \\
& \sqcap \exists \mathsf{hasParameter}.\textsc{OntoNameToParam} \\
\textsc{NamedGraphParam} \equiv\ & \exists \mathsf{hasName}.\{\text{"namedGraph"}\} \sqcap \\
& \textsc{ActionParameter} \sqcap \\
& \exists \mathsf{hasObjectValue}.\textsc{rdfg:Graph} \sqcap \\
& = 1\ \mathsf{hasObjectValue}.\top \\
\textsc{StatementParam} \equiv\ & \exists \mathsf{hasName}.\{\text{"statement"}\} \sqcap \\
& \textsc{ActionParameter} \sqcap \\
& \exists \mathsf{hasObjectValue}.\textsc{rdf:Statement} \sqcap \\
& = 1\ \mathsf{hasObjectValue}.\top \\
\textsc{StringGraphParam} \equiv\ & \exists \mathsf{hasName}.\{\text{"stringGraph"}\} \sqcap \\
& \textsc{ActionParameter} \sqcap \\
& = 1\ \mathsf{hasDatatypeValue}.\top \\
\textsc{OntoNameParam} \equiv\ & \exists \mathsf{hasName}.\{\text{"ontologyName"}\} \sqcap \\
& \textsc{ActionParameter} \sqcap \\
& = 1\ \mathsf{hasDatatypeValue}.\top \\
\textsc{OntoNameFromParam} \equiv\ & \exists \mathsf{hasName}.\{\text{"ontologyNameFrom"}\} \sqcap \\
& \textsc{ActionParameter} \sqcap \\
& = 1\ \mathsf{hasDatatypeValue}.\top \\
\textsc{OntoNameToParam} \equiv\ & \exists \mathsf{hasName}.\{\text{"ontologyNameTo"}\} \sqcap \\
& \textsc{ActionParameter} \sqcap \\
& = 1\ \mathsf{hasDatatypeValue}.\top
\end{aligned}
$$

Another part of the ontology is formed by the relationships between ontologies as defined in [42] and described in section 4.2. The relationships between ontologies are in addition to FIPA proposal ordered by property subsumption, so when relationship between two ontologies is known, the ontology agent is able to use reasoning to derive whether other relationships hold or not. The ontology relationship properties are subproperties of `owl:OntologyProperty` that has domain and range `owl:Ontology` — first two axioms are copied from OWL to express this fact.

$$
\begin{aligned}
\top &\sqsubseteq \forall \mathsf{owl{:}OntologyProperty}^{-1}.\textsc{owl:Ontology} \\
\top &\sqsubseteq \forall \mathsf{owl{:}OntologyProperty}.\textsc{owl:Ontology} \\
\mathsf{approximatelyTranslatableTo} &\sqsubseteq \mathsf{owl{:}OntologyProperty} \\
\textsc{Trans}(&\mathsf{approximatelyTranslatableTo}) \\
\mathsf{weaklyTranslatableTo} &\sqsubseteq \mathsf{approximatelyTranslatableTo} \\
\mathsf{stronglyTranslatableTo} &\sqsubseteq \mathsf{weaklyTranslatableTo} \\
\mathsf{isExtendedBy} &\sqsubseteq \mathsf{stronglyTranslatableTo} \\
\mathsf{approximatelyTranslatableTo} &\equiv \mathsf{approximatelyTranslatableFrom}^{-1}
\end{aligned}
$$

$$\begin{aligned}
\mathsf{stronglyTranslatableFrom} &\equiv \mathsf{stronglyTranslatableTo}^{-1} \\
\mathsf{weaklyTranslatableFrom} &\equiv \mathsf{weaklyTranslatableTo}^{-1} \\
\mathsf{extends} &\equiv \mathsf{isExtendedBy}^{-1} \\
\mathsf{equivalentTo} &\sqsubseteq \mathsf{owl{:}OntologyProperty} \\
\textsc{Trans}(\mathsf{equivalentTo}) & \\
\mathsf{equivalentTo} &\equiv \mathsf{equivalentTo}^{-1} \\
\mathsf{identicalTo} &\sqsubseteq \mathsf{equivalentTo} \\
\mathsf{identicalTo} &\equiv \mathsf{identicalTo}^{-1}
\end{aligned}$$

7.3.2 Usage of OntoAgent Ontology

The actions described in the OntoAgent ontology are requested using communicative act `REQUEST`. The action is then described by its type, actor and additional parameters.

Let us show an example of a message[8] that requests asserting to an ontology. The message communicative act is `REQUEST` and the message content is describing an action `Assert` that has the ontology agent `OA@MO` as an actor. The parameter of the message is a graph expressed in a string. The request is to add triple `vegetable:Lemon rdfs:subClassOf vegetable:Citrus` to the ontology `http://vegetable.org/onto.owl`.

```
(REQUEST
 :sender  ( agent-identifier  :name PAA@MO:1099/JADE)
 :receiver  (set ( agent-identifier :name OA@MO:1099/JADE))
 :content  "
    @prefix agent: <http://fipa.org/ns/agent.owl#> .
    @prefix ontoagent: <http://fipa.org/ns/ontoagent.owl#> .
    [] a ontoagent:Assert;
    agent:hasActor
      [ a ontoagent:OntologyAgent;
        agent:hasAgentID
          [ a agent:AgentID;
            agent:hasName \"OA@MO\" ] ];
    agent:hasParameter
    [ a ontoagent:OntoNameParam;
      agent:hasDatatypeValue
                  "http://vegetable.org/onto.owl#" ];
    agent:hasParameter
```

[8] All the message examples in this text are shortened and formatted for brevity and clarity.

```
    [ a ontoagent:StringGraphParam;
      agent:hasDatatypeValue \"\"\"
       @prefix vegetable: <http://vegetable.org/onto.owl#> .
       @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
       vegetable:Lemon rdfs:subClassOf vegetable:Citrus.
      \"\"\" ]. "
 :reply-with  PAA@MO:1099/JADE1172476449895
 :language N3
)
```

For queries, SPARQL language [82] is used (see section 5.4). For SELECT, CONSTRUCT and DESCRIBE queries, the `QUERY-REF` communicative act is used, for ASK queries, the `QUERY-IF` communicative act is used.

7.3.3 Example Scenario

The described ontology agent is suitable to be used in some of the scenarios described in the FIPA ontology proposal. An example of a scenario from [42] follows (together with sample ontologies that were created to support the scenario).

A picture-archive agent is asked to provide pictures of "citrus". However, he does not find any picture with this description in his database. So, he asks ontology agent to provide classes that would be equivalent to or subclasses of "citrus" (since he knows that such descriptions can be used as well):

```
(QUERY-REF
 :sender  ( agent-identifier  :name PAA@MO:1099/JADE)
 :receiver  (set ( agent-identifier :name OA@MO:1099/JADE))
 :content  "
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    PREFIX vegetable: <http://vegetable.org/onto.owl#>
    SELECT ?subSpecCitrus
    WHERE { ?subSpecCitrus rdfs:subClassOf vegetable:Citrus }"
 :reply-with  PAA@MO:1099/JADE1172476512556
 :language SPARQL-SELECT
)
```

Note that using namespaces directly solves the problem of referring multiple ontologies in one message, which is not possible in the FIPA SL language. The ontology parameter of ACL message is not used, since it would be redundant information here. After this message, the ontology agent replies that "orange" and "lemon" are subclasses of "citrus":

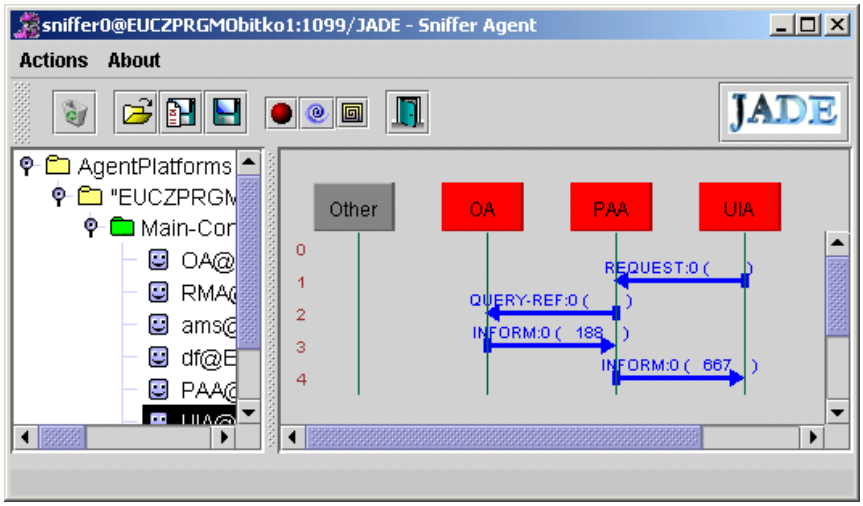

Figure 7.2: A snapshot of the JADE sniffer agent: message flow between agents in the sample FIPA scenario.

```
(INFORM
 :sender  ( agent-identifier  :name OA@MO:1099/JADE)
 :receiver  (set ( agent-identifier :name PAA@MO:1099/JADE)
 :content  "
   @prefix resultSet:
    <http://www.w3.org/2001/sw/DataAccess/tests/result-set#> .
   [] a <resultSet:ResultSet> ;
      <resultSet:resultVariable> \"subSpecCitrus\" ;
      <resultSet:solution>
        [ <resultSet:binding>
            [ <resultSet:value>
                <http://vegetable.org/onto.owl#Orange> ;
              <resultSet:variable>
                \"subSpecCitrus\"
            ]
        ] ;
      <resultSet:solution>
        [ <resultSet:binding>
            [ <resultSet:value>
                <http://vegetable.org/onto.owl#Lemon> ;
              <resultSet:variable>
                \"subSpecCitrus\"
```

```
            ]
    ] . "
 :in-reply-to  PAA@MO:1099/JADE1172476512556
 :language  N3
)
```

The picture-archive agent finds pictures with this description, so he can provide these pictures as pictures with "citrus". The flow of messages is illustrated in the figure 7.2.

The RDF encoded result returned from SPAQRL SELECT query may be too complicated for easy questions. An appropriate CONSTRUCT query may lead to straightforward answer. For the example above, the query message would be as follows.

```
(QUERY-REF
 :sender  ( agent-identifier  :name PAA@MO:1099/JADE)
 :receiver  (set ( agent-identifier :name OA@MO:1099/JADE))
 :content  "
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    PREFIX vegetable: <http://vegetable.org/onto.owl#>
    CONSTRUCT { ?subSpecCitrus rdfs:subClassOf
                vegetable:Citrus }
    WHERE { ?subSpecCitrus rdfs:subClassOf
                vegetable:Citrus } "
 :reply-with  PAA@MO:1099/JADE1172476192753
 :language SPARQL-CONSTRUCT
)
```

The answer from the ontology agent is then as follows.

```
(INFORM
 :sender  ( agent-identifier  :name OA@MO:1099/JADE)
 :receiver  (set ( agent-identifier :name PAA@MO:1099/JADE)
 :content  "
    @prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
    @prefix vegetable:  <http://vegetable.org/onto.owl#> .
    vegetable:Lemon rdfs:subClassOf vegetable:Citrus .
    vegetable:Orange rdfs:subClassOf vegetable:Citrus . "
 :in-reply-to  PAA@MO:1099/JADE1172476192753
 :language  N3
)
```

7.4 Adding OWL Semantics to XML

The popularity of XML as a syntax for exchanging information in structured documents lead to the usage of XML for exchanging messages between agents [91, 92]. In the multi-agent systems implementation, XML is used since unlike SL language it is supported directly by other systems that the multi-agent systems encapsulate.

XML is a standard for encoding documents, while RDF is a standard for encoding data that can be distributed among multiple documents. XML serves for marking up text with arbitrary tags, while RDF serves for data representation with clear semantics. For this reason, there is no standardized way of mapping from XML documents to RDF. The RDF representation in XML, known as RDF/XML means that there is a special XML form in which RDF can be serialized, but this does not help with the general mapping — it only means that some special XML documents can be almost directly read as RDF data.

When integrating existing multi-agent systems that use XML syntax for communication, and also when working with any systems that use XML, it would be desirable to have XML to RDF/OWL conversion. We have summarized some of the possibilities in [71] — they include giving special meaning to XML, using fixed translation algorithm, or using only special form of XML that would be compatible with RDF.

Our alternative approach to these proposals is to provide semantics in a form of OWL constructs and define the semantics by providing a mapping from XML to OWL [69, 70].

Instead of using proprietary extension or instead of a fixed translation, more alternatives of translation are offered as well as alternatives for possible restrictions for the ontology. However, a human assistance is required to select the alternative of intended semantics of given XML constructs. Only XML documents or messages are required, no other definition is necessary.

The data flow diagram for is shown in Figure 7.3. The XML document or a set of XML documents is loaded by the mapping definition wizard. The wizard offers possible translation and possible restrictions in several steps, and user can select the desired output. The result is OWL ontology and mapping definition. Using this information, messages can be translated between OWL and XML for communication between agents. Also, the OWL ontology can be used for reasoning over the content of converted XML messages.

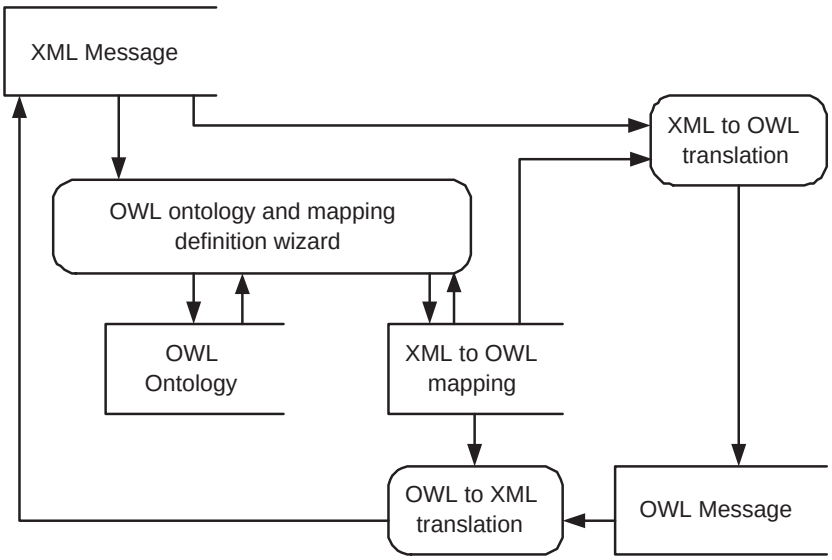

Figure 7.3: Data flow diagram of translating between XML and OWL.

7.4.1 Basic Heuristics for Translation

There are several heuristics that can be used for determining the mapping and the target ontology. The XML elements, attributes and texts included in the elements must be handled. Texts can be handled as special attributes of XML elements.

The default mapping heuristics as offered by the wizard is as follows:

- XML tag — it is assumed that a tag represent an instance of a class, and so is turned to be a class (i.e., subclass of `owl:Thing`)
- XML attribute — it is assumed that the value of the attribute represents a literal value and is linked with the tag (represented by OWL individual of class as defined for the tag) via `owl:DatatypeProperty`; the property name is formed as the attribute name with "has" prefix
- text within XML tag — it is assumed that this is a text linked in the similar way as XML attribute, i.e. a literal text value connected via `owl:DatatypeProperty` named `hasText` (the name is only suggested and can be changed if it would clash with a XML attribute of name "text")
- nested XML tag — it is assumed that the tags (represented by classes) are linked via owl:ObjectProperty; the property name is the nested

XML tag name with "contains" prefix (the prefix is different from datatype properties, so that we avoid the possibility of having one property both object and datatype property — which would get us out of OWL DL to OWL Full)

In addition, the domains and ranges of both datatype and object properties are computed and may be included to the ontology. They are computed from all occurrences in the XML documents, so for correct computation, it is required that appropriate XML documents are processed. The domain is set to be the union of all tags (classes) that appeared as the subject of the property. The range for object properties is the class that they connect (see above), the range for datatype properties is guessed from the format (e.g., number, boolean, date/time, or general string).

