Ontology-based Infrastructure for Intelligent Applications

Andreas Eberhart

Dissertation zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

Saarbrücken, 2004

Tag des Kolloquiums: 18.12.2003
Dekan: Prof. Dr. Slusallek
Vorsitzender: Prof. Dr. Andreas Zeller
Berichterstatter: Prof. Dr. Wolfgang Wahlster, Prof. Dr. Andreas Reuter

Für Hermann und Martin in liebevollem Gedenken

Abstract

Ontologien sind derzeit ein viel diskutiertes Thema in Bereichen wie Wissensmanagement oder Enterprise Application Integration. Diese Arbeit stellt dar, wie Ontologien als Infrastruktur zur Entwicklung neuartiger Applikationen verwendet werden können, die den User bei verschiedenen Arbeiten unterstützen. Aufbauend auf den im Rahmen des Semantischen Webs entstandenen Spezifikationen, werden drei wesentliche Beiträge geleistet. Zum einen stellen wir Inferenzmaschinen vor, die das Ausführen von deklarativ spezifizierter Applikationslogik erlauben, wobei besonderes Augenmerk auf die Skalierbarkeit gelegt wird. Zum anderen schlagen wir mehrere Lösungen zum Anschluss solcher Systeme an bestehende IT Infrastruktur vor. Dies beinhaltet den, unseres Wissens nach, ersten lauffähigen Prototyp der die beiden aufstrebenden Felder des Semantischen Webs und Web Services verbindet. Schließlich stellen wir einige intelligente Applikationen vor, die auf Ontologien basieren und somit großteils von Werkzeugen automatisch generiert werden können.

Abstract

Ontologies currently are a hot topic in the areas of knowledge management and enterprise application integration. In this thesis, we investigate how ontologies can also be used as an infrastructure for developing applications that intelligently support a user with various tasks. Based on recent developments in the area of the Semantic Web, we provide three major contributions. We introduce inference engines, which allow the execution of business logic that is specified in a declarative way, while putting strong emphasis on scalability and ease of use. Secondly, we suggest various solutions for interfacing applications that are developed under this new paradigm with existing IT infrastructure. This includes the first running solution, to our knowledge, for combining the emerging areas of the Semantic Web and Web Services. Finally, we introduce a set of intelligent applications, which is built on top of ontologies and Semantic Web standards, providing a proof of concept that the engineering effort can largely be based on standard components.

Abstract

Ontologien sind derzeit ein viel diskutiertes Thema in Bereichen wie Wissensmanagement oder Enterprise Application Integration. Diese Arbeit stellt dar, wie Ontologien als Infrastruktur zur Entwicklung neuartiger Applikationen verwendet werden können, die den User bei verschiedenen Arbeiten unterstützen. Hierbei werden die folgenden wesentlichen Fragestellungen beantwortet.

1. Wie viel und welche Art von semantisch gekennzeichneten Daten sind im Web derzeit verfügbar?

   Wir suchen das Web mittels vier verschiedener Methoden nach RDF ab. Die gefundenen RDF Fakten werden aufgrund des Prädikatnamespaces und dem Fundort analysiert. Die Resultate zeigen, dass RDF noch nicht von einer großen Nutzergruppe verwendet wird. Dennoch sind viel versprechende Anzeichen wie die RDF Unterstützung der Adobe Werkzeuge zu erkennen.

2. Können Regeln und Ontologien in eine etablierte IT Landschaft aus Datenbank und objektorientierten Programmiersprachen integriert werden?

   Wir präsentieren die OntoJava und OntoSQL Inferenzmaschinen, die sich auf bewährte Technologien wie Java und relationale Datenbanken stützten. Dabei werden RDF Schema Ontologien und RuleML in Java beziehungsweise SQL übersetzt. Trotz einiger inhärenter Einschränkungen bewährten sich diese Werkzeuge als Basis für die Entwicklung weiterer Applikationen. Leistungstest zeigen, dass Datenbanken durchaus mit traditionellen regelbasierten Inferenzmaschinen wie XSB mithalten können.

3. Können Ontologien und Ereignis Bedingung Aktionsregeln zur Spezifikation des Verhaltens von Agenten verwendet werden?

   Die deklarative Spezifikation von Agenten ist ein sehr viel versprechender Ansatz, da er den Entwickler von vielen der Lasten befreit, die die Programmierung von Agenten normalerweise mit sich bringt. Mit OntoAgent stellen wir eine Plattform vor, die es erlaubt einen Agenten komplett mit Standards des Semantischen Webs, insbesondere RDF, RDF Schema und RuleML, zu spezifizieren. Die Implementierung basiert auf OntoSQL und einigen Modulen zur Realisierung von Reaktionsregeln, einer Kommandobibliothek und einer Kommunikationseinheit. Wir diskutieren ferner etliche der getroffenen Designentscheidungen.

4. Ist es möglich semantisch gekennzeichnete Dienste ad hoc aufzurufen?

   Wir präsentieren WSDF, eine semantische Annotation von Web Services basierend auf WSDL, RDF Schema und RuleML. Mit Hilfe eines WSDL Kompilers und unseres OntoSQL RuleML-SQL Konverters wird ein Klient in die Lage versetzt, die generische Beschreibung des Dienstes in lauffähigen Code umzusetzen. Dadurch ist lediglich eine gemeinsame Ontologie die Voraussetzung für das Aufrufen des Dienstes. Obwohl wir uns auf die einfachste Form, nämlich Dienste ohne Seiteneffekte, beschränken ist dies ein wesentlicher Schritt nach vorn, da bisher immer explizit gegen die Schnittstelle programmiert werden musste oder ein vorab bekanntes UDDI tModel implementiert sein musste. Verglichen mit anderen Ansätzen wie DAML-S oder WSMF bieten wir nicht nur die Sprache, sondern erstmals auch eine funktionierende Laufzeitumgebung die es dem Klienten erlaubt generische Dienste aufzurufen und die Ergebnisse korrekt zu interpretieren.

5. Ist es möglich linguistische Funktionen mittels derselben Mechanismen zu verwirklichen, die auch zur Einbindung von anderen Informationsquellen verwendet werden?

   Die OntoLang und SmartDialog Sprachsysteme demonstrieren diese Möglichkeit. OntoLang verwendet linguistische Annotationen um einfache Sätze zu verstehen und die darin enthaltenen Fakten in die RDF Wissensbasis einzufügen. Dadurch ist OntoLang in der Lage über die Inferenzmaschine mit beliebigen anderen Applikationen zu interagieren.

6. Bietet die einer Software zugrunde liegende Ontologie eine ausreichende Basis um verschiedenen Komponenten eine Kooperation zu ermöglichen?

   Angenommen die Ontologie ist sowohl in dem Datenmodell der Faktenbasis als auch in der Inferenz- und der Anfrageschicht vertreten, so zeigen wir, dass Applikationsteile über diese Schnittstelle kooperieren können. Das gemeinsame Datenmodell fungiert hierbei als Mediator.

7. Sind Ontologien bei der Entwicklung von komplexen Applikationen hilfreich?

   Wir stellen die Tragfähigkeit unserer Konzepte dadurch unter Beweise, indem wir einige Beispielapplikationen mit unseren eigenen Werkzeugen entwickeln. SmartGuide und SmartDialog demonstrieren hierbei wie eine komplexe Document Retrieval Anwendung die den Nutzerkontext mit einbezieht und klärende Fragen stellt mit vertretbarem Aufwand implementiert werden kann.

Abstract

Ontologies currently are a hot topic in the areas of knowledge management and enterprise application integration. In this thesis, we investigate how ontologies can also be used as an infrastructure for developing applications that intelligently support a user with various tasks. In particular, the following key questions are answered.

1. How much and what kind of semantically tagged data is available on the Web today?

   We crawled the Web for RDF data using four different search strategies. The RDF facts were analyzed according to their predicate namespaces and their location. The results show that RDF has not yet caught on with a large user community. However, there are promising signs such as Adobe supporting RDF in its tools.

2. Can rules and ontology be integrated into a mainstream IT infrastructure of databases and object oriented programming languages?