Let us illustrate the default mapping with computation of the domains and ranges on an example. Sample XML message content from the transportation domain [91] (with the semantics defined in tabular form with natural language description, like in the FIPA specifications [42]) is as follows:

```
<component type="ConveyorBelt" name="b1">
   <connection from="w1" to="d1" defaultCost="12.5">
</component>
```

This file is an input for the ontology and mapping definition wizard. The wizard parses the document and suggests how to construct the OWL ontology from it and how to translate from the XML document to the OWL content.

We have implemented the proof-of-concept framework as a standalone application that uses Xerces2 XML parser[9] to process XML and that uses Jena framework for processing RDF/OWL information, see figure 7.4 for a screenshot.

With the settings conforming to the settings above without any additional settings and modifications, the following ontology and translation of the message is constructed. The first letters of XML tags and attributes were capitalized by the wizard to get better readability of OWL classes and properties.

```
# ontology
:Component a owl:Class .
:Connection a owl:Class .
:hasType a owl:DatatypeProperty ;
      rdfs:domain :Component.
```

[9] http://xml.apache.org/xerces2-j/

```
:hasName a owl:DatatypeProperty ;
      rdfs:domain :Component.
:containsConnection a owl:ObjectProperty ;
      rdfs:domain :Component ; rdfs:range :Connection.
:hasFrom a owl:DatatypeProperty ;
      rdfs:domain :Connection.
:hasTo a owl:DatatypeProperty ;
      rdfs:domain :Connection.
:hasDefaultCost a owl:DatatypeProperty ;
      rdfs:domain :Connection.

# message (xml document)
[ a :Component;
  :hasType "ConveyorBelt"; :hasName "b1";
  :containsConnection
    [ a :Connection; :hasFrom "w1"; :hasTo "d1";
      :hasDefaultCost "12.5" ] ].
```

During the process, both ontology and translation of the XML message are generated. Other XML messages can be read and converted to conform to the generated ontology automatically.

The ontology above is the simplest one generated by the wizard. The names of all classes and properties can be renamed, and domains and ranges can be changed or not set at all. In addition, disjoint axioms (stating disjointness of all generated classes) can be generated automatically (see the figure 7.4 for initial selection).

7.4.2 Additional Possibilities

In addition to the basic heuristics for mapping, the following options can be considered.

It may be helpful to express the relation of containing (nesting) elements directly in the ontology. This would simplify queries involving deeper nesting and would also simplify further refinement of ontology with regard to this relation. This relation can be added in a way that all object properties are subproperties of one transitive property. The use of this feature is shown later in the chapter 9.

Since XML is originally text markup language, text may appear between tags. This is usually not the case of messages between agents, where the text usually appears only in the leaf tags; however, it is possible in a XML document. In this case the text may be divided by other tags and it may

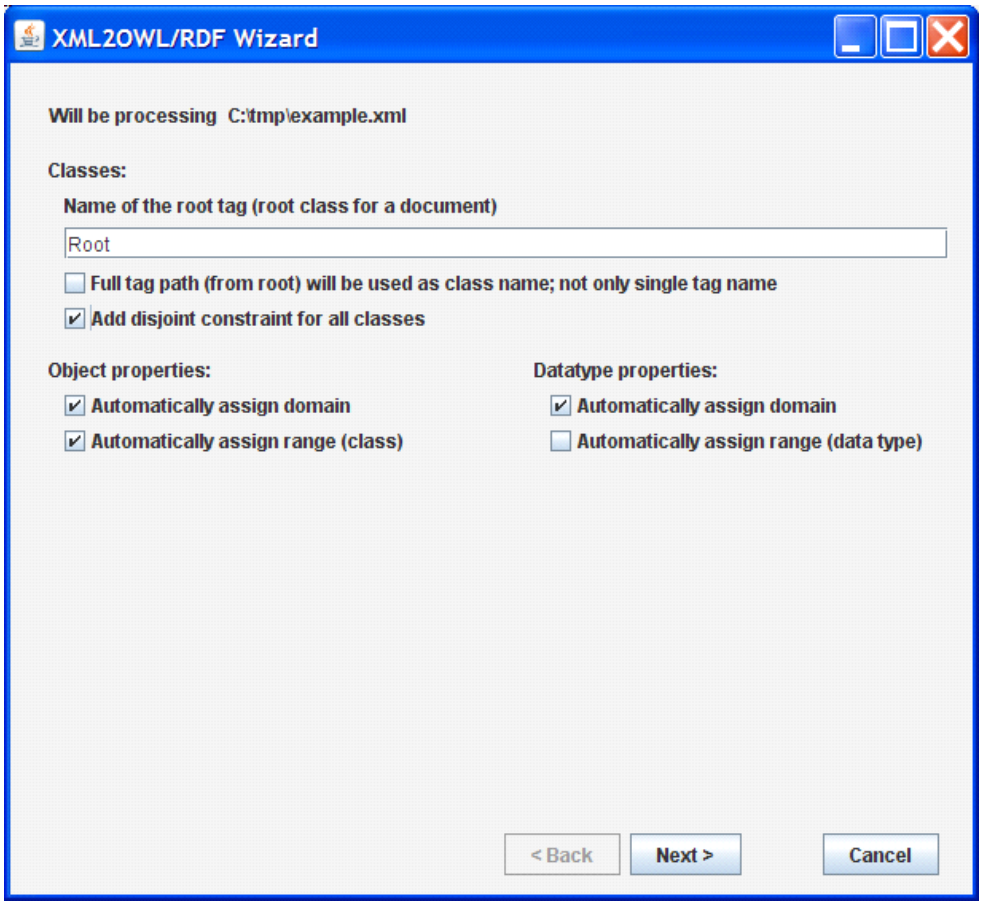

Figure 7.4: XML2OWL/RDF Wizard asking for first properties of the conversion.

be needed to split it into parts for OWL representation (by default, all texts within a tag would be stored separately as texts related using one multiply added property, so the order is not preserved). Multiple properties can be used to relate the text, but in addition, it is needed to number the values — in XML document, the order is important, while in RDF graph, the order is neither preserved nor important and has to be specified explicitly if needed. RDF lists can be used for this ordering. A similar situation is when multiple tags with the same name appear within another tag. Again, explicit numbering may be needed for OWL representation.

It may be desirable to extend the mapping possibilities in addition to the above. For example, another possibility of mapping is to map an attribute to a class — i.e., to an object property relating the class with another datatype

property that holds the values. Also, a pair of an XML element and its attribute with particular value can be mapped to a single OWL class [71]. This is useful when the attribute indicates further specialization of the element in the intended model.

For example, for the XML message above, we may want to create a class `ConveyorBelt` from the pair `<component type="ConveyorBelt" />`. This could be made in the mapping wizard; however, it is also possible to express this fact in OWL language as follows.

```
:ConveyorBelt a owl:Class;
   owl:intersectionOf
     ( :Component
       [ a owl:Restriction;
         owl:onProperty :type;
         owl:hasValue "ConveyorBelt"] ).
```

In the description logic notation the equivalent information is expressed as

$$\textsc{ConveyorBelt} \equiv \exists \mathsf{type}.\{\text{"}\mathsf{ConveyorBelt}\text{"}\} \sqcap \textsc{Component}$$

Then, if an individual *myIndividual* of the type Component exists that has the value of the type property set as "ConveyorBelt"

$$\textsc{Component}(\mathit{myIndividual})$$
$$\mathsf{type}(\mathit{myIndividual}, \text{"}\mathsf{ConveyorBelt}\text{"})$$

it is classified[10] under the ConveyorBelt class.

This further enhancement of the ontology seems to be better for us, since it does not require special XML to OWL mappings that are hard to maintain. Our recommendation from our experience (see the chapter 9) is to use basic transformation to OWL and then enrich the semantics in the OWL form. The disadvantage of this approach is that reasoning must be used for such enrichment. Also, not all enrichments are easily expressible in OWL — it is possible that rules may have to be used.

Of course, the ontology generated by the wizard can be updated as needed. For further meaningful automatic conversion between XML and OWL it is only necessary to preserve the generated classes and properties, since they are used to describe OWL individuals as expressed in particular

[10]Unfortunately, for such a reasoning, the DIG 1.1 interface cannot be used, since it does not support datatype constraints. For the tools that use DIG interface for communication with reasoner, this conclusion would not be found. Embedded reasoner handles this case without problems.

XML documents. For the example of processing larger set of larger documents using the wizard, see the chapter 9.

Chapter 8

Translation between Ontologies

As we have already said in the introduction, it is unlikely that there would ever be one ontology for even a simple domain that would satisfy everybody and forever [79]. Because of that the problem of translation between ontologies is an important issue for integration of multiple systems. In multi-agent systems, the problem can be described as follows: two (or more) agents want to communicate together about facts in a shared domain of interest, but they use different ontologies of the domain. The question is how they can communicate even when they express the same or similar information in a different way, using different ontologies.

Like in the world of natural language, where the translation of utterances between languages is used, the translation of messages between ontologies can be used to enable agent communication. In this chapter we discuss how to proceed with such a translation to facilitate communication between agents that use different ontologies. The translation can be then used for the integration of multi-agent systems that come for example from different companies.

We deal with translation between different ontologies that are all expressed in the same ontology language — OWL. The task of translation between different formalisms for expressing ontologies is discussed for example in [18].

8.1 Examples of Transportation Domain Ontologies

To illustrate our approach to the integration of multi-agent systems, we have chosen an important part of the domain of manufacturing or virtual enterprises — transportation. In this section, we will briefly describe selected

ontologies that describe the transportation domain [75].

8.1.1 Berlin Transportation Ontology

The Berlin Transportation ontology [95] has a goal to describe Berlin local transport service, but defines general concepts and relations for general transportation description (see figure 8.1[1]). This ontology was created within the Agentcities project.

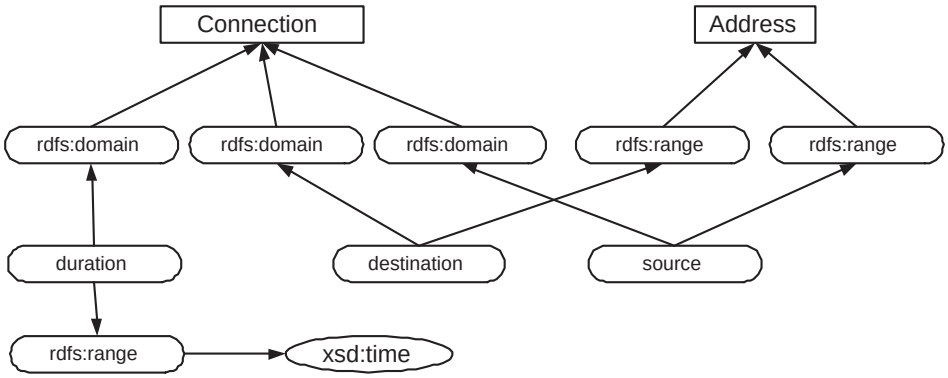

Figure 8.1: Berlin transportation ontology (upper part with main concepts).

This ontology is expressed in DAML+OIL[2]. The main classes are "Connection" and "Address", that can be connected via relations "destination" and "source". Based on these main entities, the ontology then describes in more details city means of transport, such as buses or taxis.

We don't know whether this ontology was actually used, but we note that without our modifications, Jena was not able to parse the original ontology. Also, there is one data-type property that has range defined as a subclass of `daml:Class`, so the ontology itself may be viewed as inconsistent. We fixed these problems, and translated the DAML+OIL ontology into OWL ontology using a script. The upper part of the ontology corresponding to the figure 8.1 is as follows.

$$\begin{aligned} \textsc{Address} &\sqsubseteq \top \\ \textsc{Connection} &\sqsubseteq \top \\ \top &\sqsubseteq \forall \mathsf{source}^{-1}.\textsc{Connection} \end{aligned}$$

[1] In the figures 8.1, 8.2, 8.3, ontologies are expressed in RDFS/OWL. A rectangle corresponds to a class, a rounded rectangle corresponds to an object property, and an oval corresponds to datatype property

[2] The predecessor of OWL.

$$\top \sqsubseteq \forall \mathsf{source}.\textsc{Address}$$
$$\top \sqsubseteq \forall \mathsf{destination}^{-1}.\textsc{Connection}$$
$$\top \sqsubseteq \forall \mathsf{destination}.\textsc{Address}$$
$$\top \sqsubseteq \forall \mathsf{duration}^{-1}.\textsc{Connection}$$
$$\top \sqsubseteq \forall \mathsf{duration}.\mathsf{xsd{:}time}$$

8.1.2 Boeing Transportation Ontology

We named this ontology [34] as "Boeing" one only because of its author affiliation. We don't know whether this ontology is actually used in Boeing.

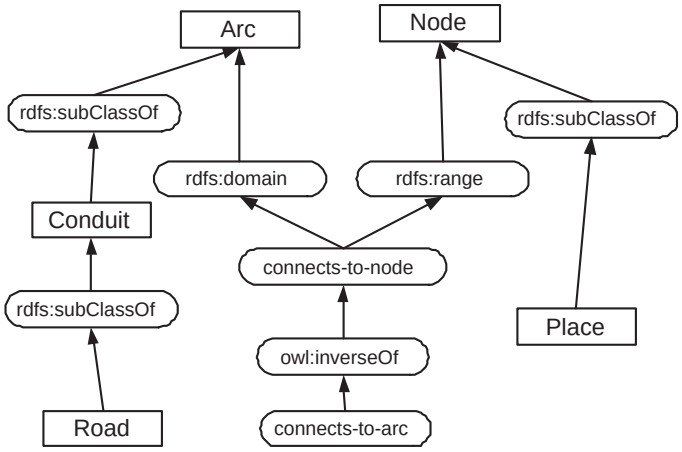

Figure 8.2: Boeing transportation ontology (upper part with main concepts).

This ontology is expressed in the Knowledge Interchange Format KIF [48] and is designed in a layered way. First, a graph ontology describing graph theory with nodes and arcs is described (see figure 8.2). Using this ontology, a basic transportation ontology is introduced, that uses graph "node" as transportation "place" and graph "arc" as transportation "conduit". The transportation ontology defines above all transporting action and transportee, and defines constraints on them. Then, as an example, car ontology is introduced, that is based on the transportation ontology. This ontology for example specializes "transporting" to "driving", "vehicle" to "car", and defines additional constraints (such as that "person" is "driving", which is not always the case of the more general concept "transporting").

We translated this ontology from KIF to OWL manually. All concepts and relations were translated to OWL, including basic constraints expressed in KIF. Since KIF is more expressible than OWL, the translation was lossy.

Some of the axioms (such as more complicated rules) could not be expressed in OWL. The upper part of the ontology corresponding to the figure 8.2 is as follows.

$$\begin{aligned} \textsc{Arc} &\sqsubseteq \top \\ \textsc{Node} &\sqsubseteq \top \\ \textsc{Road} &\sqsubseteq \textsc{Conduit} \\ \textsc{Conduit} &\sqsubseteq \textsc{Arc} \\ \textsc{Place} &\sqsubseteq \textsc{Node} \\ \top &\sqsubseteq \forall \mathsf{connects\text{-}to\text{-}node}^{-1}.\textsc{Arc} \\ \top &\sqsubseteq \forall \mathsf{connects\text{-}to\text{-}node}.\textsc{Node} \\ \mathsf{connects\text{-}to\text{-}arc} &\equiv \mathsf{connects\text{-}to\text{-}node}^{-1} \end{aligned}$$

8.1.3 Material Transportation Ontology

The material transportation ontology [91] was originally designed for material transportation in a factory. Its use is described e.g. in [91].

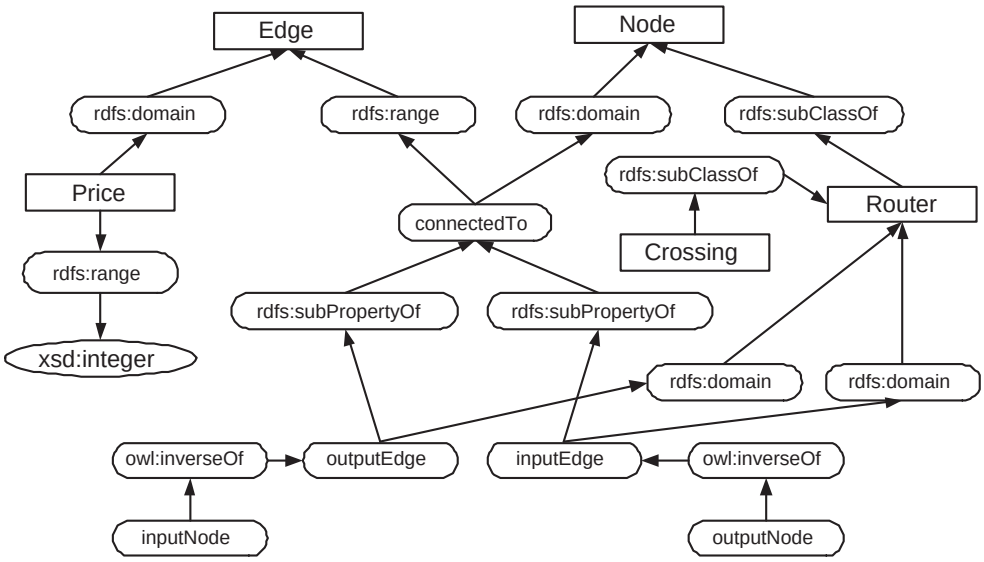

Figure 8.3: Material transportation ontology (upper part with main concepts).

This ontology is expressed as the list of XML tags and attributes with their meaning described using natural language in a tabular form (the same approach is used for all ontologies in FIPA specifications). Based on this, we

have created a transportation ontology expressed in OIL[3] [40], and added the axioms corresponding to the natural language description. The OIL ontology is described in [68]. For the purposes of this thesis, we have converted this OIL ontology into DAML+OIL format and then to OWL format (see figure 8.3) using a conversion script. The upper part of the ontology corresponding to the figure 8.3 is as follows.

$$
\begin{aligned}
\textsc{Edge} &\sqsubseteq \top \\
\textsc{Node} &\sqsubseteq \top \\
\textsc{Router} &\sqsubseteq \textsc{Node} \\
\textsc{Crossing} &\sqsubseteq \textsc{Router} \\
\top &\sqsubseteq \forall \mathsf{connectedTo}^{-1}.\textsc{Node} \\
\top &\sqsubseteq \forall \mathsf{connectedTo}.\textsc{Edge} \\
\mathsf{inputEdge} &\sqsubseteq \mathsf{connectedTo} \\
\top &\sqsubseteq \forall \mathsf{inputEdge}^{-1}.\textsc{Router} \\
\mathsf{outputEdge} &\sqsubseteq \mathsf{connectedTo} \\
\top &\sqsubseteq \forall \mathsf{outputEdge}^{-1}.\textsc{Router} \\
\mathsf{outputNode} &\equiv \mathsf{inputEdge}^{-1} \\
\mathsf{inputNode} &\equiv \mathsf{outputEdge}^{-1} \\
\top &\sqsubseteq \forall \mathsf{price}^{-1}.\textsc{Edge} \\
\top &\sqsubseteq \forall \mathsf{price.xsd{:}int}
\end{aligned}
$$

8.2 Translation

To enable agents to communicate, a translation between messages can be used. As soon as information about mapping between ontologies (such as rules or relations between classes) is available, the information can be used for translation of messages between different ontologies.

We assume that messages contain information about particular state of the world, information about actions, beliefs, etc. This information is expressed in a vocabulary of some ontology. During translation, the parts of the message that are parts of the source ontology are translated to the target ontology.

We are interested in expressing translation in declarative form, that is easier to maintain and to share (for example during communication) than a procedural form expressed as a part of a program.

[3]Ontology Inference Layer — a description logic that is the base for the semantics of DAML+OIL, the predecessor of OWL.

8.2.1 Translation Using Rules

First we discuss our proposal of translation using rules. Rules are generally viewed as the next layer over ontologies and logic for achieving the goals of the semantic web (see the figure 5.1). Since the current proposed W3C standards [55, 17] are still emerging, we have decided to use the rules as implemented in Jena [16]. The syntax and semantics of these rules is described in [16].

Let us illustrate the mapping specification on a few examples. We will start with the mapping from the Boeing transportation ontology to the Material one. The easiest rule is for one to one mapping, where a class or a property name can be translated exactly to another class or property name in the other ontology. Of course, this is possible only when such lossless and exact translation exists. In our case, we can translate "Arc" directly to "Edge". The corresponding rule needs to be specified for two cases — when the class is used in subject and when the class is used in object in the triples.

```
[ (boeing:Arc ?p ?o) ->
   remove(0) (material:Edge ?p ?o) ]
[ (?s ?p boeing:Arc) ->
   remove(0) (?s ?p material:Edge) ]
```

The triples in the second part of the rule are added when the triples in the first part of the rule are found in the message that should be translated (i.e. if there is a match for the triple pattern). The `remove(i)` primitive removes the i-th matched triple.

Note that we are specifying one way translation only. In this particular case, "Edge" can be translated back to "Arc", but generally, this is not always the case for any two concepts. For example, "Conduit" from Boeing ontology doesn't have its direct counterpart in the material ontology. So, even when "Conduit" can be translated to "Edge" (with the loss of specialization), "Edge" cannot be translated to "Conduit", but only to "Arc". Translation of "Edge" is illustrated in the first case in the figure 8.4, translation of "Conduit" is illustrated in the second case in the same figure.

The same approach can be used for one to one mappings of properties. For example, in the mapping from the material transportation ontology to the Berlin one, the following rules can be used:

```
[ (?e material:outputNode ?n) ->
   remove(0) (?e berlin:destination ?n)]
[ (?e material:inputNode ?n) ->
   remove(0) (?e berlin:source ?n) ]
```

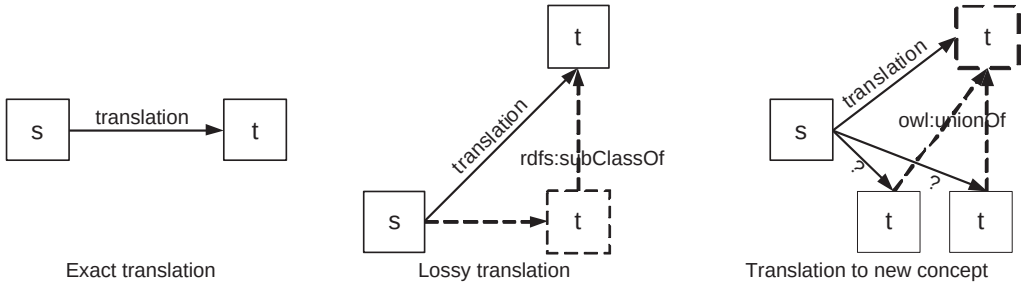

Figure 8.4: Illustration of translations from source to target ontologies in various situations.

The source ontology may have concepts that do not have direct counterparts in the target ontology, but these counterparts can be easily constructed — e.g. by joining more concepts in a union (which is illustrated as the third case in the figure 8.4) or by adding additional constraint. In both cases, a new concept is in fact created in the target ontology, and is then used for translation. In the case of classes, this new concept would typically correspond to an RDF anonymous node. However, it is not possible to use anonymous node for property definition, so a new property must be explicitly created.

For example, when translating from the Boeing transportation ontology to the Berlin one, then the property "connects-to-node" can be translated to the union of "destination" and "source". The rule then looks as follows.

```
[ (?e boeing:connects-to-node ?n)
  makeTemp(?a1) makeTemp(?a2)     ->
  remove(0) (?e berlin:destSrc ?n)
  (berlin:destSrc owl:unionOf ?a1)
  (?a1 rdf:first berlin:destination) (?a1 rdf:rest ?a2)
  (?a2 rdf:first berlin:source) (?a2 rdf:rest rdf:nil) ]
```

In RDF triples, this means that a new property `destSrc` is created as the union of destination and source properties. The `makeTemp` primitive will create a new anonymous RDF node. Due to the fact that the predicate in RDF triple cannot be formed by anonymous node, a new named property had to be constructed rather than using anonymous union. However, the approach of anonymous new resource can be used for classes.

Let us add that the rules can use additional primitives for more complicated matchings or target triples creations. Using the rules as explained above, mappings can be created between different ontologies. These rules are then directly used to translate messages between ontologies.

Note that the head of the rule can contain multiple concepts, and so a rule expressing $n : m$ translation can be defined and used as well. This allows to describe more complex mappings than the ones described in the figure 8.4. Also, the primitives in the rules include arithmetical operations, so a conversion for example between different metric systems in different ontologies can be expressed and applied.

8.2.2 Example Message Translation

To give an overview of the whole process, we will illustrate one example message translation. An agent that uses Boeing transportation ontology is informing about a connection between two places (for example as an answer to a query, or as otherwise requested information). The message content (for the rest of the FIPA ACL message in OWL, see the section 8.3) is:

```
@prefix : <http://boeing.com/transportation/A1KB#> .
@prefix boeing: <http://boeing.com/transport/Ontology#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
:placeA a boeing:Place.
:placeB a boeing:Place.
:path1 a boeing:Conduit;
    boeing:connects-to-node :placeA, :placeB.
```

The rules fired during translation are as follows (note that these are only selected rules fired during this particular translation):

```
[ (?e boeing:connects-to-node ?n)
  makeTemp(?a1) makeTemp(?a2)     ->
  remove(0) (?e berlin:destSrc ?n)
  (berlin:destSrc owl:unionOf ?a1)
  (?a1 rdf:first berlin:destination) (?a1 rdf:rest ?a2)
  (?a2 rdf:first berlin:source) (?a2 rdf:rest rdf:nil) ]

[ (?s ?p boeing:Place) ->
  remove(0) (?s ?p berlin:Address) ]

[ (?s ?p boeing:Conduit) ->
  remove(0) (?s ?p berlin:Connection) ]
```

The message translated from Boeing transportation ontology to the Berlin one, as returned by the ontology service agent, is then:

```
@prefix : <http://boeing.com/transportation/A1KB#> .
@prefix berlin: <http://berlintransport.com/Ontology#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
:placeB a berlin:Address .
:placeA a berlin:Address .
berlin:destSrc
   owl:unionOf (berlin:destination berlin:source).
:path1 a berlin:Connection;
   berlin:destSrc :placeB, :placeA .
```

8.2.3 Translation Using SPARQL

The SPARQL language [82] is designed as an RDF query and access language (see section 5.4), but it is powerful enough to allow constructing RDF graphs. In particular, `CONSTRUCT` is a query form that allows specifying construction of RDF graphs from other RDF graphs depending on matches, and so can be used to express transformation rules [84].

This feature can be used to express operations similar to the rule translation discussed above. Let us follow the same examples as above for illustration. The exact (1 : 1) translation from the class “Arc” (of Boeing ontology) to the class ”Edge” (of Material ontology) used to denote a type of an individual is expressed as follows.

```
PREFIX berlin: <http://berlintransport.com/Ontology#>
PREFIX material: <http://transport.com/onto.owl#>

CONSTRUCT { ?a a boeing:Arc.}
WHERE     { ?a a material:Edge. }
```

When the query is evaluated it is returned for any individual of type `material:Edge` that it is of type `boeing:Arc`. A single query expressing translation of all triples with possible occurrences of `material:Edge` is then as follows.

```
CONSTRUCT
{
   ?a ?ra boeing:Arc.
```

```
   boeing:Arc ?rb ?b.
} WHERE {
   OPTIONAL { ?a ?ra material:Edge. }
   OPTIONAL { material:Edge ?rb ?b. }
}
```

The `OPTIONAL` matching is necessary, since for `CONSTRUCT` query, the `WHERE` part is first matched and then the RDF graph as specified in the `CONSTRUCT` part is constructed. Without `OPTIONAL`, if any of the triples would not match, then no result would be returned. If a variable used in a triple in the `CONSTRUCT` form is not bound from the optional matching, then the triple will not appear in the result.

The translation can be also expressed in multiple queries. In that case the `OPTIONAL` construct is not needed. All of these queries would have to be performed and the results would have to be inserted to a single RDF graph that is then the translated message.

The translation query involving translation to a new property as illustrated in the section 8.2.2 follows. Since N3 notation with the syntactic shortcuts is used in SPARQL queries, expressing the union of properties (i.e., RDF list) is simpler when compared with the syntax that needs to be used in rules.

```
CONSTRUCT
{
   ?add a berlin:Address.
   berlin:destSrc owl:unionOf
          ( berlin:destination berlin:source).
   ?e berlin:destSrc ?n.
} WHERE {
   OPTIONAL {?add a boeing:Place.}
   OPTIONAL {?e boeing:connects-to-node ?n.}
}
```

The feature of translation using SPARQL `CONSTRUCT` queries is that only what is specified to be constructed is returned as translation. With rules, the parts of the message that are not explicitly defined to be translated are left intact. Whether this is an advantage or disadvantage depends on the intended use — it may be helpful to get the rest of the message even if it is not understood in the given ontology, and it may be helpful to get only those parts of the message that can be guaranteed to be understood in the given ontology.

Note that even for including all RDF, RDFS and OWL constructs (that should be understood by agents without committing to a special ontology) the SPARQL translation query has to contain the full vocabulary of these languages with all the possible occurrences in subject, predicate and object, to get all the triples in the translated message.

8.2.4 Translation Using OWL Reasoning

The RDFS and OWL languages can be used to express taxonomic relations between classes and properties from different ontologies. These relations can be used to express translation information. The relation CONDUIT $\sqsubseteq$ EDGE is expressed as the triple

`boeing:Conduit rdfs:subClassOf material:Edge.`

Similarly, the relation ARC $\equiv$ EDGE is expressed as the triple

`boeing:Arc owl:equivalentClass material:Edge.`

Similar relations can be expressed between properties using RDFS and OWL relationships `rdfs:subPropertyOf` and `owl:equivalentProperty` (see table 5.5).

In this way, we can express the relations between classes and properties that define all kinds of translations illustrated in the figure 8.4. Exact translation is expressed using equivalence, lossy translation is expressed using subsumption, and translation to a new concept is expressed using combination of subsumption and DL means to construct a new concept or property.

Following the example with mapping from `boeing:connects-to-node` to Berlin ontology, it could be stated that connects-to-node $\equiv$ destination$\sqcup$source — however, due to the limitations of the OWL DL caused by embedding DL into RDF, this must be stated via destSrc property as above.

An OWL reasoner can be used to derive relationships in both ontologies based on the mapping. Following the example in the section 8.2.2, the mapping is expressed as follows.

$$\begin{aligned} \text{ADDRESS} &\equiv \text{PLACE} \\ \text{destSrc} &\equiv \text{destination} \sqcup \text{source} \\ \text{connects-to-node} &\equiv \text{destSrc} \end{aligned}$$

When the message to be translated, i.e.,

$$\text{PLACE}(placeA)$$

$$\begin{aligned}&\textsc{Place}(placeB)\\&\textsc{Conduit}(path1)\\&\mathsf{connects\text{-}to\text{-}node}(path1, placeA)\\&\mathsf{connects\text{-}to\text{-}node}(path1, placeB)\end{aligned}$$

is combined with the mapping information, a reasoner will derive all the conclusions — 81 triples are in the deductive closure computed by Pellet in this simple case, since all the OWL axioms are applied and the result is added to the existing RDF graph. These conclusions include the translation, i.e.,

$$\begin{aligned}&\textsc{Address}(placeA)\\&\textsc{Address}(placeB)\\&\textsc{Connection}(path1)\\&\mathsf{destSrc} \equiv \mathsf{destination} \sqcup \mathsf{source}\\&\mathsf{destSrc}(path1, placeA)\\&\mathsf{destSrc}(path1, placeB)\end{aligned}$$

but it is not easy to extract the translation from all the triples automatically. Custom filter for URIs of the resources can be used, so that only those triples would be selected that contain only resources of the URIs that an agent understands (i.e., of ontologies that the agent is committed to). This will eliminate part of some unnecessary deductions, but there will still remain other triples that are not necessary to be contained in the translation.