   We present the OntoJava and OntoSQL inference engines which leverage proven technologies such as Java and relational databases by translating RDF Schema ontologies and RuleML into Java and SQL respectively. Despite some inherent limitations on the rule expressiveness, the tools prove to be a valuable backbone for application development. Performance measurements show that SQL databases can very well keep up with traditional rule engines such as XSB.

3. Can ontologies and event condition action rules be used to define the behavioral aspects of agents?

   Declaratively specifying agents is an extremely promising approach since it relieves programmers from many of the burdens that are usually inherent with the implementation of agent systems. We present OntoAgent, a framework, which builds on Boley et. al's idea of specifying an agent entirely using the Semantic Web mark-up languages RDF, RDF Schema, and RuleML. The implementation is based on OntoSQL and a set of add-ons for realizing reaction rules, a command library, and a messaging subsystem based on HTTP. Several design choices and trade-offs are discussed.

4. Is it possible to invoke a semantically tagged service on the fly?

   We present WSDF, a semantic annotation of Web Services based on the existing languages WSDL, RDFS, and RuleML. Using a WSDL compiler and our OntoSQL RuleML to SQL converter, a client application is able to process the generic descriptions into running code. Web Services can therefore be invoked simply by agreeing on an RDFS ontology a priori. Even though we restrict ourselves to the simplest case of methods without side effects, this is a major step forward, since traditionally programmers must either be aware of a certain UDDI tModel or simply read a textual description and program corresponding client code. Compared to other approaches like WSMF or DAML-S, we provide a complete solution that not only specifies service mark-up, but for the first time also specifies how the client has to interpret the results.

5. Is it possible to base linguistic features on the same mechanisms that are used for obtaining data from information sources other than the user?

   The OntoLang and SmartDialog language systems demonstrate this possibility. OntoLang uses linguistic annotations to parse and understand simple natural language statements, which are fed into the RDF fact base. This way, OntoLang is seamlessly integrated with the inference engine and other applications that also access the inference component. SmartDialog leverages the knowledge base to predict how answers to potential questions affect the pool of possible documents to be suggested.

6. Does an underlying ontology provide the necessary foundation for various system components to interoperate seamlessly?

   Assuming that the ontology is represented in both the facts' data model and the query and inference layer, we show that all applications can be interfaced via this foundation. The mediator is the shared knowledge and data model.

7. Are ontologies a useful help for developing a complex application?

   By developing an entire suite of sample applications, we provided a proof of concept that an ontology-based infrastructure is useful for developing intelligent applications. SmartGuide and SmartDialog demonstrate how a complicated document retrieval application, which takes the user's current context into account and asks clarification questions, can be built with relatively little engineering effort.

Contents

Chapter 1

Introduction

Information and communication technologies are increasingly penetrating almost every aspect of our everyday lives. Many examples for this trend can be given: digital photography is about to replace celluloid pictures, packages can be tracked via the Internet, and enterprise information systems are gaining importance also for smaller corporations. This development opens up enormous opportunities. However, much of this potential has not yet been fully realized. In our opinion many problems arise from a general lack of data integration. Massive amounts of data are available but usually, it is virtually impossible to cross-link the various data sources. Consider a travel reservation system, which feeds off a large database containing schedule and booking information. While such a service provides a lot of value to customers, quite often they are in a situation where schedule information needs to be combined with geographic information, for instance when the closest railway station to a specific landmark is to be determined. The amazing growth on the Internet lays the technical foundation by interconnecting almost all computer systems worldwide. The wide adoption of Internet markup languages such as HTML and XML are helpful, however, they only solve the syntactic problems of character encoding, escape sequences, and standardized parsing approaches. The interpretation of the semantics, i.e. the meaning of the data, is usually left up to the user. In the travel scenario mentioned above, this means that the user has to interact with both the travel and a geographic information system. Another major obstacle to arriving at the full potential of information technology lies in the limited functionality of most of today's applications. Obviously, there are no strict criteria for defining the characteristics of a smart or intelligent application. Relating to the point above, we think that smart systems should proactively be able to integrate data from various sources. A travel system that can tell the user which train to take, at which station to get off, and how to get to the landmark will definitely be perceived as being smarter than the individual solutions. The software engineering perspective is also very important. We believe that today's applications lack intelligent behavior due to the associated implementation cost.

The idea is to apply ontologies in order to tackle these problems. An ontology

can be defined as a formal specification of important domain concepts and their relationships, which is agreed upon by a large community. Ontologies have been used before. Their impact, however, was definitely limited, mostly because the construction and maintenance of an ontology is a very costly task. The recent, amazing comeback of this technique is fueled by the Semantic Web, which applies ontologies on the Internet. The vision of a Web of cross-linked ontologies, managed in a heterogeneous and distributed fashion is obviously quite appealing.

Our thesis is that data integration and software engineering can benefit from establishing ontologies as the basis for a variety of intelligent applications. This statement draws an analogy to today's situation, where relational datamodels are the basis for a large array of enterprise applications. We claim that it is not necessary to reinvent the "AI wheel". However, there is a clear need for more efficient engineering techniques and a large-scale integration of the various existing approaches. We believe that ontologies can be the catalyst for achieving these goals.

1.1 Application Scenario

We chose a help system scenario to develop and validate the thesis. A concrete example allows us to analyze the requirements of such an intelligent system and how ontologies can help during its development and operations. The example also serves as a proof of concept, since we applied our own ideas while developing various aspects of a help system. The broad context of the scenario is online learning. More specifically we address help systems where the user searches for a small piece of information rather than an entire course on a subject. Note that the system will not generate the answer. Instead, it will point the user to a document[1] that contains the answer.

We choose this topic for the following reasons. Online learning is a very challenging and important area since the world's leading nations are currently witnessing the transition from the industrial to the knowledge society. Furthermore, the area of online learning definitely suffers from a lack of data integration. Today, a learning system typically comes with pre-packaged content as well as an authoring system for producing one's own material. A well-controlled set of metadata describing the difficulty level, didactic style, learning goals, and other attributes is used to realize fairly sophisticated functionality such as curriculum planning or learner performance assessment. However, due to the high production cost for online material, most of the closed systems suffer from their small content base. Alternatively, one can try to leverage material off the entire Internet. The desired document will most likely exist, finding it and making it work with one's system, however, becomes the problem. Last but not least, without being explicitly stated, ontologies already play a role in today's document re-

[1] In the following text, we often refer to documents. We are not limiting this to text documents. Instead, by document we mean any kind of multimedia unit such as an animation, an image, a video, an HTML page, or a text document.

trieval techniques. Consider a search using an Internet directory. Many people argue that the directory's taxonomy already defines important concepts and a commonly agreed upon vocabulary for them. Furthermore, as we mentioned before, the metadata of teaching objects commonly talks about learning goals and relationships to other resources. Again, important concepts are formalized, thus at least partly fulfilling ontological requirements. This becomes an important issue, especially when content is supposed to be exchanged between different content repositories.

1.2 Requirements Analysis

The common approaches for finding information, namely full text search, keyword search, and searching a taxonomy of document categories, offer quite a bit of potential for improvement. This section motivates looking for alternatives by taking a close look at how humans collaborate in a social network in order to retrieve the right information. The analysis of his analogy will illustrate some of the shortcomings of the usual brute force methods applied by computers. It also provides us with the requirements of more advanced systems.

Whom to Ask? Today, using search engines only unleashes the immense computational power of modern computers. This approach undoubtedly yields very good results for some types of queries, but we all face situations where this is not necessarily the case. Let us look at how we obtain information in our daily lives: an important aspect could be characterized under the term "ask the expert". Assume we know that Jim is the database guru in our company. Therefore, Jim could probably point us to a good tutorial on JDBC. What is important here is that Jim also knows me, thus he knows which level of difficulty would be appropriate for documents he would suggest to read. If someone encounters a software setup problem while working on a term project, the right person to ask would probably be an experienced computer user who is taking the same class. Chances are that this person has already encountered and solved the same problem. This addresses the issue of personalization. We would probably also find the relevant information on the Internet, but in such a personalized environment, the search precision is much higher. It is important to note that in such a scenario things can be quite fuzzy. The more people you ask, the more answers you get. Ultimately, it is up to the searcher to judge whom to trust and whose answer to believe. This can be based on previous good experiences with somebody's advice or the person's reputation. A similarity to this observation can be found when searching for information on the Web: an experienced Web user applies a lot of rules on which site to use in order to find a certain piece of information. Someone's homepage is best found using the Google search engine, which usually brings up the correct URL as the first result. Organizations or companies are usually registered with major Web portals. One probably chooses a German portal like web.de or yahoo.de if a German company is to be found. Finally, the population of a country is best found at a site like the CIA world

fact book. All these examples demonstrate that searching the Web is not an easy skill.