However, this may mean no problem when an agent is able to handle part of the graph that is expressed in the requested ontology and ignores the rest. In this case, the whole deductive closure can serve as a translation, and only relevant parts of the result are used. This is compatible with the layering of the Semantic Web languages — agents may get information that they are not able to process fully, but the part that they understand can be used.

Our conclusion is that OWL is suitable for expressing relationships between elements (classes, properties, individuals) of ontologies and knowledge bases. However, together with reasoning applied to get the translation the result may not be directly usable for translations as illustrated above, i.e., translations between messages for agent communication so that an agent would get the message only in the desired ontology.

8.2.5 Comparison

We have shown three ways of translation between OWL ontologies — Jena rules, SPARQL `CONSTRUCT`, and OWL reasoning. The result after applying

Triples #	100	200	300	400	500
Jena Rules [s]	0.01	0.01	0.03	0.03	0.04
SPARQL [s]	2.0	35	132	528	1232
OWL Reasoning [s]	0.02	0.02	0.02	0.04	0.06

Table 8.7: Comparison of translation time depending on the number of triples.

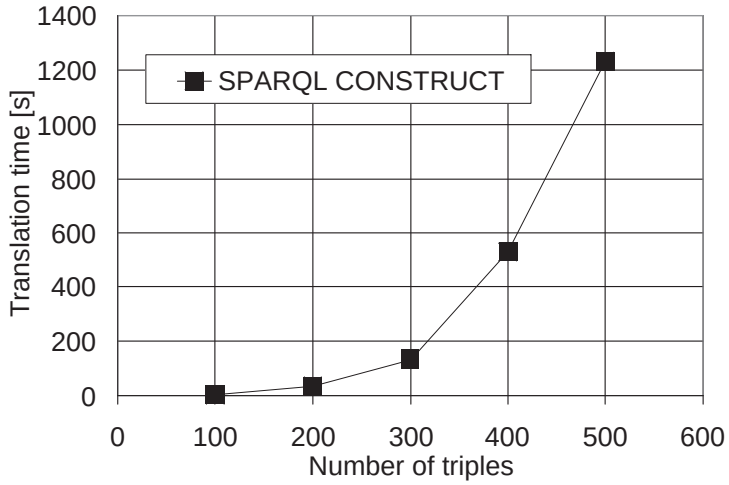

Figure 8.5: Translation time depending on the number of triples for SPARQL `CONSTRUCT` query.

rules or performing SPARQL query is only the translation. If something is not specified to be translated, it is not included in the result. On the other hand, translation using OWL reasoning as we have shown it will include translation to the original graph and moreover will include the deductions of both the original graph and translation. Another difference is that general Jena rules have an advantage of the possibility of specifying special operations such as arithmetic or string operations, and custom special operations can be added. Only some of these operations can be used in SPARQL `CONSTRUCT` specification, and none of them can be used in OWL specification.

We have compared the performance of these three translation methods in the table 8.7 — as we can see, both Jena rules and OWL reasoning can be used for hundreds of triples (a typical message between agents will not usually contain so many triples), but SPARQL `CONSTRUCT` in the Jena im-

plementation takes very long time and from the figure 8.5 we can see that it is not usable for larger number of triples. The evaluation of Jena rules and OWL reasoning performance for larger number of triples can be found in the section 9.5.

8.3 Translation in Ontology Agent

To help agents with communication when they use different ontologies, the translation can be requested from and is performed by the OWL ontology agent[4] that we have described earlier in the section 7.3. The `ontoagent` ontology is used for the specification of translation, so for example the request to translate the example mentioned above is expressed as follows.

```
(REQUEST
 :sender  ( agent-identifier  :name BOEING@MO:1099/JADE)
 :receiver  (set ( agent-identifier :name OA@MO:1099/JADE))
 :content  "
    @prefix agent: <http://fipa.org/ns/agent.owl#> .
    @prefix ontoagent: <http://fipa.org/ns/ontoagent.owl#> .
    [] a ontoagent:Translate;
    agent:hasActor
    [ a ontoagent:OntologyAgent;
      agent:hasAgentID
          [ a agent:AgentID;
            agent:hasName \"OA@MO\" ] ];
    agent:hasParameter
    [ a ontoagent:OntoNameFromParam;
      agent:hasDatatypeValue
                  "http://boeing.com/transport/Ontology#" ];
    agent:hasParameter
    [ a ontoagent:OntoNameToParam;
      agent:hasDatatypeValue
                  "http://berlintransport.com/Ontology#" ];
    agent:hasParameter
    [ ontoagent:StringGraphParam;
      agent:hasDatatypeValue \"\"\"
       @prefix : <http://boeing.com/transportation/A1KB#> .
       @prefix boeing:
```

[4]In the current implementation the translation is performed only using Jena rules that have to be predefined for the agent.

```
                  <http://boeing.com/transport/Ontology#> .
       :placeA a boeing:Place.
       :placeB a boeing:Place.
       :path1 a boeing:Conduit;
           boeing:connects-to-node :placeA, :placeB.
      \"\"\" ]. "
 :reply-with  PAA@MO:1099/JADE1172476587793
 :language N3
)
```

The ontology agent reads the request and performs the translation as described above. The message translated to other ontology is returned so that the agent using Boeing ontology can use it for communication with an agent that uses Berlin ontology.

```
(INFORM
 :sender  ( agent-identifier  :name OA@MO:1099/JADE)
 :receiver  (set ( agent-identifier :name BOEING@MO:1099/JADE)
 :content  "
    @prefix : <http://boeing.com/transportation/A1KB#> .
    @prefix berlin: <http://berlintransport.com/Ontology#> .
    @prefix owl: <http://www.w3.org/2002/07/owl#> .
    :placeB a berlin:Address .
    :placeA a berlin:Address .
    berlin:destSrc
       owl:unionOf (berlin:destination berlin:source).
    :path1 a berlin:Connection;
       berlin:destSrc :placeB, :placeA . "
 :in-reply-to  PAA@MO:1099/JADE1172476587793
 :language  N3
)
```

We note that the information for translation (i.e., rules in our case) have to be supplied to the agent. The agent is not able to derive this information. We discuss the possibility of learning translation between ontologies by pointing game in a shared environment in [72].

8.4 Architectural Considerations

In the previous text we have described how the translation of a single message can proceed. We will now discuss the architectures used for translation of

messages in communication between agents. There are various approaches that differ in how much it is necessary for agents to be aware of ontology translation. By "aware of ontologies" we mean how much the ontology is hardcoded in the agent or how much agent is able to process the ontology, acquire new ontologies, etc.

Indeed, many agents have their "ontologies" hardcoded, and they are not truly able to accept other ontologies. For example, one of the recommended ways of creating ontologies for Jade agents is to make them frame based, map them to the Java language, and compile them with the agent(s). Clearly, any change of the ontology requires recompilation of the agent, and the agent is not really able to do any reasoning using that ontology.

The approaches to translating messages between ontologies during communication can be divided as follows [76]:

- Agents handle all the translation themselves
- Agents use specialized services or agents to handle the translation
- The translation is completely handled by the multi-agent platform

8.4.1 Agents Handle all the Translation Themselves

The first case, where agents handle all the translation themselves, naturally requires agents to be very knowledgeable about ontologies. The agents must be aware of ontologies and must be able to reason about them and to be able to translate messages between them, which also means that they have to support all the ontologies that they want to use for communication with other agents.

In this case, agents are very powerful in the sense of ontology processing, which also possibly enables them to handle very hard tasks like learning new ontologies and learning to translate to previously unknown ontologies. One of the possible approaches to learning to translate between ontologies that uses a shared domain and a pointing game is described in [72]. Also, the fact that agent themselves are able to reason about ontologies means that they are able to make more complicated conclusions about inter-ontology relations themselves, and that they are not reliable on any other service to do this job. In this way, they are completely autonomous as far as ontology processing is concerned.

But there are certainly also some drawbacks. The need of handling all the ontology related tasks by each individual agent means that even the agents that are supposed to do very simple tasks must be able to process and reason

about ontologies. Especially when wrapping a simple service that requires very simple communication, this approach may mean unnecessary overhead.

We can also imagine the mixture of the agents that are able to fully process ontologies and agents that are not aware of ontology processing — until two such agents need to communicate together directly and use different ontologies. This requires a balance between the required capabilities of agents and the practical implementation — for example, one can hardly imagine why two milling machines from two different factories would want to communicate directly. The holonic approach where the upper holons are communicating together and then distribute the tasks for their parts is very appropriate here.

8.4.2 Agents Knowingly Use Specialized Services

The other approach is when agents are aware of ontologies usage, but do not make all the ontology related task themselves, i.e. they use specialized services or agents to handle the translation between ontologies. This approach is supported by the idea of dividing tasks between agents that are specialists for the task.

We have described this approach in detail in our OWL ontology agent in sections 7.3 and 8.3. When using this approach, agents must be aware of existence of ontologies and of potential ontological heterogeneity, however, they do not have to be able to process ontologies or even know anything about other ontologies. They only need to know the ontology accepted by the agent that they want to communicate with and they can ask the ontology agent to make the translation for them during their communication.

8.4.3 Translation is Transparent

The two previous approaches still depend on agents being aware of existence of ontologies. This means that when agents need to be integrated with agents that use other ontologies, they have to know about the possible ontological heterogeneity, or need to be explicitly wrapped by another layer that enables them to process ontologies and make the translation or at least ask for translation.

We have proposed [77] an approach that removes this requirement by integrating the translation directly in the multi-agent or holonic platform. In this case agents are communicating directly, and the platform handles translation of messages transparently when needed. This means that even agents that are not aware of existence of ontologies and do not have any means or intentions to process ontologies can communicate to other agents with similarly limited capabilities even when these agents use different ontologies.

Of course, this means that the platform has to know what ontologies agents are using.

When the translation is transparent to agents then they do not have to be aware of existence of multiple ontologies. The service that makes the translation must have the following inputs:

1. The content of the message to be processed (possibly translated)
2. Ontology used in the processed message (i.e., ontology used by the sending agent)
3. Ontology used by the receiving agent

It is assumed that the message has a common known format, which is satisfied for all FIPA-compliant agents, where the parameters of the message are predefined. For the purposes of the translation there are the following important parameters — `content`, `language` and `ontology`. By being able to decode these fields, we are able to get inputs 1 (message `content`) and 2 (message content `ontology` and `language`[5])

The third needed input, ontology used by every communicating agent, is reported to Directory Facilitator (agent that provides yellow pages service) when the agent is registering. So, after the agent is registered, the Directory Facilitator of the platform knows the ontology of the receiving agent, and can provide this input for the platform.

In the case of the agents described in the chapter 7 that use RDF and OWL for the content of the messages, the situation is a bit different. As we have said, the ontology does not have to be specified in the ACL message explicitly. However, it is specified in the content of the message, so we get the input 2 as well. As for the third input, the agent must again report its preferences to the Directory Facilitator agent.

When the translation service running transparently in the multi-agent or holonic platform has all of these inputs, it can use them for making translations between agents that communicate using different ontologies.

8.4.4 Transparent Translation Implementation

We have made the proof of concept implementation of the transparent message translation idea in the Jade package [5]. Agents do not have to ask for the translation explicitly, that they do not have to know that the other agent

[5]We do not discuss translation between different languages, which might be also needed during the communication. However, the approach is very similar to translating between ontologies, at least from the architectural point of view discussed in this thesis.

uses different ontology, and most importantly, they do not have to be aware of any ontology processing.

The agents in Jade register with Directory Facilitator, and inform DF about ontologies they understand. Then, whenever a message is being delivered to the agent, the multi-agent platform makes sure that the message is translated to appropriate ontology.

We embedded the translation directly into the Jade platform send command. Whenever send command is processed, the message is translated for each receiving agent (a message can have multiple receivers). The AMS agent queries the DF about what ontology the receiving agent uses, performs the translation, and lets the translated message to be sent to the receiving agent.

8.4.5 Discussion

The approach of transparent translation has one obvious advantage — the only action needed from agents is to inform about the ontology they understand during the registration with the DF. After that, the multi-agent platform itself will take care of delivering messages in the correct ontology. This approach means that only the underlying platform needs to be changed, while agents do not need any change at all to be able to communicate with other agents that use different ontologies. Moreover, agents do not have to be aware of ontologies, and they even do not have to know the ontology accepted by the receiver of their messages.

For manufacturing agents that represent physical devices, whose primary goal is not information processing, it is easier to adapt the underlying platform instead of changing and enhancing these agents. This approach is ideal when integrating existing agents to agents from for example different company when the change of all the involved agents is not possible or would be too hard.

Of course, there are also some disadvantages. Since the agents are not aware that a translation may have been made before they received their message, they may not count with for example uncertainty during translation that could change the meaning of the message[6]. When the agents are aware of ontology processing in their communication, they may be more cautious.

The table 8.8 shows the comparison of performance for different architectures. The architecture of very powerful agents where agents make the translation themselves is the fastest one — the agents know what they need and just do the translation. The architecture where for each sent message

[6]We do not use any uncertainty in our translation rules, however exact translation between two ontologies is not always possible, see figure 8.4.

Translation architecture	Time per message [ms]
Agents alone	2.67
Ontology agent	11.14
Platform	2.70

Table 8.8: Comparison of the performance of translation architectures.

another two messages to and from ontology agent are needed, is obviously the slowest one. The architecture where the translation is made directly in the platform is almost as fast as if the translation would be processed by agents themselves. The small delay is caused by decoding of the message parameters in the platform — in opposite to immediate translation by agents themselves.

Chapter 9

Application: Semantic Search in Manufacturing Industry

In this chapter we describe an application of the framework described in the previous text to the domain of semantic search in manufacturing industry.

The design of an assembly line is very knowledge intensive task. The relevant information is stored in many forms, and each form requires special treatment. Often different forms of information overlap or at least information can be somehow connected.

We describe a framework for a semantic search within files produced for assembly line design. The files are stored in a semantically enriched form that can be searched. An important feature is that this enriched form is generated automatically — no human annotations are needed for producing a searchable form.

9.1 Overview and Motivation

First we describe two products that are used for assembly line design, operation, and maintenance — ladder logic programming package RSLogix 5000 and human-machine interface package RSView Enterprise. Both of them produce specific data and it is often needed to search within these data.

9.1.1 RSLogix 5000

RSLogix 5000 is a family of IEC 61131-3 [57] compliant ladder logic programming package that supports Allen-Bradley Programmable Logic Controllers (PLCs). The 61131-3 is the standard for programmable controller programming languages and specifies specifies the syntax, semantics and display for

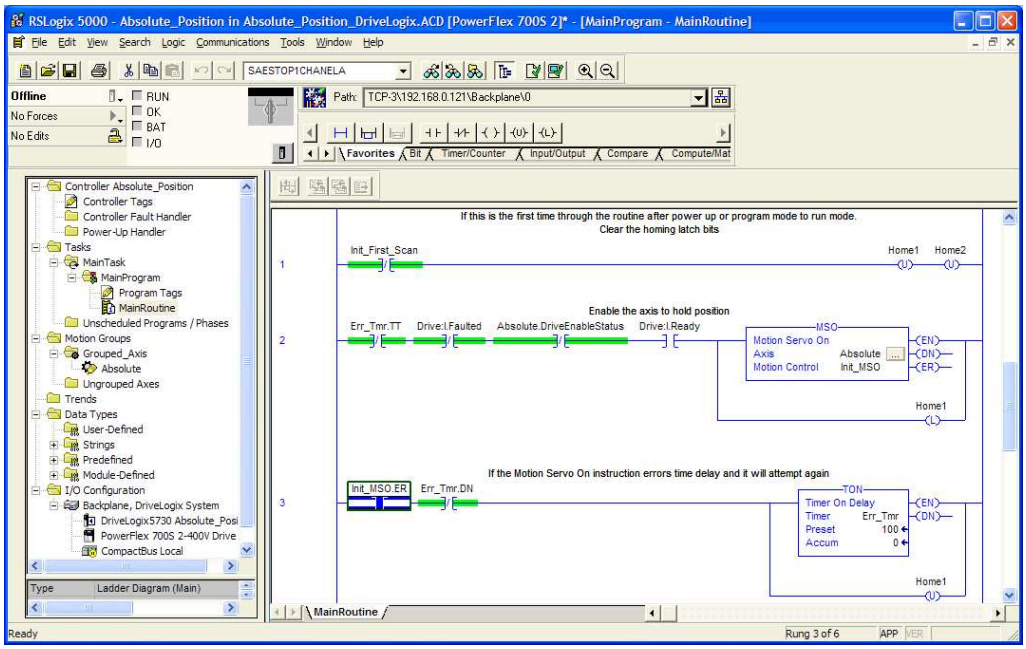

Figure 9.1: RSLogix 5000 screenshot — editing ladder code.

the ladder diagrams, sequential function charts, function block diagrams, structured texts and instruction lists.

The ladder logic was originally invented to describe logic made from relays and it still serves as a method of drawing electrical logic schemes. The name is based on the fact that programs in this language resemble ladders with two vertical "rails" and a series of horizontal "rungs" between them. Ladder logic is useful because a wide variety of engineers and technicians can understand and use it without much additional training. It is used to program PLCs where a sequential control of a process or manufacturing operation is required. It is intended for simple but critical control systems; however, the ladder code may contain very complex instructions today so complex automation systems can be controlled using it.

Ladder logic can be thought of as a rule-based language, rather than a procedural language. A rung in the ladder represents a rule. When implemented with relays and other electromechanical devices, the various rules execute simultaneously and immediately. When implemented in a programmable logic controller, the rules are typically executed sequentially by software, in a loop. By executing the loop fast enough, typically many times per second, the effect of simultaneous and immediate execution is obtained. An example of a ladder code in the RSLogix screenshot is shown in the figure 9.1.

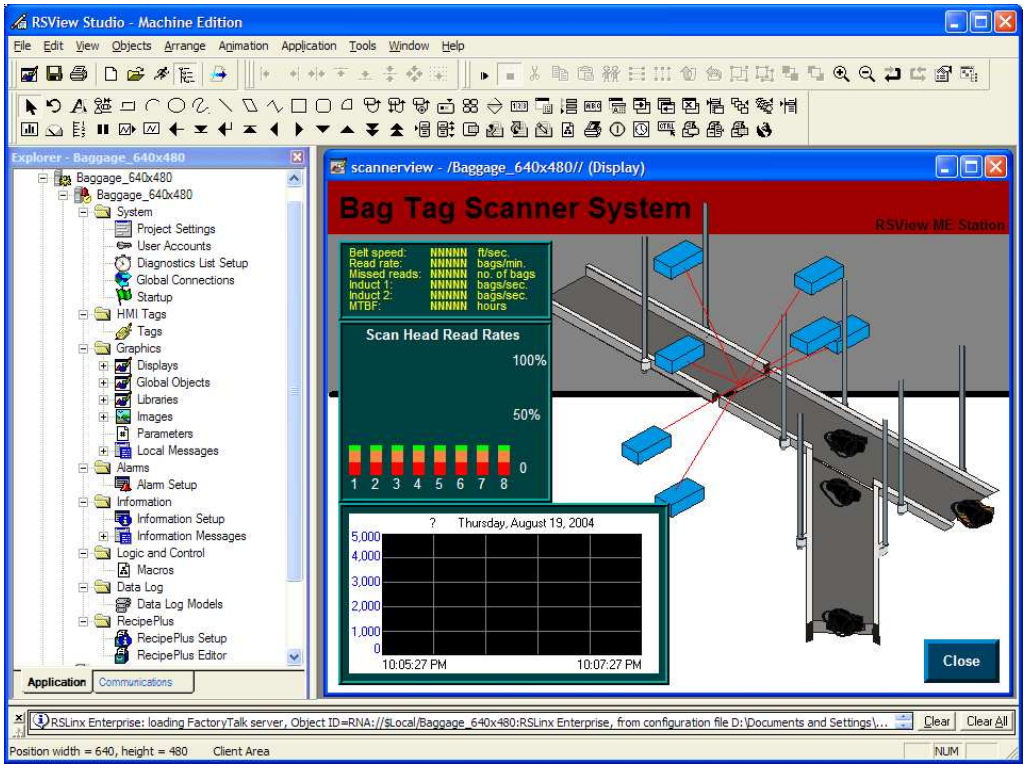

Figure 9.2: RSView Enterprise ME screenshot — editing display.

9.1.2 RSView Enterprise

RSView Enterprise is a line of software products for industrial visualization and Human-Machine Interface (HMI). The HMI is a part of so called Supervisory Control And Data Acquisition (SCADA) system that is used to monitor and control automated processes. The system is typically connected to PLCs (that control devices) and is usually used to control a complete site or a system on a higher level than a single PLC. The SCADA system may allow an operator to change control parameters or even to control devices to some degree manually. This process takes place through HMI that presents process data to a human operator, and through which the human operator controls the process.

The HMI software allows to design a user interface for controlling a process and allows to run the designed interface on terminals connected to PLCs. An HMI can range from a set of simple push buttons showing on small screens to systems showing historical analysis of data in graphical forms.

RSView Enterprise is such a system — it allows to design set of displays,

to define the flow between these displays, to connect objects on the displays to PLCs, and to do basic programming of these displays. A simple example is a push button connected to PLC that would trigger start of a process. The connection is made via so called tags that are in fact variables (in the sense of programming language variables) stored in PLC or in HMI project.

An example of a display design in RSView Enterprise Machine Edition (ME) is shown in the figure 9.2.

9.1.3 Semantic Search Motivation

The machinery in an assembly line is controlled by many PLCs that are connected to many panels running HMIs. The design and maintenance of both ladder logic and HMI displays requires many projects in software like RSLogix and RSView. In addition, these projects are interconnected and are often versioned (i.e., multiple versions of a project exist and it may be needed to return to older versions). The assembly line is designed by multiple engineers who need to maintain their own projects and also need to communicate together.

For these purposes it is often needed to search within these projects and relate the projects together. This can be made by individually opening each project[1] when looking for a particular search pattern, but the infrastructure presented in the previous chapters can help with searching for more complicated patterns. A few examples of the queries follow:

- RSLogix related query: *Find where XIO instructions for tags Err_Tmr.TT and Drive:I.Faulted occur in one rung.*
- RSLogix related query: *In which rung the constant 0.5 is used in the SUB instruction as "Source B"?*
- RSView related query: *Find all the text objects with background color navy that have "%" in their caption.*
- RSView related query: *Find all BarGraph controls that have threshold value 1000 and show to which tags the properties of these controls are connected.*
- Combined query: *Find which RSLogix projects have a tag used in a Gauge control on any RSView display.*

[1]Since these are binary files, even the free text search is not possible using the standard means usually offered by operating system.

The information for semantic search queries should be created without human intervention as much as possible. The semantics of particular file content (project) should be generated automatically for a given ontology, so that the information can be continuously updated for accurate search.

9.2 Conversion to RDF and OWL

In order to be able to search within RSLogix and RSView projects using the infrastructure described in the previous chapters, we have to store the information from projects in RDF. For both RSLogix and RSView the same way is used — the information is first exported to XML to get a data interchange format. Then, using the framework described in section 7.4 the XML form is converted to RDF and an additional information is added to define additional semantics suitable for search, such as part-of information.

9.2.1 RSLogix 5000

An RSLogix 5000 project consists primarily of routines and tags. Each routine contains ladder code. The routines are organized in tasks and programs. The tags are scoped per PLC or programs may have their own tags. Other information includes custom data types, motion groups and I/O configuration.

A part of a routine can be exported to XML manually from RSLogix. The full export is not supported, however, a converter from RSLogix project to XML was written that allows to export selected parts of RSLogix project[2].

Using a set of exported XML files and running the XML to RDF conversion wizard (see section 7.4), an ontology was produced that we then enhanced manually. Every RDF graph produced by conversion from a particular XML file is then a set of instances (A-Box) for the ontology (T-Box), and reasoning within the ontology can be used to infer additional facts that are not present explicitly in the exported graph.

The information added includes:

- Domains and ranges for properties — where applicable. Sometimes union of multiple classes needed to be used as a domain. This information is useful for example when presenting the information to the user to see possible continuation of the selected part of a RDF graph during query construction.

[2]The converter was written by Michal Fořtík. We use the result of the conversion, i.e., the XML form, for conversion to RDF.

- Since the structure of RDF graph is nested (from the conversion from XML), it would be complicated to formulate a query by explicitly writing the full path from the root resource to the leaf resource with the searched information. To make this easier, a transitive object property contains was introduced. Each of the property that represents the "contains" relation is a subproperty of the contains property. When querying an OWL reasoner for the transitive relation, the whole transitive closure is returned, and so the query is easier to formulate.

- Similarly, especially for free text search, a transitive datatype property hasSearchableText was introduced that is a super-property of the datatype properties that have searchable value.

The whole RSLogix ontology has 54 classes, 53 object properties and 251 datatype properties. A few selected axioms from the ontology around the concept CONTROLLER follow[3]:

$$
\begin{aligned}
&\text{TRANS}(\mathsf{contains})\\
\mathsf{containsController} &\sqsubseteq \mathsf{contains}\\
\top &\sqsubseteq \forall \mathsf{containsController}^{-1}.\text{RSLOGIX5000CONTENT}\\
\top &\sqsubseteq \forall \mathsf{containsController}.\text{CONTROLLER}\\
\top &\sqsubseteq \forall \mathsf{containsTags}^{-1}.\text{CONTROLLER} \sqcup \text{PROGRAM} \sqcup\\
&\qquad \text{PARAMETER}\\
\mathsf{hasCommDriver} &\sqsubseteq \mathsf{hasSearchableText}\\
\top &\sqsubseteq \forall \mathsf{hasCommDriver}^{-1}.\text{CONTROLLER}
\end{aligned}
$$

9.2.2 RSView Enterprise

An RSView project consists primarily of displays and tags. Each display has graphical elements such as shapes (rectangle, oval, ...), various types of push buttons (momentary, maintained, display change, ...), text input and output controls, and more complicated elements like a trend graph. These elements have properties of which some may be bound to tags that serve as I/O points (they may correspond to RSLogix tags). An RSView project may also contain other settings, such as alarms, information messages, macros, recipes, etc.

RSView supports manual export of displays to XML. Using a set of exported XML files, an ontology was created in a similar way as for RSLogix

[3]The full RSLogix and RSView ontology would take 155 pages together (in the description logic notation), 32 pages without disjoint axioms.

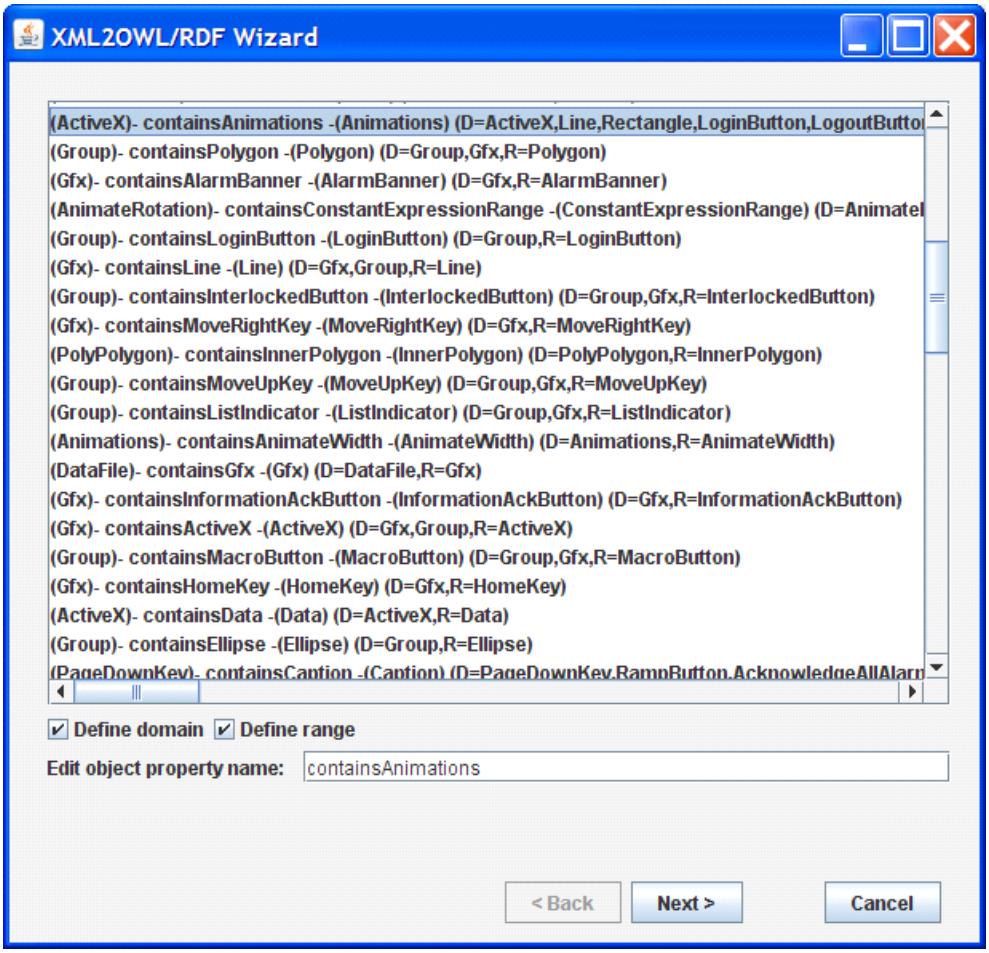

Figure 9.3: XML2OWL/RDF Wizard asking for object properties names and whether to use domain and range for them during the conversion of RSView display files.

(see figure 9.3 for a screenshot of the wizard during conversion). The whole RSView ontology has 94 classes, 93 object properties and 226 datatype properties. A few selected axioms from the ontology around the concept MomentaryButton follow:

$$\begin{aligned} \mathsf{containsMomentaryButton} &\sqsubseteq \mathsf{contains} \\ \top &\sqsubseteq \forall \mathsf{containsMomentaryButton}^{-1}.\textsc{Group} \sqcup \textsc{Gfx} \\ \top &\sqsubseteq \forall \mathsf{containsMomentaryButton}.\textsc{MomentaryButton} \end{aligned}$$

$$
\begin{aligned}
\top &\sqsubseteq \forall \mathsf{hasButtonAction}^{-1}.\textsc{MomentaryButton} \\
\mathsf{containsStates} &\sqsubseteq \mathsf{contains} \\
\top &\sqsubseteq \forall \mathsf{containsStates}^{-1}.\textsc{MultistateButton} \sqcup \\
&\quad \textsc{LatchedButton} \sqcup \\
&\quad \textsc{MomentaryButton} \sqcup \\
&\quad \textsc{MaintainedButton} \sqcup \\
&\quad \textsc{ControlListSelector} \sqcup \\
&\quad \textsc{MultistateIndicator} \sqcup \\
&\quad \textsc{InterlockedButton} \sqcup \\
&\quad \textsc{ListIndicator} \sqcup \\
&\quad \textsc{DisplayListSelector} \sqcup \textsc{Symbol}
\end{aligned}
$$

9.2.3 File Metadata

Both of the ontologies above describe the information contained in a file. In addition to this information, metadata about files need to be stored as well so that they can be used for search. These metadata include:

- fileName – original file name, i.e. on which file the action is performed
- computerName – name of the computer, to be stored together with fileName as the unique identification of the file
- dateCreated – date when the file was created, in `xsd:dateTime` format
- dateModified – date when the file was modified, in `xsd:dateTime` format
- dateAccessed – date when the file was accessed, in `xsd:dateTime` format
- fileSize – file size in bytes
- attrReadOnly – whether the file is read-only, true/false
- attrHidden – whether the file is hidden, true/false
- attrSystem – whether the file is a system one, true/false

All metadata are stored as values of datatype properties of File class individual. The same individual is related to file content (described using the ontologies as shown above) via the contains property.