Clarification Dialogue Once the expert is identified, people usually engage in a conversation where the expert tries to find out more. A user saying: I am having trouble setting up software X might prompt the expert's question: which operating system are you working on? Finally this interaction is ended with the expert providing an answer, pointing to a document, advising to consult another expert, or saying: I don't know.

Integrating Data Obviously these processes are very complex and several quite fuzzy heuristics are involved in every step. However, a common feature that can be identified is that humans are able to leverage large amounts contextual and background knowledge for their reasoning process. This becomes clear when the input of a simple full text search on JDBC, i.e. the query "JDBC tutorial", is compared to the example above where data about the skills and preferences of the user, detailed knowledge of the domain, and rich information on the documents are combined in Jim's decision making process.

The points above clearly show that there is room for improvements. However, it also becomes evident, that approaches from several different areas need to be considered when attempting to build a system that mimics some of this behavior. Natural language processing plays a role for the user dialog, agent technology can help to model the collaboration of humans with different backgrounds, knowledge representation plays a crucial role in enabling machines to perform some sort of reasoning, and last but not least middleware and data integration issues are important, for instance to merge information from the university enrollment system with metadata of a teaching object.

1.3 Key Research Questions

Based on the requirements analysis, we will now outline the key research questions, which are addressed in this thesis.

1. The Semantic Web allows users to mark up their data in a machine understandable way. Consequently, it is a key ingredient for our architecture, since only a large shared pool of semantically enriched data will make intelligent approaches superior to today's brute force mechanisms. While the technical specifications and tools are already in place, it is unclear whether a wide community of users will adopt the Semantic Web. Therefore, a key question is how much and what kind of semantically tagged data is available on the Web today.

2. From the requirements analysis it is clear, that an intelligent help system cannot be viewed from an isolated perspective. On the one hand, it should

base on an ontology to facilitate sharing data and to benefit from existing knowledge. On the other hand, it must be easy to integrate a solution with existing information systems. Furthermore, we expect scalability to be a major concern, since we operate is a Web setting. Consequently, the second key question is whether rules and ontology can be integrated into a mainstream IT infrastructure of databases and object oriented programming languages.

3. We identified an agent-oriented architecture as an important component. Rather than relying on a single document and knowledge repository, a solution should include various agents coming from different backgrounds and contexts in finding the right information. This is obviously a very challenging task. In this context, the first question to answer is whether ontologies and event condition action rules can be used to define the behavioral aspects of agents. A second step, which is outside the scope of this thesis, would deal with developing suitable behavioral patterns for solving the information retrieval problem in a distributed environment.

4. While agent based solutions have yet to catch on to everyday IT solutions, Web Services are a promising alternative, which is viable in the short-term. A significant number of services are already being offered. With Web Service support being built into tools like Microsoft Office, this number is likely to grow very rapidly. Services providing language translation features, knowledge base searches, or GIS data can be an essential source of information to an intelligent help system. The key question is, whether it is possible to invoke a semantically tagged service on the fly, i.e. without having seen the service interface description before or the service implementing a known standardized API. Note that this could be interpreted as the first step from a traditional distributed system with predefined APIs to an agent based architecture.

5. As pointed out by the clarification dialogue requirement, a sophisticated help system must provide natural language functionality. The key question is whether it is possible to base these features on the same mechanisms that are used for obtaining data from information sources other than the user. This includes marking up linguistic information, maintaining contextual information, as well as computations to be performed for understanding or generating language.

6. Consider the previous points from the agent infrastructure to linguistic features. All the various system components need to share the same data model and the same knowledge base in order to collaborate. Therefore, arguably the most important question is whether an underlying ontology provides the necessary foundation for these components to interoperate seamlessly.

7. The last key question is whether ontologies prove to be a useful help for developing a complex application. We answer this question by applying

our ontology driven infrastructure to a small help system prototype. This serves two purposes: Firstly, it provides a proof of concept that ontology based software development and data integration works. Secondly, it shows that some of the alternatives to today's brute force methods described above are viable.

1.4 Overview of the Remaining Chapters

This thesis is organized as follows. The next chapter provides a more in-depth background on some of the issues and technologies like document retrieval, digital libraries, metadata standards, application integration, and engineering issues that were mentioned briefly in this introduction. From the metadata standards, we discuss how certain vocabularies, and in a general sense, ontologies, play an important role for the interoperability of metadata sets. The term ontology is defined and different communities' viewpoints on this very ambiguous term are explained. The related work chapter mainly introduces work in the areas of intelligent user interfaces, question answering, and knowledge management and compares them to our ideas on help systems. In chapter 4, we analyze which requirements are essential for making such intelligent help and document retrieval systems successful. Chapter 5 introduces the state of the art in the area of the Semantic Web, which we believe to be a key technology in the solution to be presented in this thesis. We cover emerging markup language standards, which allow us to handle metadata, domain knowledge, and rules in a uniform way. Chapter 6 provides a survey of available resources that one could build upon when implementing a new system. This includes Internet-based knowledge sharing efforts, data collections from the open source community, well-established databases such as Wordnet, and also an overview of software supporting these emerging standards. Chapters 7, 8, and 9 describe our major design decisions, our ontology-based application building framework, and our case study, the implementation of the SmartGuide intelligent librarian system. We introduce our basic components, which lay the foundation for higher-level applications such as the SmartDialog clarification dialog system. Here, an analysis of the knowledge base is used to determine which clarification question should be answered by the user. This thesis ends with chapters 10 and 11 that provide an outlook with an outline of possible extensions to the work presented as well as a summary.

Chapter 2

Background

This chapter begins with a definition of ontology and its relation to terms like vocabulary or schema. We analyze various communities' views on this issue since several misconceptions about the term ontology arise from the vastly different application backgrounds. The background information provided in this section attempts to resolve this and provide a general definition. We then pick up and explain our application scenario in more detail. This is followed by a description of current document search techniques and metadata standards. We will illustrate how ontological issues are already present in today's metadata standards, even though they are not explicitly mentioned as such. Current work in the area of online learning systems is a good case for the need for an ontological underpinning.

2.1 What is an Ontology?

The Encyclopedia Britannica defines the term ontology as "the theory or study of being as such; i.e., of the basic characteristics of all reality". This definition reflects the field of ontology, which has been around for centuries and is a subdiscipline of Philosophy and Metaphysics. If a search for "ontology" is entered into the Google search engine today, an estimated number of 352,000 pages are returned. Google rates sites according to the number of references to the site from other hosts. The top results come from fields like e-Commerce or Biology, which are completely unrelated to the original meaning of ontology. Rather than Metaphysics and the study of being as such, we can read about "enabling virtual business" or the "Gene Ontology Consortium". This observation is in line with McGuinness' noting "the emergence of ontologies from academic obscurity into mainstream business and practice on the web" [110].

Gruber provides one of the most cited new definitions of ontology as "a formal, explicit specification of a shared conceptualization" [72]. Here, a conceptualization refers to people's conceptual understanding of a certain domain. While being very general, this definition captures the essence of what ontology

means, regardless of potential application areas one might have in mind. Many other definitions can be found. Campbell and Shapiro consider an ontology to "consist of a representational vocabulary with precise definitions of the meanings of the terms of this vocabulary plus a set of formal axioms that constrain interpretations and well-formed use of these terms" [20]. This definition emphasizes the formal logic aspects of ontology, the intention of wanting to reason with ontological constructs. In contrast to this viewpoint, Uschold and Jasper give a very loose definition and state, "An ontology may take a variety of forms, but necessarily it will include a vocabulary of terms, and some specification of their meaning" [141].

The range of definitions relating ontology to strict formal logic on the one hand to comparing it with a simple vocabulary of terms on the other hand, makes it very evident that there is a lot of disconsensus as to what an ontology actually is. This is the cause for quite some misunderstanding when researchers talk about this topic. Obviously, this also makes it very hard to establish ontologies as the foundation of the next generation web.

We approach this issue by outlining how different user communities use the term ontology.

2.1.1 Ontologies as Reference Vocabularies

People in different countries, companies, and sometimes even among various company departments often develop their own language for everyday terms they are dealing with. A middleware layer that manages access to a central database might be referred to as connector, pool manager, access layer, and so on. At the most basic level, an ontology is simply a set of terms with very detailed and unambiguous descriptions attached. Another essential property of such a reference dictionary is that a large community accepts it. When users within this community communicate, they can refer to this repository in order to understand each other. This works much like people from different countries agreeing on one language they use to talk to each other.

2.1.2 Ontologies as Taxonomies

Taxonomies are going one step further. Here, the terms and concepts that comprise the vocabulary are also placed in a hierarchy of is-a relationships. This structuring usually comes very naturally, and a taxonomy can be found in almost every ontology. They are also referred to as the taxonomic backbone [75]. Taxonomic classification is frequently used, with the yellow pages being the most prominent every day example. Here, companies are categorized according to an agreed-upon hierarchy of commercial sectors. If the taxonomy would be poorly designed or unnatural for people, search precision and recall would definitely suffer. There are many well-established categorization hierarchies. Examples are the North American Industry Classification System (NAICS), the Universal Standard Products and Services Codes (UNSPSC), and the ISO 3166 geographic taxonomies. These standards are also used with the Universal

Description Discovery and Integration (UDDI) Standard, explained in detail in section 5.8.2.

We believe that Internet directories like the Open Directory and Yahoo! can also serve as an important reference point. Labrou and Finin support this view [96]. It is quite clear that an Internet directory can only serve as a fairly high-level ontology since no detailed concepts are available in there. However, their breadth is very appealing. The wide acceptance and visibility of such portals are even more important. Obviously, the chances of successfully communicating with another agent are much higher when such a mainstream ontology is used. As an analogy, one could compare this to a situation of asking for directions, where it is also advisable to choose a mainstream language such as English.

The Gene Ontology initiative is another example for this kind of ontology definition. In order to facilitate the integration of the various genome databases, one important base is to have a taxonomy of terms and concepts that a wide user community agrees upon. Besides the basic subclass relationship, the Gene Ontology also contains the part-of relationship [25].

Even though a taxonomic arrangement of terms is usually quite natural to create, Guarino and Welty argue that there needs to be a formal approach to design and test taxonomies [74]. The rationale is that errors in the design of the backbone taxonomy will create multiple problems when trying to use these ontologies at a later point. Again we can draw on experience from data modeling, where a flawed database schema will greatly complicate the development of database applications. Overall, there is little formalism to help ontology engineers do their job. Guarino and Welty introduce identity, rigidity, and unity which describe properties of taxonomic concepts and which can be attached to the concepts. These descriptions are inherited from parent concepts. The idea is to eliminate all inconsistencies that arise from a situation where a concept is tagged with an attribute that clashes with the attributes of a parent class. An example would be the parent concept being rigid and the child concept not being rigid. Instead of explaining the theoretical underpinning, which can be found in [74], we want to list the most common design errors given in that paper.

Errors commonly result from mixing up subclasses and instances. For example, human cannot be a subclass of species, since a human is identified by a position in space and time. For instance, if two humans are at different locations at the same time, they must be different. This clashes with the identity of a species, which is determined by the position in a biological taxonomy. Since meta properties, in this case identity, would be inherited from species to human, human cannot be a subclass of species. Another common mistake is the misuse of the part-of and subclass-of relations. The example bases on rigidity. A property is rigid if it is true for any instance of the class. An engine is part of a car and not a subclass of car, since the rigid properties of both, i.e. being able to accommodate persons and being able to generate rotational force, do not match. Again, the rigid property would be inherited by the subclass and this clash reveals the design flaw. Often the subclass construct is used to model a disjunction limitation. An artificial class car-part with its subclass engine is an

example. What the modeler wants to express is the fact that a car part must be either an engine, a wheel, or so on. However, this is different from saying that all engines are car parts, which is not true if a boat's engine is considered. This statement is implied by the subclass relationship though. Polysemy, a concept well known in data modeling, is another frequent source for errors. The ontology engineer needs to be careful to distinguish the class of book, having a certain content and written by a certain author, and an instance of this class, a specific copy of a book being owned by a certain person and having a dent in the cover. It is crucial to model this as an instance-of and not a subclass relationship. Finally, the constitution relationship is often falsely modeled using the subclass of construct. An example would be an ocean consisting of water. Water is not a subclass of ocean though.

2.1.3 Ontologies and Schemata

The similarities and differences between the terms schema and ontology are not trivial to identify. Within the context of the OntoBroker project, ontologies are used to create document type definitions (DTDs) [47], suggesting both a close relationship between the terms but also ontology to be the higher-order concept. On W3C mailing lists, for example, the relationship between XML and RDF Schemata on the one hand, and ontologies on the other hand is one of the most frequently discussed issues. Similar arguments can be made about entity relationship or unified modeling language diagrams. Again, the answer depends on one's point of view and one's definition of ontology. Consider the well-known MathML[1] or ChemML[2] mark-up languages as an example. On the respective websites, we can find schemata containing terms like trigometric operation or hydrogen count. In addition to simply introducing these terms, some information and some constraints about their relationship are encoded as well. A simple example would be a molecule consisting of several different atoms. In the ChemML DTD and XML Schema, a molecule is defined as a complex data type having the type atom as one of its constituents. Typically, the relationship, i.e. oxygen being a *part of* the molecule water, is not defined explicitly. By employing the notion of simple and complex types, DTDs and XML Schema allow us to model hierarchical data structures with the types defined. As we will further illustrate in section 5.3.1, the Resource Description Framework (RDF) takes a different approach. Here, the relationships are at the center of attention. Rather than defining the molecule - atom relationship in the type definition of molecule, this relationship is explicitly defined in RDF. The type definitions do not include attributes or properties. The second major difference is the introduction of the RDF addressing scheme that bases on URIs. If the same URI appears in different contexts, it always refers to the same logical resource. This allows us to model directed labeled graphs, breaking the tree restriction imposed by XML Schema and DTDs. In order to have some kind of relationship with

[1] http://www.w3.org/Math/
[2] http://www.xml-cml.org/

another piece of data, an element does not have to have this data as its parent anymore. Generally, schemata support the definition of some constraints. Type constraints disallow non-atoms to be parts of molecules, cardinality constraints prevent zero-valued or negative coefficients, other constraints allow the restriction of non-numerical value ranges, etc. In general, we can say that if the weak definition of ontology as a vocabulary is chosen, schemata with a reasonable public acceptance can be considered to be an ontology.

If we switch to the strict definition of an ontology as a vocabulary with formal axioms, the differences and the deficiencies of schemata become more obvious. Andersen argues that the constraints provided by schema languages are far too weak to enforce the requirement that an ontology needs to define the kinds of things that exist in the application domain[3]. The example given shows that a simple XML Schema for address data fails this formal criterion, since in can by no means model the fact that the address "Hauptstrasse 56, D-69118 Heidelberg, US", though being valid with respect to the schema, is a nonsense address since Heidelberg is in Germany, not the US. Another example addresses temporal constraints, an issue that still creates a lot of problems in the database world. The statement that "Ponce de Leon lived at 21 Citrus Drive, Miami FL, 12345, US" cannot be true since the US was not around during Ponce de Leon's lifetime.