9.3 Architecture

The proposed architecture of the semantic search system is illustrated in the figure 9.4. The File Converter Agents update information about file systems on individual computers and send the updates to a Storage Agent. The RDF Storage Agent stores all the information and enables to perform semantic search. The user interface for the search is provided by User Interface Agent.

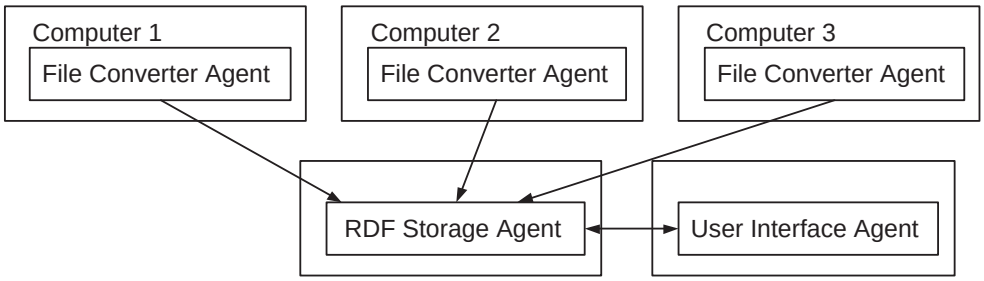

Figure 9.4: Semantic search multi-agent system architecture.

9.3.1 RDF Storage Agent

The RDF Storage Agent is based on the ontology agent described in the section 7.3. It stores information about RSLogix and RSView files in RDF form in a relation database so that the information can be searched and retrieved in a semantically enriched way.

The ontology of messages accepted by the storage agent consists of updating the RDF storage and of querying it. The usage of queries is the same as with the OWL ontology agent — they are expressed in SPARQL. For an example message with SPARQL query see the scenario in the section 7.3.3.

The update actions correspond to the changes in file system — a file can be created, changed, or deleted. When a file is created or changed, the file content is described in the same way as in the case of the ASSERT action with OWL ontology agent (see the section 7.3.1). In addition, file metadata may be specified via hasObjectValue property of ACTION. Since only one value of each metadata property is allowed, the agent deletes old value before asserting new one. If file name is changed, the old file name has to be specified to locate the old information and a special property hasNewFileName is used to specify new name.

The ontology that the RDF Storage Agent accepts is based on the generic `agent` ontology described earlier. It specifies actions with RDF storage that

correspond to file creation, change or deletion. Also, file metadata parameters are specified in this ontology which are then translated to file metadata information for storage for search (i.e., properties in another ontology, see above). A selected part of the ontology follows — the classes and properties from `agent` ontology are imported to the ontology and are used for the definitions.

$$
\begin{aligned}
\textsc{ComputerNameParam} &\equiv \exists \mathsf{hasName}.\{\text{"computerName"}\} \sqcap \\
&\quad \textsc{ActionParameter} \sqcap \\
&\quad = 1\ \mathsf{hasComputerName}.\top \\
\mathsf{hasComputerName} &\sqsubseteq \mathsf{hasDatatypeValue} \\
\top &\sqsubseteq \forall \mathsf{hasComputerName.xsd{:}string} \\
\textsc{FileNameParam} &\equiv \exists \mathsf{hasName}.\{\text{"fileName"}\} \sqcap \\
&\quad \textsc{ActionParameter} \sqcap \\
&\quad = 1\ \mathsf{hasFileName}.\top \\
\mathsf{hasFileName} &\sqsubseteq \mathsf{hasDatatypeValue} \\
\top &\sqsubseteq \forall \mathsf{hasFileName.xsd{:}string} \\
\textsc{FileAction} &\equiv \textsc{Action} \sqcap \\
&\quad \exists \mathsf{hasParameter}.\textsc{FileNameParam} \sqcap \\
&\quad \exists \mathsf{hasParameter}.\textsc{ComputerNameParam} \\
\textsc{FileDelete} &\sqsubseteq \textsc{FileAction} \\
\textsc{FileCreate} &\sqsubseteq \textsc{FileAction} \\
\textsc{FileChange} &\sqsubseteq \textsc{FileAction} \\
\mathsf{hasDateCreated} &\sqsubseteq \mathsf{hasDatatypeValue} \\
\top &\sqsubseteq \forall \mathsf{hasDateCreated.xsd{:}dateTime} \\
\mathsf{hasAttrReadOnly} &\sqsubseteq \mathsf{hasDatatypeValue} \\
\top &\sqsubseteq \forall \mathsf{hasAttrReadOnly.xsd{:}boolean} \\
\mathsf{hasFileSize} &\sqsubseteq \mathsf{hasDatatypeValue} \\
\mathsf{hasXMLData} &\sqsubseteq \mathsf{hasDatatypeValue}
\end{aligned}
$$

9.3.2 File Converter Agents

The file converter agent should observe selected part of file system and send the updated information to the RDF Storage agent[4] using the ontology described above.

Whenever an event requiring updating of file content information occurs the agent should perform automatic conversion from binary format to XML document. The XML document is then sent to the RDF storage agent that

[4]File observation is not implemented yet and so the RDF Storage agent is updated by manually generated messages from this agent so far. Also, the RSView conversion has to be made manually for our prototype.

converts it to RDF automatically. Then the agent merges it with the ontology appropriate for the file type (see above) and takes care about inferring additional information as specified in the ontology. The result (RDF) graph is stored and is used for search.

9.3.3 User Interface Agent

The user interface agent provides an interface for making SPARQL queries. A `SELECT` query is sent to the RDF Storage agent and the response is shown to the user in a tabular form, where columns correspond to the values of variables bound from the query.

9.4 Semantic Search

Let us illustrate an example of a semantic search on two selected samples from the section 9.1.3. The first selected query is search for an RSLogix file: *In which rung the constant 0.5 is used in the SUB instruction as "Source B"?*

We will not show the full messages as they are similar to the ones illustrated in the section 7.3.3; only the SPARQL query together with the results is shown. The query expressed in SPARQL is as follows:

```
PREFIX rock: <http://rockwell.com/NS/Generic.owl#>
PREFIX logix: <http://rockwell.com/NS/RSLogix.owl#>
SELECT ?computername ?filename ?val ?text
WHERE
{
    ?ins a logix:Instruction. # look for instruction
    ?ins logix:hasName "SUB".
    ?ins rock:contains ?param.
    ?param a logix:Parameter.
    ?param logix:hasNumber "1". # Source B
    ?param logix:hasValue "0.5".
    ?ins logix:hasValue ?val. # get the full description
    ?rung a logix:Rung. # in which rung is it?
    ?rung rock:contains ?ins.
    ?rung logix:containsComment ?comment.
    ?comment logix:hasTagText ?text. # we want the text
    ?file a rock:File. # in which file is the rung?
    ?file rock:contains ?rung.
```

```
    ?file rock:hasComputerName ?computername.
    ?file rock:hasFileName ?filename.
}
```

Note that the query uses knowledge expressed in ontology and inferred over the data. Without the additional knowledge the query would be much more complicated. The result returned by the user interface agent in a tabular form[5] is as follows.

Name	Value
?computername	"euczprgmobitko4"
?filename	"C:\\tmp\\test\\logix\\Motion.acd"
?val	"SUB(gear_ratio,0.5,gear_ratio)"
?text	"Enable gearing when the Gearing_Button, GearRatioUp_Button or ..." *(shortened)*

The identification of the file where the pattern was found is returned as well as the full description of the instruction and the comment of the rung.

The advantage of one storage of all RDF data is that these data are interconnected and so the queries can not only search for a single file, but also for a pattern that would connect multiple file even of different formats. Let us illustrate this feature on a query combining RSLogix and RSView files: *Find which RSLogix projects have a tag used in a Gauge control on any RSView display.*

```
PREFIX rock: <http://rockwell.com/NS/Generic.owl#>
PREFIX logix: <http://rockwell.com/NS/RSLogix.owl#>
PREFIX view: <http://rockwell.com/NS/RSView.owl#>
SELECT ?exp ?gfxcomputername ?gfxfilename
       ?acdcomputername ?acdfilename
WHERE
{
    ?g a view:Gauge. #find gauge
    ?g rock:contains ?conn.
    ?conn view:hasExpression ?exp. #and the tag
    ?gfx a rock:File.
    ?gfx rock:contains ?g. #the file it is in
    ?gfx rock:hasComputerName ?gfxcomputername.
    ?gfx rock:hasFileName ?gfxfilename.
```

[5]The table returned has variables as column headers and values in the columns, however the layout would not fit to this text, so we rotated the table.

```
    ?tag a logix:Tag. #now try to find match in RSLogix
    ?tag logix:hasName ?exp. #is there tag with this name?
    ?acd a rock:File. # if so, which file it is in?
    ?acd rock:contains ?tag.
    ?acd rock:hasComputerName ?acdcomputername.
    ?acd rock:hasFileName ?acdfilename.
}
```

A sample result follows. The result contains tag name and the two files that contain the tag — RSView display file and RSLogix file:

Name	Value
?exp	"Fan2Amps"
?gfxcomputername	"euczprgmobitko4"
?gfxfilename	"C:\\tmp\\test\\view\\FanRoom.gfx"
?acdcomputername	"euczprgmobitko4"
?acdfilename	"C:\\tmp\\test\\logix\\PumpRTD.acd"

9.4.1 Translation for User Queries

The RDF files produced by the conversion from XML may use names of classes or properties that are different from those that one would expect based on the user interface of the software producing XML (i.e., RSLogix or RSView in our case). For users that are used to the software it may be then difficult to formulate queries using different vocabulary. An example was shown in the query *In which rung the constant 0.5 is used in the SUB instruction as "Source B"?* where the *"Source B"* known from user interface corresponds to `Parameter` with `hasNumber=1` (see the SPARQL query).

To allow users to use the vocabulary that they are used to a translation from the exported form to the form expected from the software user interface is needed. Based on the evaluation of the translation methods (in the section 8.2.5) and on the fact that including the original form as well as the translation form in one place is beneficial for us, we have chosen the method of OWL reasoning.

The translation to the vocabulary expected by the user is included in ontology that is used for reasoning over the exported form to produce the form that is queried by the user. For example for the *"Source B"* parameter of the instruction, the axiom

$$\textsc{SourceB} \equiv \exists \mathsf{hasNumber}.\{"1"\} \sqcap \textsc{Parameter}$$

is part of the ontology so that user can use directly the name *Source B* instead of saying "Parameter that has number 1".

The fragment of the SPARQL query is then as follows (the part of the original query is commented using #):

```
...
?ins a logix:Instruction. # look for instruction
?ins logix:hasName "SUB".
?ins rock:contains ?param.
# ?param a logix:Parameter. (don't need this)
# ?param logix:hasNumber "1". (don't need this)
?param a logix:SourceB.
?param logix:hasValue "0.5".
...
```

Again, as we have already noted in 7.4.2, maintaining the ontology is easier for us than maintaining a program that would change the names during export from XML.

9.5 Performance

In order to use the knowledge expressed in the ontology, inference must be made over all the information from all files. In principle, there are two possibilities of when such reasoning can take place:

- "Offline" reasoning — reasoning is made during the time of adding information to RDF storage. Whenever new file information is obtained, a deductive transitive closure of the A-Box (data) based on the corresponding T-Box (ontology) is computed and the whole transitive closure of deductions (i.e., all deductions entailed by the data and ontology) is stored for querying. When a query is evaluated against RDF storage, no inference has to be performed during querying.

- "Online" reasoning — reasoning is performed during the time of querying. The file information is stored to the RDF storage as converted from XML, without any inference. During querying, the query is evaluated against reasoner that performs reasoning over the whole RDF storage. In practice, the first query triggers initialization of the reasoner that will precompute the deductions. The reasoner needs to be continuously updated with new data.

For reasoning implementation, we have evaluated the following possibilities for computing deductions:

- Programmatic — the "reasoning" was implemented in a program, i.e., the program uses Jena API to check for occurrence of patterns and adds new triples if needed to compute the deductive closure based on a simple ontology. A simple implementation is used — whenever new triple is added, it is checked whether it would trigger addition of any other new triples. The description of reasoning information is the program code.

- Jena reasoner — Jena contains its own custom reasoner that uses mixture of forward and backward chaining[6]. Jena reasoner is not claimed to be complete, but as we verified, it produces all the conclusions for the purposes of this test. The description of reasoning information is OWL ontology that is loaded into reasoner.

- Pellet reasoner — OWL DL reasoner based on the tableau algorithm; used as embedded to Jena, due to DIG interface problems. The description of reasoning information is OWL ontology that is loaded into reasoner.

- Jena Rules — rules written for a general purpose Jena rule engine to support inferencing contained in an ontology. The description is in the form of Jena rules, such as

```
[ruleMakeTransitivePropertyTransitive:
    (?tp rdf:type owl:TransitiveProperty)
    (?a ?tp ?b) (?b ?tp ?c)   ->   (?a ?tp ?c) ]
```

The performance measurements for offline reasoning for these possibilities are summarized in the table[7] 9.4 and in the graphs in the figure 9.5. The first row of the table 9.4 is the number of triples that were input for reasoning; these triples were generated from RSView and RSLogix. Each column corresponds to one file of different size. As we can see from the table, the general purpose Jena rule engine is not very fast, which is not surprising due to its generality. Also, as we can see from the graph, its applicability is limited to the small number of triples — then the time is not acceptable

[6]Jena provides multiple reasoners with different level of reasoning support; we used the one obtained from `OWLFBRuleReasonerFactory`.

[7]Missing values in tables 9.4 and 9.5 mean that the process run out of 1GB memory space.

Num. of triples	6430	8083	10120	13094	17518	105611	143461
XML to RDF [s]	0.34	0.43	0.53	0.67	0.92	6.7	8.5
Program [ms]	50	70	80	160	210	600	1800
Jena [ms]	0.11	0.11	0.11	0.11	0.11	0.11	0.11
Pellet [ms]	0.19	0.19	0.19	0.19	0.19	0.20	0.21
Rules [s]	14	36	57	100	250	-	-

Table 9.4: Comparison of offline reasoning time — number of triples, conversion to RDF, and reasoning using naive program, Jena reasoner, Pellet and Jena rules.

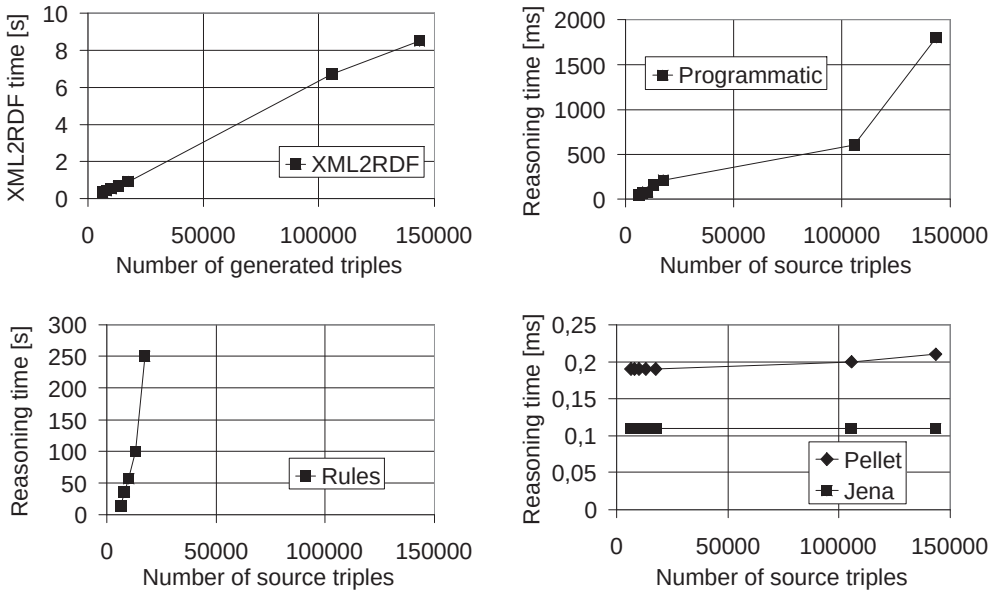

Figure 9.5: Time for XML to RDF conversion, and comparison of offline reasoning methods.

and it also runs out of memory. The naive implementation in a program is slow when compared with reasoners, however, it is still acceptable for one file (i.e., relatively small number of triples). Both reasoners, Jena and Pellet are fast enough. Note that most of the time is spent on conversion from XML to RDF (i.e., producing triples) — it is in the order of seconds, while reasoning is in the order of tenths of milliseconds.