In our opinion, the motivations behind establishing ontologies and schemata are often different. Schemata are usually designed with a specific application in mind. If an XML Schema is written for a B2B application, one looks at what data is available in the respective information systems to begin with. A common design criterion for database schemata is the question of how a certain design will influence the applications at a later point. Often one faces a decision on whether to completely denormalize a schema or whether a small amount of redundancy actually makes things easier. In contrast to this, an ontology is usually designed without a specific application in mind. The focus is to conceptualize a certain domain of interest within a larger community of users. Consider the following simple example of students taking courses and courses being taught by instructors. A database schema would most likely not talk about the relationship of students being taught by instructors. The schema will implicitly assume that an instructor teaches a student if the student takes a course taught by the instructor. It would be redundant to explicitly store this information. The database application will compute this information by joining the respective tables. An ontology, on the other hand, would explicitly define all three relationships and probably also the three inverse relationships. The advantage is that any kind of statement can be interpreted, whether we say John teaches Joe or Joe is taught by John. However, this puts a bigger burden on the ontology software.

[3] http://lists.w3.org/Archives/Public/www-webont-wg/2002Apr/0001.html

2.1.4 Ontologies and Logic

Sowa describes the relationship between logic and ontology as follows: "Logic provides the formal structure and rules of inference. Ontology defines the kinds of things that exist in the application domain. Computation supports the applications that distinguish knowledge representation from pure philosophy" [132]. According to these statements, it is logic that makes ontologies useful for computer scientists. Therefore, logic is an integral part of any ontology-based solution according to this definition.

In this thesis, we support this point of view, but on a more practical level. We think of the requirements stated by Andersen and Sowa as very desirable, but long-term goals. The fact that a basic technology like XML creates such a stir in the IT world demonstrates that today's major problems are much simpler in nature than the complex issues pointed out in the address example above. We look at logic from a software engineering perspective as a tool that allows us to write simpler and more reusable applications. Chapter 8 will provide more detail on this and show how at least a small subset of logic statements can be executed by mainstream software tools.

2.1.5 Resolving the Different Viewpoints

In light of all these varying viewpoints, the question arises whether it is feasible at all to attempt to develop reusable ontologies, which form the basis for a suite of different applications. Aren't ontologies bound to be isolated solutions developed for a specific purpose? We do not think so. The level of descriptiveness might vary in the examples mentioned before, but the less descriptive parts, such as a concept taxonomy, can be found again in the more elaborate forms. Also, close ties to natural language and the desire to achieve consensus with respect to the interpretation of the terms specified are common attributes throughout all levels of definition detail. As we will see in the next chapter, the Semantic Web initiative provides users with a set of standard mark-up languages, allowing us to express ontologies in various levels of complexity. Any organization can pick up a basic taxonomy of terms, cross-link, and enrich it by axioms. Ultimately, a marketplace of ontologies will spring up, much like today's Web, where the usefulness of an ontology will be decided upon by the number of users it has.

2.2 Intelligent Help Systems

Our everyday lives are getting more and more complicated. Even simple devices are being loaded with functionality, most of which the user is unlikely to ever need to use. Consider an employee having to use a feature of the company's information system for the first time. Assistance in this case can be sought online. These problem solving support cases are different from a student learning about a more general topic via an online course. It will be much easier to locate the entire course rather than an FAQ or a short tutorial on a specific aspect of a software package, for instance. Also, once the course is started, the learning

management system or simple hyperlinks in the course's components will provide navigation support to locate the individual parts and subsections. Searching is not a very big issue in this case. In the other example, the documents or tutorials found will be consumed relatively quickly. The user is in the process of performing a certain task or working on a problem. In the process of doing so, he or she will come back to and use the search facility to find other resources quite frequently. Another aspect is that a course on a certain subject will most likely have a consistent look and feel in terms of the educational style and the onscreen design. When a user is jumping back and forth between the task to perform and a help or support system, it is not that important for the individual units to have a consistent look and feel. After all, the user will most likely search for resources out of very different contexts. Thus, compared to a pure learning scenario, the possible base of relevant documents is larger and searching is a more frequent and more important operation in such support or online help cases. Nevertheless, it should be possible to adopt the design and implementation principles of a solution developed in this context to other application areas as well.

There are two major approaches to question answering. The classical way, implemented in most traditional expert systems, will attempt to *generate* an answer based on the reasoning process. With the ubiquitous availability of billions of documents on the Internet, a second approach of the system pointing to a document that *contains* the answer becomes more viable. We refer to this as the "smart librarian" approach. With this method, question answering can be seen as a subset of the more general problem of document retrieval. The next section will introduce the major concepts of document retrieval.

2.3 Document Retrieval

Document retrieval can be defined as a function SE that maps a list of search inputs Q to a list of documents $(d_1, d_2, ..., d_n)$:

$$SE(Q) = (d_1, d_2, ..., d_n) \tag{2.1}$$

The input depends heavily on the system. The classical inputs would be a single keyword, a category to be displayed, or a Boolean expression with terms specifying whether a word or phrase must or must not appear. However, it is also possible to use someone's nationality or educational background as inputs. The output will usually be ranked according to the documents' relevance or contain a probability rating associated with each result.

2.3.1 Definition of Precision and Recall

Precision and recall are typically used measures to quantitatively assess the performance of a retrieval algorithm. Let D_{rel} be the set of relevant documents and D_{result} be the set of documents appearing in the search result. Then, precision P and recall R of the search $SE(Q)$ are defined as follows:

$$P_{SE(Q)} = \frac{|D_{rel} \cap D_{result}|}{|D_{rel}|} \tag{2.2}$$

$$R_{SE(Q)} = \frac{|D_{rel} \cap D_{result}|}{|D_{result}|} \tag{2.3}$$

This traditional definition of recall is only applicable for a limited document base. Obviously it is impossible to determine all the relevant documents on the Web with respect to a specific search. Furthermore, since in a large documents repository several documents will contain similar information, it is very likely that a user will be fully satisfied after examining a subset of all relevant documents. For these reasons, recall is much less important than precision. An important observation for measuring the recall is that one can define several measures of relevance for documents. Many earlier studies work with discrete classes such as "relevant", "partly relevant", and "not relevant". Search engines often compute percent values of relevance and a document might actually only be partially relevant. The formulae above only characterize a yes/no decision.

There is a general trade-off between precision and recall. If all documents are returned, recall is guaranteed to be 100% but precision will be close to 0%. If the search yields only one matching document, the situation is reversed. Therefore, it is crucial to find the right balance for the respective application.

2.3.2 Text-based Search

In this section we want to briefly introduce the major concepts of full text or text-based search. This method is definitely the most frequently used since it is very cheap and easy to implement. Full text search does not require additional metadata to be entered. It relies completely on the computational power, the immense storage capacities, and the available network bandwidth in modern computer networks.

Crawling The first step in the process of setting up a full text search environment is to determine which documents should be covered. This can be the contents of certain folders on a file system or a set of pages on the Inter- or Intranet. Crawling refers to automatically traversing web pages by following hyperlinks found inside of the documents. The documents' locations are typically stored in a database. Figure 2.1 shows a possible database schema. This way one can avoid the crawler from accessing the same page twice in the process, since a history of visited pages in available.

A problem lies in the diverse media types. While it is quite easy to obtain the text from HTML, Office files, or PostScript documents, the sheer amount of different formats poses some problems here. Extracting information for indexing from multimedial content such as animations, audio, and video files in arbitrary formats is not an impossible, but at least a rather complex and very time-consuming task [101, 121]. Uhl and Lichter present a powerful alternative

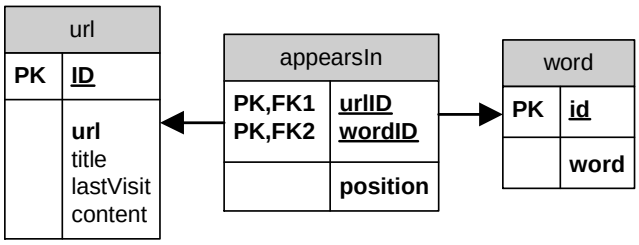

Figure 2.1: A possible schema for a full text search system.

approach for searching multimedial content in a distributed fashion [139]. However, we believe that it is unlikely that such an architecture will be adopted on a wide scale.

Indexing Figure 2.1 illustrates how data collected by crawlers could be organized in a central database. The locations scanned and words found are stored in respective tables. This makes it easier and more space efficient to reference them in the central appearsIn table that denotes the position of every word found in every document. Usually prepositions, articles, and other words that are unlikely to be searched will be skipped when a document is processed. In order to be able to still provide the textual context of a keyword found, the entire content can be stored as a compressed binary large object in the url table. Considering that pure text takes up only a fraction of the space required by images, audio, and video, such an approach will be able to store and retrieve the text of a very large document base. The search process then involves joining the tables under the respective Boolean and text phrase conditions. Indexes on the word IDs make sure that these searches can be performed efficiently.

Page Ranking Due to the massive amount of data on the Internet, conventional measures like recall become less important. Ranking the search results, however, is crucial. If an Internet search returns 5000 matching documents, a user will most likely only browse though the first couple of top ranked pages. A link appearing after that will most likely be ignored. A whole collection of strategies has been developed for this purpose. The quotient of word occurrences to document size or the proximity of the search words within a document are commonly used methods. The Google engine's popularity is largely based on its outstanding page ranking system. It uses two fairly obviously but extremely effective approaches. If the search terms appear in Google's Internet directory pages (see section 2.3.4 below), the matching directory entry is returned. This way, a user searching for a company's homepage is able to find it within seconds. Otherwise, Google ranks pages by the number of links referencing the page from other hosts. The rationale is that high quality pages will be referenced frequently. The advantage is that, in a way, a large user community

"decides" upon the ranking.

Linguistic Features The approach described so far treats words and documents like ordinary data structures. Some very basic linguistic features are able to augment the overall system. One problem is posed by the several possible endings of a verb like "wait" for instance. The strings "waited", "waiting", and "waits" are all treated as different. Stemming reduces these forms to their morphological root, greatly improving the search. Stemming algorithms differ greatly in the level of sophistication and depend heavily on the base language [58]. They are often used to improve recall in document retrieval applications [93].

Another frequently applied method is looking up synonyms in a thesaurus. A search for "man" will then also return a document containing the word "guy". These features protect the user form having to try out different words with the same stem or different synonyms in the search queries.

The third problem lies with words that have several meanings. A quote can refer to a stock quote and a literature quote, possibly causing completely irrelevant documents to be suggested. This issue can be addressed by disambiguating the meaning using the context. If other words like "Wall Street" and "Dow Jones" appear in the same sentence as "quote", it is safe to assume that a stock quote is meant.

2.3.3 Keyword-based Search

The idea behind keyword-based search is that the pure occurrence of a word does not imply that a text is mostly about a concept characterized by the word. A simple example are negations such as "this document is *not* about glycerin". Instead of relying strictly on the text, keywords must be provided along with the document. Technically, the information can also be stored in a structure as the one shown in Figure 2.1. When choosing the keywords it is important to specify both general and quite specific ones to make sure that both general and specific queries will retrieve the respective document. Again, a trade-off between precision (specific keywords) and recall (general keywords) can be observed here.

2.3.4 Taxonomy-based Search

The most prominent example of a taxonomic classification of Internet pages and also its first adopter is definitely the Yahoo! portal. The Web is partitioned into several top-level categories, which are themselves subdivided. This hierarchy of areas is referred to as a taxonomy, since the subcategories are usually subclasses, or specializations, of the upper class or category. In contrast to pure full text indexing, several human editors decide whether a certain site is worth listing and make sure that accepted sites appear under the appropriate category. This is obviously a large effort, however, the high quality of its portal ensured Yahoo! a loyal user base. Yahoo!'s arguably biggest competitor is the Open Directory project which has been adopted by Google. This project functions much like

Yahoo, but it bases on currently over 50,000 volunteer editors. Section 6.1.2 will describe this initiative in more detail.

2.4 Metadata Standards

The following examples for metadata standards are intended to provide an overview of the typical kinds of information stored in web- and online learning related metadata sets. It is by no means a complete listing. Along with this overview, section 2.4.3 shows that it is often not quite clear how the data is to be interpreted. This observation will led us to several issues related to application integration and semantics which we will elaborate on in section 2.1.

2.4.1 HTML Metadata

The hypertext markup language is undoubtedly the most common and most accepted format on the Web. The World Wide Web Consortium's specification[4] provides for metadata to be inserted directly at the beginning of the document. The following example illustrates how a set of relevant keywords can be specified using the HTML Meta tag.

```
<html>
    <head>
        <META NAME="keywords" CONTENT="Schach Chess Echecs">
    </head>
    ...
```

HTML metadata can be grouped into two categories. Content related fields are the keywords, a textual description of the page, plus information on the author and the tool used to generate the file. The aim of this data is to help search engines index documents. The second category is more technical in nature. It deals a lot with browser and caching issues. Furthermore the page author can provide information for robots crawling the site during the indexing process.

Even though HTML metadata is so simple, it is considered to be a prime vehicle in increasing the ranking of websites with search engines. Playboy Inc. actually sued two adult web site operators for including the keywords "Playboy" and "Playmate" hundreds of times in their metadata. This caused several search engines to return those sites ahead of playboy.com in their search result pages. This practice, which is often referred to as keyword spamming, led several search engines to take countermeasures against these practices in order to improve the quality of the results.

[4] http://www.w3.org/TR/html4/

2.4.2 Dublin Core Metadata

The Dublin Core Metadata initiative[5] organizes discussion forums and workshops that aim establishing consensus-driven efforts to promote widespread acceptance of metadata standards and practices. Currently, a vocabulary for describing quite straightforward fields like title, author, format, keywords, language, or audience is defined. The Dublin Core website lists about seventy major projects such as the Australian Government Locator Service or the Berkeley Digital Library Catalog, that use Dublin Core metadata. Due to its pioneer efforts and wide acceptance, the Dublin Core elements are a building block for several other initiatives, among them SCORM, which is presented in the following section.

2.4.3 Sharable Content Object Reference Model

Several initiatives have been launched that deal with defining a standard metadata model. Overall, most of them can be characterized as conceptually similar. The ultimate advantage of standardization is the possibility to exchange content between different learning and content management systems. It is clear that the didactic approach and the visual look and feel of a teaching unit are hard to standardize. Therefore, it will probably never be possible to compose a consistent large teaching unit from components written and designed by a large number of authors that simply upload in their learning object into some system. Unless the authors collaborate closely, such an undertaking will most likely result in a heterogeneous, fractioned end product[6]. On a technical level, a standard like SCORM allows to move content between different systems. An XML metadata serialization and packaging format is defined within the framework. Therefore, teaching system A can serialize and export metadata in this XML format in order for system B to be able to load it into its own internal data structures. This is obviously a very appealing feature that brings us a step closer to a global marketplace of teaching content. This solution can be though of as some sort of business to business (B2B) data exchange standard for the education sector, similar to the well known B2B exchange formats like ebXML[7], the Rosetta Net[8] initiative for semiconductor manufacturing companies, or SWIFT[9], the de facto standard in the financial sector.

Unlike the HTML meta tags described in section 2.4.1 above, the SCORM metadata can practically reside anywhere and appear in many physical representations such as a main memory data structure of the learning system, inside a relational database, or as a serialized stream of XML characters inside a file or traveling across the network.

[5] http://dublincore.org/

[6] Within the Multibook [134] project, a dramatically new approach to authoring is illustrated that might eventually provide a solution to this problem.

[7] http://www.ebxml.org/

[8] http://www.rosettanet.org/

[9] http://www.swift.com/

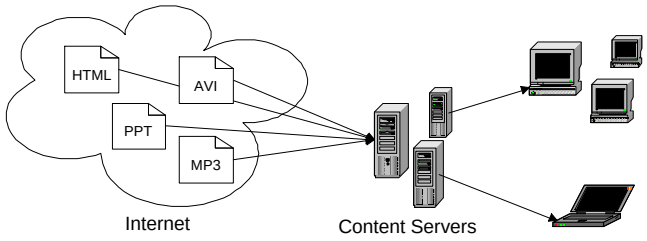

Figure 2.2: The SCORM metadata standard and API definitions are supposed to display an appropriate combination of learning units in a variety of situations.