The offline reasoning operates on individual files, and so it does not operate on a large quantity of triples. The situation is different with online

Number of millions of triples	0.1	0.25	0.5	1
Jena reasoner [s]	5.0	10	21	-
Pellet reasoner [s]	2.6	6.7	17	45

Table 9.5: Comparison of online reasoning time — number of millions of triples, reasoning using Jena reasoner and Pellet reasoner

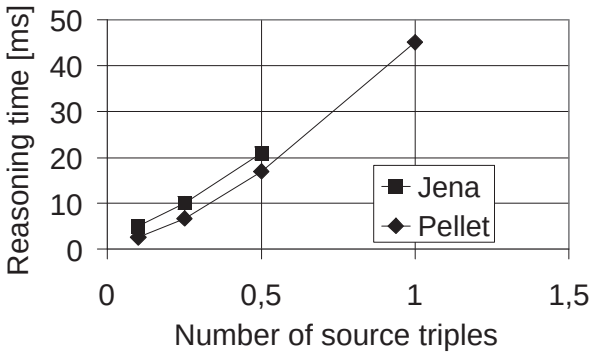

Figure 9.6: Comparison of online reasoning time.

reasoning, where the whole RDF storage content is loaded from a relational database. Because of the results for online reasoning, we evaluate only the Jena reasoner and Pellet reasoner in the case of online reasoning — see the summary in the table 9.5 and in the graph 9.6. The performance for the same number of triples differs from the offline reasoning because of the different access to triples. From the table we can see that Pellet with its tableau reasoning is comparable with the rule-based Jena reasoner for a smaller number of triples. However, the Jena reasoner runs out of memory for one million of triples, while Pellet is able to handle this number of triples in a reasonable time.

When we compare the reasoning times, we can conclude that there is no major difference between online and offline reasoning, as we expected — if the reasoning and RDF storage is kept in memory.

The real difference is in serialization time — the time of adding triples to database is significant when compared to reasoning (in this thesis, we are interested in reasoning only, so we will not discuss different relational database engines in detail). The offline reasoning spends more time during storing and later deleting triples, however all queries are answered from the database without any other initialization. The online reasoning does not spend

so much time in the operations with database; however, long initialization is needed before the first query is asked. Also, for online reasoning as we measured it, the whole RDF graph needs to be in memory, and cannot be in database as in the offline reasoning.

Part IV

Discussion and Conclusion

Chapter 10

Discussion and Conclusion

In this chapter, we first summarize the content of the thesis, discuss our work, and compare it with relevant work. Then we propose future research directions and summarize thesis main contributions.

10.1 Summary and Discussion

After introduction and motivation for the dissertation topic and goals we have summarized the state of the art relevant to the dissertation.

Even when ontologies help to specify conceptualization, the definition of the word ontology is still debated. For the purposes of this dissertation, the definition "formal explicit specification of conceptualization" is sufficient. "Formal" means expressed in a formal way, like a program in programming language (not in natural language). "Explicit specification" means that ontology is written down and that it is not enough to use "implicit" specification by a software code as it is often made in software systems, including multi-agent systems. "Conceptualization" is a view of the world, where we distinguish concepts, their properties, relations, etc. A conceptualization of a domain can have different ranges or levels of detail, depending on the application. It is important to note that systems can express only things that can be expressed in an ontology (either in a knowledge base internal for the system or for communication between systems).

We have discussed formal representation of ontologies with the focus on description logics. Description logics have their root in formalization of frames and semantic networks. DLs are important because they offer relatively rich constructs and still maintain good computational properties, such as decidability in an acceptable time. We have summarized operations on ontologies together with relationships between ontologies. We have summa-

rized how ontologies can be authored in a modular way and what is their classification from this point of view.

From the overview of ontologies it follows that description logics are a suitable formalism for expressing ontologies. We have summarized the semantic web technologies, such as Resource Description Format RDF and additional vocabulary RDF Schema, and have summarized how description logic is embedded into RDF to form the Web Ontology Language OWL. We have briefly discussed query language SPARQL and the possibilities of reasoning using OWL.

Another topic important for this thesis is the field of multi-agent systems. We have made an overview of the FIPA specifications, in particular with regard to communication in multi-agent systems. We have summarized FIPA ontology related proposals, primarily the unimplemented proposal of a specialized ontology agent that should help other agents with the ontology related tasks.

10.1.1 Semantic Web Technologies in Multi-Agent Systems

In the first third of the description of our work we have described the possibilities of usage of semantic web technologies in multi-agent systems. We have summarized the differences between FIPA proposals and Semantic Web technologies from which it is clear that the RDF and OWL languages can be used for communication between agents with many advantages. We have shown specific way how the semantic web languages can be used in multi-agent system, including an OWL ontology that can be used for expressing actions, which is important for communication of FIPA agents.

The idea of using semantic web approaches in non-semantic web agents was also presented on a multi-agent system operating in the trading domain [94] where agents communicate using only RDF/OWL. In [94] it is demonstrated how OWL is suitable for expressing information that needs to be exchanged between trading agents. Also, one of the FIPA specifications [43] proposes a way how to use RDF for communication between agents. In a custom Jade codec this proposal is implemented as a serialization of SL expressions in RDF/XML. Our proposal uses the full potential of OWL and adds ontologies that can be used as a base for communication and that can employ the potential of OWL reasoning. An approach that uses the potential of OWL reasoning for agent communication is described in [64] where an event based model is used for the common ontology. However, the ontology mixes FIPA ACL communication acts and message content and requires

agents to conform to the event based model. Our approach is not putting additional constraints over the original communication based on SL. The semantic integration and description of services is solved also within Semantic Web Services research, such as in the OWL-S project [35]. Ontologies developed within this project are comparable to our OWL ontology for describing FIPA agents; however, our ontology is more influenced by FIPA agents and concepts rather than Web Services.

We have also shown the implementation of ontology agent proposal [42] using the semantic web languages. An ontology that this agent accepts is described in OWL using the generic agent ontology described above. An important functionality of the agent is translation between ontologies (summarized below). We have shown how the agent operates on a sample scenario from the FIPA proposal. A partial implementations of this agent using the OKBC model is presented in [89]. The agent supports assert and retract actions, but the translation is not supported, which is not surprising for the OKBC model. Another relevant implementation is described in [59], which is not directly implementation of the FIPA proposal, but is an implementation of an agent close to the FIPA ontology service idea. It is an agent for ontology storing with the emphasis on consistency checking using a special custom algorithm (which is in our opinion not needed if any OWL reasoner is used). Again, translation is not supported.

We have also paid attention to the group of agents that use XML for communication. The XML language is popular as a syntactic interlingua; many systems use it for export and import of information, and so it is important to be able to use this information as easily as possible in our semantic web based agents. We have proposed a mapping wizard that helps with generation of ontology and mapping from a set of XML documents. During the run of the wizard, human can change the default settings that are proposed by heuristics. The exported ontology can be then enriched manually, but the mapping can be reused for further automatic conversion of XML documents. This way is more flexible than giving special meaning to XML or than using fixed translation algorithm as summarized in [71]. We note that a standard for conversion from XML to RDF is evolving within W3C activities [38] that is primarily intended for XHTML web pages and proposes to use XSLT for the conversion.

10.1.2 Translation Between Ontologies

The second third of our work description deals with translation between ontologies. When two agents operate on the same domain but use different ontologies then a translation between these two ontologies is needed. All of

the messages exchanged by these agents need to be translated between these two ontologies and we describe a specific way of doing that.

We have show an example of three transportation domain ontologies where similar information is expressed in different way. On this example we have illustrated translation using Jena rules, using rules expressed as SPARQL `CONSTRUCT` query and using OWL mapping information. It is shown how to capture the translation in these three methods and how the translation knowledge is used to translate information between ontologies. We have discussed features of these methods as well as their performance with the conclusion that Jena rules and OWL reasoning is usable. The difference is whether it matters that the original information is merged with the translated information which will happen with the OWL reasoning method.

We then show how the translation is used within the ontology agent. Translation is an important feature of the ontology agent for helping other agents with communication. No other implementation of the ontology agent that is known to us contains this feature.

We also discuss architecture of embedding translation into multi-agent system. Three options are discussed — agents handle the translation themselves, agents use specialized ontology agent, and proposed transparent translation. Transparent translation is a a way of integration of agents that are not aware of ontologies (i.e., they have only "implicit" specification of conceptualization and do not know that other agents may use other ontologies) without changing them. Such a translation is made directly in MAS platform provided that the platform has information about what ontology is accepted by an agent — this information can be obtained from directory facilitator.

There are algorithms proposed for (semi-)automatic finding of alignment of ontologies, summarized for example in [39], but we are not aware of any evaluation of actually translating messages between ontologies during agent communication using semantic web languages. The usage of SPARQL as a rule language has been proposed in [84] — we used this idea to implement the rules for translation and compared it to our proposal using Jena rules.

10.1.3 Application: Semantic Search in Manufacturing Industry

The usage of semantic web technologies as described above is illustrated on an application of semantic search in manufacturing industry. The design and maintenance of assembly lines requires handling of many files describing control code and human-machine interface. Searching often means opening all individual files and even within an opened file search might not be easy.

We show an application of the semantic web enabled agents as described above to enable semantic search.

The content of the files is converted to RDF/OWL using our mapping wizard and file metadata are added. The information is stored by an agent that is an extension of the ontology agent. This RDF storage agent accepts new information, makes reasoning, and answers queries. The operation is illustrated on sample queries including a query that connects more files of different format together.

We have also evaluated performance of reasoning using different methods and it was shown that the best option is to use Pellet DL reasoner.

This application is close to the idea of semantic desktop [87, 85] that is intended to manage personal data like e-mails, calendar, together with the files like text documents or spreadsheets. In [85] so called Personal Information Model is used to annotate (via tags) all the information and to search according to these annotations.

In the approach we describe we are primarily interested in a semantic search without any need to annotate anything manually. Our intention is to make the information in the files explicit and connected including background knowledge so that search can be performed. However, in the future we may incorporate the annotation support as well.

10.2 Future Work

In the dissertation we have discussed the translation between ontologies, but except the help of the wizard providing mapping from XML to RDF/OWL, we have always assumed that the mapping between ontologies is provided externally, typically by a human. An important research question is to what degree it is possible to derive the mapping automatically in multi-agent systems.

We have already shown a way of finding mappings by a pointing game in a shared domain in [72], but a proper implementation in the framework described in this dissertation is still remaining for future work.

During the description of translation, we have shown a disadvantages of OWL reasoning for translation. However, the representation of translation between OWL ontologies is best encoded directly in OWL. We would like to investigate how this information can be used for translation without the disadvantage of leaving transitive closure and the source information. One of the possibilities would be translation to rules.

Also, we have made all the rule-based experiments and descriptions using Jena rules, but we would like to investigate the usage of SWRL [55] and RIF

[17] rules for the purposes of translation for agent communication as well.

We have presented our ontology for the description of agents and actions that is based on FIPA specifications. We would like to compare it in detail to Semantic Web Services research, such as to the OWL-S project [35], to see whether the Semantic Web Services approach can be used for an integration of FIPA-based agents used in industrial applications.

10.3 Dissertation Contributions and Goals Evaluation

The main contributions of this dissertation can be summarized as follows. In the summary we will follow the goals as set in the beginning of the dissertation.

- *Investigate and propose possibilities of using the results of research from the semantic web area in the area of multi-agent systems in order to help multi-agent systems to overcome the problems with ontologies that they face.*
  We have proposed how to use semantic web technologies (see the figure 5.1) in FIPA compliant multi-agent system (see the figure 7.1). The comparison with the current specifications has been shown (in the table 7.1). We have shown that the proposal solves some of the problems that were not resolved when using original FIPA proposals, such as usage of automated reasoning and especially the communication in heterogeneous systems. In addition, we have studied the possibilities of reusing existing agent implementations that use XML as a syntax for information interchange. We have proposed a wizard that suggests mapping between XML and OWL using simple heuristics and allows to change these suggestions (see the section 7.4). In this way, the information expressed in XML can be semantically enriched so that the OWL advantages can be employed. This goal has been attained.

- *Investigate, propose and evaluate ways of translation between different ontologies, especially for agent communication, using semantic web languages, when the translation information would be expressed in declarative form.*
  We have investigated the support of translation between ontologies between semantic web languages on the example of three transportation ontologies. We have shown three ways of expressing the translation information in a declarative form — rules (Jena syntax), SPARQL

`CONSTRUCT` queries, and relationships expressed in OWL (see the section 8.2). We have compared the features of these translations, including performance (see the section 8.2.5). This goal has been accomplished.

- *Investigate the possibility of implementation of the ontology agent that has been proposed by FIPA using semantic web technologies.*
  We have shown an implementation of this ontology agent using semantic web technologies (see the section 7.3). Again, we conclude that the usage of semantic web technologies has been very beneficial over the originally proposed SL and OKBC languages. Unlike other available implementations, this agent has a capability to help other agents with translation between ontologies (see the section 8.3), which is one of the most important features for the integration of multi-agent systems. In addition, we have studied the architecture of using translation in multi-agent systems (see the section 8.4) and proposed and evaluated transparent translation embedded in the multi-agent platform that enables to integrate even agents that are not aware of ontology translation during communication in a heterogeneous environment. This goal has been attained.

- *Show and evaluate a manufacturing domain application of the semantic web technologies enabled agents.*
  We have illustrated the usage of semantic web enabled agents on an application of semantic search in files produced by software needed for assembly line design, operation, and maintenance (see the chapter 9). For this application, we have reused the framework described above including the functionality of the ontology agent and including the XML to OWL mapping wizard. On sample queries we have shown that the OWL language (together with ontologies that we developed and that are used in the storage agent) is able to grasp the semantics needed for efficient search and is able to connect structured information for search (see the section 9.4). We have also evaluated the efficiency of reasoning. This goal has been accomplished.

10.4 Conclusion

We have described the usage of ontologies in multi-agent systems and analyzed the FIPA proposals with the emphasis on the proposed ontology service agent. We have compared the proposals to Semantic Web technologies and shown that these technologies are suitable for multi-agent systems. We

have discussed ways of translation between ontologies and the architecture of translation.

The ability of handling multiple ontologies and translating between them is a necessary condition for the operation of open heterogeneous multi-agent systems. A specialized ontology agent could help to connect agents without the capabilities to make translations capabilities, and the transparent translation can integrate any agents, even those that are not aware of existence of multiple ontologies.

We have illustrated the usage of the semantic web enabled agents on an application of semantic search within files of applications that are needed to operate an automated assembly line. We have shown that the capabilities of the agents are suitable for such an application.

The work described in the thesis enables integrating agents that use different ontologies which is a necessary requirement for larger scale integration, such as for virtual enterprises, where whole companies of different kinds (and so using different ontologies) are involved in one process.

Bibliography

[1] Common Logic Working Group Documents. http://cl.tamu.edu/.

[2] DL Implementation Group Interface (DIG). http://sourceforge.net/projects/dig.

[3] FaCT++ — OWL DL Reasoner. http://owl.man.ac.uk/factplusplus/.

[4] *FIPA — Foundation for Intelligent Physical Agents Website.* http://www.fipa.org/.

[5] *JADE — Java Agent DEvelopment Framework Homepage.* http://jade.tilab.com/.

[6] Pellet: An OWL DL Reasoner. http://pellet.owldl.com/.