In 1997 the Department of Defense and the White House Office of Science and Technology Policy launched the Advanced Distributed Learning (ADL)[10] initiative. From our point of view, this initiative seems to be the one with the most momentum and the widest acceptance. The purpose of the ADL initiative is to ensure access to high-quality education, training and decision aiding materials that can be tailored to individual learner needs and made available whenever and wherever they are required. Figure 2.2 illustrates the ADL vision. The World Wide Web is thought of as a giant content repository. A SCORM server can read and interpret the associated metadata and compose a set of documents, sounds, or other multimedia files into an appropriate learning or support unit. Apart from the pure metadata definitions, SCORM also defines an application programming interface (API) that allows a content object to interact with a so-called learning management system (LMS). This API would enable an exam object to write the student's score back into some sort of grades database hosted in the LMS.

The SCORM metadata standard references the IMS learning resource metadata information model[11]. In turn, IMS bases on the IEEE Learning Technology Standards Committee (LTSC) Learning Objects Metadata (LOM) Specification[12] that was developed as a result of a joint effort between the IMS Global Learning Consortium, Inc. and the Alliance of Remote Instructional Authoring and Distribution Networks for Europe (ARIADNE)[13]. Together, these specifications form the SCORM content aggregation model. Rather than a complete data model, SCORM is though of as a base that can be extended to meet a specific project's needs. D. Suthers [136] for instance, reports of having to extend the metadata model when applying it to a set of primary and secondary school resources.

[10] http://www.adlnet.org/
[11] http://www.imsglobal.org/
[12] http://ltsc.ieee.org/
[13] http://www.ariadne-eu.org/

2.4.4 SCORM Data Model

The following summarizes the individual parts of the SCORM metadata model.

General The General category contains the information describing the resource with its title, the language it uses, its keywords, a textual description, etc. All in all, this category contains the most basic data needed.

Lifecycle The Lifecycle category groups the features related to the history and current state of this resource as well as the users who have edited this resource during its evolution. Sample attributes include the authors, their roles, the version, and the status of the item. An interesting aspect is that a person's information is stored in the vCard format[14].

Meta-metadata The Meta-metadata category holds authoring and version information on the meta-data record itself, rather than the original teaching object being described. Consequently, these categories fields are very similar to the lifecycle category.

Technical The Technical category groups the technical requirements and characteristics of the resource. The attributes include the resource's mime type, the size, the URL, or the client's software requirements.

Educational The Educational category consists of the educational and pedagogic characteristics of the resource. The interactivity level, the difficulty, the intended end user role, and targeted age group are among the attributes here. Educational properties are obviously quite fuzzy in their nature. SCORM defines a value range for these fields. The semantics of the individual values are defined by a longer textual description. Examples are the values for the difficulty of an object, which can range from "very low" to "very high".

Rights The Rights category deals with the intellectual property rights and conditions of use for the resource. This category only contains very rudimentary information that would have to be extended to be the base of a digital rights management application.

Relation The Relation category groups features that define the relationship between this resource and other targeted resources. This category is quite interesting since it allows storing information about the resource's prerequisites and its position within a navigation structure. A chapter and its sections could all be modeled as resources, with the chapter having a containment relationship with the sections.

[14] http://www.imc.org/pdi/

Annotation The Annotation category provides comments on the educational use of the resource and information on when and by whom the comments were created.

Classification Finally, the Classification category describes where this resource falls within a particular classification system. In our point of view, this category is of utmost importance since it allows stating what the document is about. This is done by referencing a standard taxonomy as it can be found in libraries or the aforementioned online taxonomies that are the basis of web portals like Yahoo!. A tutorial on the Power Builder database development suite might reference the Yahoo category Home > Computers and Internet > Software > Databases > Development > Power Builder. Furthermore, the resource's discipline, the required educational level, the didactic style, and other aspects can be modeled by referencing respective taxonomies. Obviously the key point here is that the taxonomy to be referenced should on the one hand be well known, while on the other hand it should be fine grained enough the describe the document in as much detail as possible. The taxonomic path shown above is definitely more descriptive as the most detailed corresponding web.de: Verzeichnis > Computer & Software > Programmieren > Datenbanken.

It must be noted that the SCORM standard only specifies the meaning and the semantics of its metadata elements. These definitions might suggest a certain use of the respective fields, however, this is left up to the application. How a learning management system uses the data in order to derive an educationally appropriate teaching object sequence for a particular user it left open entirely. Another problem is that the highly relevant relation and classification categories are vaguely specified. SCORM only talks about how a classification scheme or related teaching unit can be referenced on a technical level. The semantics of such references and best practice suggestions are missing entirely.

Chapter 3

Related Work

There has been interesting work done in fields related to our sample application area of online learning and help systems. One aspect of knowledge management, for instance, is concerned with representing and conceptualizing knowledge such that it and also mostly its transfer from employee to employee can be properly managed and supported by tools. In the introduction, help systems were mentioned as one main application area. Help systems imply that a user is trying to get an answer for a specific query. This is quite similar to a well-established information retrieval topic known as open domain question answering. Software agents are supposed to find the answer to trivia-like questions within a publicly available repository like newsgroup posts, a set of newspaper articles, or the Web as the largest available repository.

3.1 Question Answering

The general goal of a question answering system is to either *generate* the answer to a natural language question or to simply *point to* a suitable answer. The answer generation approach has been more popular. It was implemented in solutions like the MIT Start geological answering agent. Generating an answer obviously is more complicated then pointing to an existing one. Therefore, these systems typically suffer from a small target application domain. With the success of the Internet, the focus began to shift. Within a few years, millions of documents on practically any topic were available. Obviously, this made the smart librarian approach of pointing to an answer a much more viable solution. Once more research was put into the retrieval strategies, the systems also became more practical. Systems like Mulder, for instance, are capable of answering questions from any domain. We will proceed with a summary of a few sample question-answering systems.

MIT Start System The MIT START system[89] was one of the first question answering systems to be exposed via a web interface[1]. It allows users to ask questions about two application domains: geographical facts and MIT's Artificial Intelligence Laboratory. Questions like "which cities are within 50 miles of Rome" or "which countries border Finland" are answered based on knowledge from the CIA world fact book. The system is also quite flexible and understands most geographical terms and phrases one can think of. Once the question is matched to certain facts in the knowledge base, START follows a pointer back to a corresponding piece of text that was used to extract this knowledge, which is then displayed as the answer. These pointers are manually fed into the system using text annotations. While the START system demonstrates outstanding accuracy and is able to understand most natural language questions, the biggest drawback is its limitation to the well-contained world of geography and the lab in which it was created. A definite advantage of choosing geography in particular is that there are plenty of well-structured knowledge bases available for this area. The limited scope of START is definitely caused by relying on manually typed annotations. Nevertheless, START was definitely an early trendsetter. After all, the Semantic Web community adopts the strategy of basing intelligent applications on manual annotation entry instead of information retrieval algorithms. Consensus has been reached, however, that this is only feasible if annotations are created and maintained by a large user community in a distributed fashion and if that data can be exchanged via the Internet. We will cover this issue in more detail in chapter 5.

Mulder Mulder is another question answering system developed by Kwok et. al. [95]. It functions radically differently from the MIT Start system. It also takes on the different, more complicated task of answering questions from any domain. Mulder employs the so-called information carnivore strategy, i.e. it resends the question asked to another search engine, which in Mulder's case is Google. The process begins by classifying the questions. Using the results from this analysis, Mulder then rephrases the original search query before it is sent on. The goal is to guess words that would appear in a sentence containing the answer. Since Google returns not only the URL of matching pages, but also the text block containing the word matches, a sentence containing the answer is likely to be part of Google's answer already. If the question is "who was the first American in space?" the rephrased query could be "first American space", which may match a solution like "the first American in space was Alan Shepard". The difficulty is to guess how specific the query should be. A fairly unspecific query like in our example might return too many hits. Alternatively, search engines can be fed with quoted phrases as well. Mulder solves this specificity / generality tradeoff by sending different query versions in parallel. The result summaries obtained from those searches are then analyzed and the potential answers extracted. This is done based on how far the question keywords are apart within the search-hit summary provided by Google. If the words are

[1] http://www.ai.mit.edu/projects/infolab/

scattered far apart in the document found, it is less likely that they form the answer. These word distances are also weighted using the popular inverse document frequency, which assigns high weights to uncommon words. Obviously when searching for answers on the Web, not all of them will be correct. The authors give the example that a commonly given but wrong answer to the question above is "John Glenn". Since the system can only return one answer, it uses a voting procedure to determine which of the multiple answers found by the search engines will be picked. Similar results are first clustered according to similarity. This avoids counting a correct but misspelled answer as a separate answer. Finally, a simple vote is taken and the most frequent answer is picked.