[7] Racer — Renamed ABox and Concept Expression Reasoner. http://www.racer-systems.com/products/racerpro/index.phtml.

[8] SUMO Ontology. http://ontology.teknowledge.com/.

[9] UMBC KQML Homepage. http://www.cs.umbc.edu/kqml/.

[10] Unicode Standard Resources. http://unicode.org/.

[11] W3C Semantic Web Activity Home Page. http://www.w3.org/2001/sw/.

[12] XML Schema Specifications. http://www.w3.org/XML/Schema#dev.

[13] *WordNet - a Lexical Database for English*, 1996. http://www.cogsci.princeton.edu/~wn/.

[14] OWL Web Ontology Language Overview, W3C Recommendation, 2 2004. http://www.w3.org/TR/owl-features/.

[15] *Information technology — Common Logic (CL) — A framework for a family of logic-based languages, ISO/FDIS 24707:2006(E)*, 2006. `http://cl.tamu.edu/docs/cl/24707-31-Dec-2006.pdf`.

[16] *Jena — A Semantic Web Framework for Java*, 2007. `http://jena.sourceforge.net/`.

[17] Rule Interchange Format (RIF) W3C Working Group Documents, 2007. `http://www.w3.org/2005/rules/`.

[18] Petr Aubrecht. *Ontology Transformations Between Formalisms*. PhD thesis, Czech Technical University in Prague, 2005.

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

[20] Dave Beckett. RDF/XML Syntax Specification (Revised), W3C Recommendation, 2 2004. `http://www.w3.org/TR/rdf-syntax-grammar/`.

[21] Dave Beckett. Turtle — Terse RDF Triple Language, 2006. `http://www.dajobe.org/2004/01/turtle/`.

[22] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. Developing Multi-agent Systems with JADE. In *Intelligent Agents VII. Agent Theories Architectures and Languages, 7th International Workshop*, pages 89–103, 2000.

[23] Tim Berners-Lee. Reification of RDF and N3 — Design Issues, 2005. `http://www.w3.org/DesignIssues/Reify.html`.

[24] Tim Berners-Lee. Notation 3 - Readable language for data on the Web, 2006. `http://www.w3.org/DesignIssues/Notation3`.

[25] Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. Uniform Resource Identifier (URI): Generic Syntax, 2005. `http://www.ietf.org/rfc/rfc3986.txt`.

[26] Paul V. Biron and Ashok Malhotra. XML Schema Part 2: Datatypes Second Edition, W3C Recommendation, 2004. `http://www.w3.org/TR/xmlschema-2/`.

[27] Tim Bray, Dave Hollander, Andrew Layman, and Richard Tobin. Namespaces in XML 1.0 (Second Edition). `http://www.w3.org/TR/REC-xml-names/`.

[28] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and Francois Yergeau. Extensible Markup Language (XML) 1.0 (Fourth Edition), 2006. `http://www.w3.org/TR/REC-xml/`.

[29] Dan Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation, 2004. `http://www.w3.org/TR/rdf-schema/`.

[30] Dan Brickley and Libby Miller. FOAF Vocabulary Specification, 2005. `http://xmlns.com/foaf/0.1/`.

[31] Jeremy J. Carroll, Christian Bizer, Pat Hayes, and Patrick Stickler. Named graphs. *J. Web Sem.*, 3(4):247–267, 2005.

[32] Jeremy J. Carroll, Christian Bizer, Patrick J. Hayes, and Patrick Stickler. Named graphs, provenance and trust. In *Proceedings of the 14th international conference on World Wide Web (WWW 2005)*, pages 613–622, 2005.

[33] Vinay K. Chaudhri, Adam Farquhar, Richard Fikes, Peter D. Karp, and James P. Rice. Open Knowledge Base Connectivity 2.0.3. http://www.ai.sri.com/ okbc/spec.html, 1998.

[34] P. Clark. Transportation Ontology, 1998. `http://www.cs.utexas.edu/users/pclark/kr-web/other/passenger-vehicle/transportation.km`.

[35] The OWL Services Coalition. OWL-S: Semantic Markup for Web Services., 2004. `http://www.daml.org/services/owl-s/1.0/`.

[36] Cycorp. CYC homepage. `http://cyc.com/`.

[37] Cycorp. OpenCYC homepage. `http://opencyc.org/`.

[38] Ian Davis and Talis. Gleaning Resource Descriptions from Dialects of Languages (GRDDL) Primer.

[39] Marc Ehrig and Jerome Euzenat. State of the art on ontology alignment — Knowledge Web Deliverable 2.2.3. Technical report, University of Karlsruhe, 2004.

[40] Dieter Fensel, Ian Horrocks, Frank van Harmelen, Stefan Decker, Michael Erdmann, and Michel C. A. Klein. OIL in a Nutshell. In *Proceedings of the European Workshop on Knowledge Acquisition, Modeling, and Management (EKAW)*, pages 1–16, 2000.

[41] Tim Finin, Don McKay, Rich Fritzson, and Robin McEntire. KQML: An Information and Knowledge Exchange Protocol. In *Knowledge Building and Knowledge Sharing*. Ohmsha and IOS Press, 1994.

[42] FIPA. Ontology Service Specification, 2001. `http://www.fipa.org/specs/fipa00086/index.html`.

[43] FIPA. RDF Content Language Specification, 2001. `http://www.fipa.org/specs/fipa00011/`.

[44] FIPA. Agent Communication Language Message Structure Specification, 2002. `http://www.fipa.org/specs/fipa00061/index.html`.

[45] FIPA. SL Content Language Specification, 2002. `http://www.fipa.org/specs/fipa00008/index.html`.

[46] FIPA. Agent Management Specification, 2004. `http://www.fipa.org/specs/fipa00023/index.html`.

[47] Enrico Franconi. Description Logics — Tutorial Course Information. `http://www.inf.unibz.it/~franconi/dl/course/`.

[48] Michael R. Genesereth and Richard E. Fikes. Knowledge Interchange Format, Version 3.0, Reference Manual. `http://logic.stanford.edu/kif/Hypertext/kif-manual.html`, 1994.

[49] Thomas R. Gruber. What is an Ontology? `http://www-ksl.stanford.edu/kst/what-is-an-ontology.html`.

[50] Thomas R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, (2), 1993.

[51] Thomas R. Gruber. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. In Nicola Guarino and Roberto Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*. Kluwer Academic Publishers, 1993.

[52] Nicola Guarino. Formal Ontology and Information Systems. In *Proceedings of the 1st International Conference on Formal Ontologies in Information Systems, FOIS'98*, 1998.

[53] Nicola Guarino and Pierdaniele Giaretta. Ontologies and Knowledge Bases - Towards a Terminological Clarification. In N.J.I. Mars, editor, *Towards Very Large Knowledge Bases*. IOS Press, Amsterdam, 1995.

[54] Patrick Hayes. RDF Semantics, W3C Recommendation, 2 2004. `http://www.w3.org/TR/rdf-mt/`.

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

[56] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: the making of a Web Ontology Language. *J. Web Sem.*, 1(1):7–26, 2003.

[57] International Electrotechnical Commission. *IEC 61131-3 — Ed. 2.0 — English: Programmable controllers —- Part 3: Programming languages*, 2003. `http://webstore.iec.ch/webstore/webstore.nsf/artnum/029664`.

[58] Michel Klein. Combining and relating ontologies: and analysis of problem and solutions. In *Workshop on Ontologies and Information Sharing, IJCAI'01*, 2001.

[59] Yang Li, Simon G. Thompson, Zhu Tan, Nick Giles, and Hamid Gharib. Beyond Ontology Construction; Ontology Services as Online Knowledge Sharing Communities. In *International Semantic Web Conference*, pages 469–483, 2003.

[60] Stuart E. Madnick. From VLDB to VMLDB (Very MANY Large Data Bases): Dealing with Large-Scale Semantic Heterogeneity. In *Proceedings of the 21st VLDB Conference*, 1995.

[61] Frank Manola and Eric Miller. RDF Primer, W3C Recommendation, 2 2004. `http://www.w3.org/TR/rdf-primer/`.

[62] Vladimír Marík and Duncan C. McFarlane. Industrial Adoption of Agent-Based Technologies. *IEEE Intelligent Systems*, 20(1):27–35, 2005.

[63] Deborah L. McGuinness, Richard Fikes, James P. Rice, and Steve Wilder. The Chimaera Ontology Environment. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI2000)*, 2000.

[64] Zoltan Balogh Ladislav Hluchy Michal Laclavik, Marian Babik. AgentOWL: Semantic Knowledge Model and Agent Architecture. *Computing and Informatics*, 25:419–437, 2006.

[65] Riichiro Mizoguchi and Mitsuru Ikeda. Towards Ontology Engineering. Technical Report AI-TR-96-1, The Institute of Scientific and Industrial Research, Osaka University, 1996.

[66] Natalya F. Noy and Mark A. Musen. SMART: Automated Support for Ontology Merging and Alignment. In *Twelfth Banff Workshop on Knowledge Acquisition, Modeling, and Management*, 1999.

[67] Marek Obitko. Ontologies — Description and Applications. Technical Report 126, Gerstner Laboratory for Intelligent Decision Making and Control, Czech Technical University in Prague, 2001. `http://cyber.felk.cvut.cz/gerstner/reports/GL126.pdf`.

[68] Marek Obitko. Translating Between Ontologies For Agent Communication. Technical Report GL 149/02, Gerstner Laboratory for Intelligent Decision Making and Control, Czech Technical University in Prague, 2002. `http://cyber.felk.cvut.cz/gerstner/reports/GL149.pdf`. ISSN 1213-3000.

[69] Marek Obitko and Vladimír Mařík. Ontologies for Multi-Agent Systems in Manufacturing Domain. In *Thirteenth International Workshop on Database and Expert Systems Applications*, pages 597–602. Los Alamitos: IEEE Computer Society Press, 2002. ISBN 0-7695-1668-8.

[70] Marek Obitko and Vladimír Mařík. Ontologies in Multi-Agent Systems. *International Journal of The International Institute for Advanced Studies in Systems Research and Cybernetics*, 2(2):12–16, 2002. ISSN 1609-8625.

[71] Marek Obitko and Vladimír Mařík. Adding OWL Semantics to Ontologies Used in Multi-Agent Systems for Manufacturing. In *Holonic and Multi-Agent Systems for Manufacturing*, pages 189–200. Springer-Verlag, Berlin Heidelberg New York, 2003. ISBN 3-540-40751-0.

[72] Marek Obitko and Vladimír Mařík. Mapping between Ontologies in Agent Communication. In *Multi-Agent Systems and Applications III*, pages 191–203. Springer-Verlag, Berlin Heidelberg New York, 2003. ISBN 3-540-40450-3.

[73] Marek Obitko and Vladimír Mařík. Ontology Service Agent with Web Ontology Language. In *EUMAS 2004 - Proceedings*, pages 477–488, 2004.

[74] Marek Obitko and Vladimír Mařík. OWL Ontology Agent based on FIPA proposal. In *Znalosti 2004*, pages 226–237. Ostrava: VSB-TUO, 2004. ISBN 80-248-0456-5.

[75] Marek Obitko and Vladimír Mařík. Integrating Transportation Ontologies Using Semantic Web Languages. In *Holonic and Multi-Agent Systems for Manufacturing, Second International Conference on Industrial Applications, of Holonic and Multi-Agent Systems, HoloMAS 2005*, pages 99–110. Heidelberg: Springer, 2005.

[76] Marek Obitko and Vladimír Mařík. Integration of Multi-Agent Systems: Architectural Considerations. In *11th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA) Proceedings*, pages 1145–1148. IEEE, 2006.

[77] Marek Obitko and Vladimír Mařík. Transparent Ontological Integration of Multi-Agent Systems. In *2006 IEEE International Conference on Systems, Man, and Cybernetics*, pages 2488–2492. New York: IEEE - Systems, Man, and Cybernetics Society, 2006.

[78] Marek Obitko and Václav Snášel. Ontology Repository in Multi-Agent System. In *Artificial Intelligence and Applications - Volume I & II*, pages 853–858. Calgary: Acta Press, 2004. ISBN 0-88986-404-7.

[79] Daniel E. O'Leary. Different Firms, Different Ontologies, and No One Best Ontology. *IEEE Intelligent Systems*, 15(5):72–78, 2000.

[80] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax, W3C Recommendation, 2 2004. `http://www.w3.org/TR/owl-semantics/`.

[81] Steve Pepper and Graham Moore. *XML Topic Maps (XTM) 1.0*. TopicMaps.Org, 2001. `http://www.topicmaps.org/xtm/index.html`.

[82] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF, W3C Working Draft, 2006. `http://www.w3.org/TR/rdf-sparql-query/`.

[83] Alan L. Rector, Nick Drummond, Matthew Horridge, Jeremy Rogers, Holger Knublauch, Robert Stevens, Hai Wang, and Chris Wroe. OWL

Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns. In *Proceedings of the European Workshop on Knowledge Acquisition, Modeling, and Management (EKAW)*, pages 63–81, 2004.

[84] Dave Reynolds and Jeremy J. Carroll. Rule languages for interoperability: HP Position paper. In *W3C Workshop on Rule Languages for Interoperability*, 2005.

[85] Leo Sauermann, Gunnar Aastrand Grimnes, Malte Kiesel, Christiaan Fluit, Heiko Maus, Dominik Heim, Danish Nadeem, Benjamin Horak, and Andreas Dengel. Semantic Desktop 2.0: The Gnowsis Experience. In *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference*, volume 4273 of *Lecture Notes in Computer Science*, pages 887–900. Springer, 2006.

[86] John F. Sowa. *Knowledge Representation — Logical, Philosophical and Computational Foundations*. Brooks/Cole, 2000.

[87] Leo Sauermann Soren Auer Siegfried Handschuh Stefan Decker, Jack Park, editor. *Proceedings of the Semantic Desktop and Social Semantic Collaboration Workshop (SemDesk 2006), 5th International Semantic Web Conference ISWC 2006*. CEUR Workshop Proceedings, 2006.

[88] Patrick Stickler. CBD — Concise Bounded Description, W3C Member Submission, 2005. `http://www.w3.org/Submission/CBD/`.

[89] Hiroki Suguri, Eiichiro Kodama, Masatoshi Miyazaki, Hiroshi Nunokawa, and Shoichi Noguchi. Implementation of FIPA Ontology Service. In *Workshop on Ontologies in Agent Systems. 5th International Conference on Autonomous Agents*, 2001.

[90] Pavel Vrba. JAVA-Based Agent Platform Evaluation. In *1st International Conference on Industrial Applications of Holonic and Multi-Agent Systems (HoloMAS)*, pages 47–58, 2003.

[91] Pavel Vrba and Vaclav Hrdonka. Material Handling Problem: FIPA Compliant Agent Implementation. In *12th International Workshop on Database and Expert Systems Applications (DEXA 2001)*, pages 635–639. IEEE Computer Society, 2001.

[92] Pavel Vrba and Vaclav Hrdonka. Material Handling Problem: FIPA Compliant Agent Implementation. In *Multi-Agent-Systems and Applications II, 9th ECCAI-ACAI/EASSS 2001, AEMAS 2001, HoloMAS 2001, Selected Revised Papers*, volume 2322 of *Lecture Notes in Computer Science*, pages 268–279. Springer, 2001.

[93] David Whitten. The Unofficial, Unauthorized Cyc Frequently Asked Questions, 1997. `http://www.robotwisdom.com/ai/cycfaq.html`.

[94] Li Ding Harry Chen Youyong Zou, Tim Finin and Rong Pan. Using Semantic web technology in Multi-Agent systems: a case study in the TAGA Trading agent environment. In *Proceeding of the 5th International Conference on Electronic Commerce*, 2003.

[95] T. Zhang. Local Transport Ontology, 2003. `http://www.agentcities.org/EURTD/Ontologies/local-transport.v2.daml`.

[96] Evgeny Zolin. Complexity of reasoning in Description Logics, 2006. `http://www.cs.man.ac.uk/~ezolin/logic/complexity.html`.