The results are surprisingly good. The TREC8 question corpus (see paragraph below) was used for testing, and the Mulder system was compared to Google and AskJeeves[2]. The performance was evaluated by the number of words a user has to read before finding the answer. Furthermore, the recall, i.e. the number of questions answered correctly, was considered. On average, Mulder outperformed Google by a factor of 6.6 in terms of user effort while achieving the same recall. AskJeeves responds to natural language questions, but surprisingly, it performs worse than Google. According to the authors, this might be an indication that AskJeeves' question answering strategy is not fully automated. It appears as if the most frequently asked questions are answered by manually pre-selected replies.

In general, Mulder can be seen as a module on top of Google making it more user friendly by applying answer extracting and query rewriting knowledge as well as by running multiple queries in an automated fashion in order to determine the necessary degree of query specificity. While the results are quite impressive and demonstrate how the Internet and its resources can be used for question answering in a powerful way, one might criticize that Mulder is probably tuned for TREC8-like questions.

FAQ Finder Like the Mulder system, the University of Chicago and DePaul's FAQ Finder project [19] also targets the Internet as a knowledge base for question answering. Several newsgroups offer a set of frequently asked questions (FAQs) to their newcomers in order to prevent those questions from being posted over and over again on the mailing lists. These FAQs were identified as a valuable source of information since usually a lot of work and experience from an entire user community went into their development. Similar to Mulder's problem setting, the system is supposed to speed up the time it takes a user to find the answer she or he is looking for and provide an alternative to wading through the FAQs manually.

In order to perform a first pre-selection within the index database of FAQs, FAQ Finder uses a technique similar to the one described for the Mulder system. Only FAQs with words similar to the ones appearing in the question are considered. In a first approach, the question and answer pairs were compared according to a vector of so-called frequency times log of inverse document

[2]http://www.ask.com

frequency values. This measure is similar to the inverse document frequency measure employed by Mulder and helps to assign higher weights to very specific terms. After the FAQ Finder determined the question answer pair whose question is closest to the original user question, the corresponding answer is returned. This approach yielded fairly good results, however, the system exposed some weakness since it was not yet able to draw simple conclusions based on synonyms and the word semantics. Extending the word matching to synonym matching is quite simple since the Wordnet knowledgebase provides the necessary synset concept already. Quillian's marker-passing algorithm was used to include some of the sentences' semantics into the similarity computation. The similarity of two terms is computed by following links within Wordnet that bring us from the first to the second concept. The term husband is related to wife through their common hypernym spouse, for instance. The authors opted to implement these additional measures. They boosted the recall from 58% with the statistical score system to 67% with the combined method.

The TREC Competition A series of Text REtrieval Conferences (TREC) was devoted to working on the problem of finding information in large data repositories. In order to benchmark the results, a test set has been defined. Based on a large collection of newspaper articles, trivia-like questions, such as "Who invented the paper clip?" need to be answered by candidate information retrieval systems. In 2001, over thirty systems took part in the competition. The following procedure seems to be the standard way of approaching the problem. First, the question is analyzed and the matching documents and passages retrieved. Then, the passages are compared and matched against the question before the results are ranked in a final step. These steps and techniques are similar to the ones presented for the Mulder or the FAQ Finder systems. A common property also seems to be the use of a question type taxonomy and leveraging Wordnet for a shallow semantic analysis. Zheng's AnswerBus [144], a system developed by Harabagiu et. al. [78], and the Webclopedia project [83] are representatives of the set of question answering systems having these common features and this somewhat standard architecture.

OntoSeek The use of Wordnet is not only restricted to question answering systems. Guarino et. al. propose a system called OntoSeek that uses Wordnet for more precise Web searches [73]. When operating on a restricted dataset such as a product catalog or an Internet directory, Wordnet can be used to increase recall by exploiting synonym relationships and to increase precision by disambiguating keywords based on the linguistic context. The system was developed as follows. In an initial step, Wordnet was merged with the Sensus ontology[3]. With the help of a user interface, the resources such as catalog descriptions were converted to conceptual graphs. Those graphs contain simple statements such as that the tubing is part of a mountain bike. The authors chose this relatively simple approach since the use of a linguistic ontology was a

[3]http://www.isi.edu/natural-language/resources/sensus.html

major design goal. Representing a resource as a graph then reduces the problem of content matching to ontology-driven graph matching. The search interface allows the user to enter a query. The system immediately uses the underlying ontology to disambiguate search keywords. The query is then also represented as a conceptual graph, allowing the resource descriptions established in the encoding phase to be matched against it.

One of the major findings of this work is that even though the Wordnet taxonomy does not follow some of the basic ontological modeling principles one of the authors describes in [75], this drawback is far outweighed by the sheer size of the Wordnet body and its broad domain coverage.

3.2 Intelligent User Interfaces

The area of intelligent user interfaces is also very much related to our research since it shows approaches and ways how the user's tasks can be supported using intelligent techniques. We believe that an intelligent user interface is a much-needed feature for an online learning application. Often, it is not only the massive amount of information that creates a burden for the learner, but also the ever-growing complexity of today's applications. To briefly illustrate this wide domain, we pick out three examples in the following paragraphs. Maybury and Wahlster provide a comprehensive overview of intelligent user interfaces in [108].

User and Discourse Models Establishing user and discourse models is a central aspect of intelligent user interfaces. Wahlster identifies them to be necessary prerequisites for "identifying the objects to which the dialogue partner is referring, for analyzing a nonliteral meaning and/or indirect speech acts, and for determining what effects a planned utterance will have on the dialogue partner." [143]. Wahlster's XTRA system was applied for assisting users with filling out tax forms. It maintains a model containing, among other information, the user's level of expertise, which influences the system's output. For instance, the user being unfamiliar with the concept of Employee Savings Benefit causes the system to use a pointing gesture onto the tax form rather than using the term directly.

Cased Based Browsing Hammond et. al. proposed so-called FindMe agents [77]. These agents use case based reasoning to help a user select products. One of the examples given is choosing a video to watch where the choice of about 20,000 titles is enormous. The rationale is that if a user liked a certain movie, chances are that she or he will like another movie with similar attributes. The authors found the approach to work best is relatively unstructured domains. If users can easily evaluate an item, i.e. liking or disliking a movie, but have trouble pinpointing and articulating why, traditional database searches tend to fail. In a way, this can also be seen as a document retrieval or search process.

Obviously, the difference is that results are not directly returned after sending a query. The results are rather the product of a longer browsing process.

Intelligent Classroom Another aspect of intelligent user interfaces, completely unrelated to search and retrieval, is demonstrated by the Intelligent Classroom project. In [59], Franklin and Hammond describe a system that aims at minimizing the number of human-machine interactions that are required to arrive at a certain goal. Examples are dimming the lights when course material is being projected, or in the case of a lecture being televised to a remote site, zooming the camera on the lecturer's writing on the board. The Intelligent Classroom uses a formal representation of concepts and actions going on in a classroom setting. Using this model, a process manager is implemented that is able to recognize plans in order to anticipate what needs to happen next. Obviously, the actions carried out by a lecturer do not always follow a predefined model. To cope with this, the system incorporates a history of events. Once the plan is recognized, the actions are simply looked up in a handcrafted knowledge based. This approach is perfectly reasonable, since the domain is small enough. In addition, deep reasoning about actions performed is not required unless a more complicated behavior is to be implemented.

3.3 Agent Systems

Agent systems are interesting to our line of research in many ways. The introduction outlined how it can be very beneficial to query different repositories depending on the situation a user is is when seeking assistance. On the one hand a large repository is more likely to contain a suitable document, however, the effort required to find it could be immense. On the other hand, a fellow user might have just encountered a similar problem and he will be much more likely to be able to help quickly by incorporating a common context into the search process. This idea implies that there should be a large set of distributed repositories that are managed individually and serve different purposes. This heterogeneity can be of great advantage when a suitable solution for a specific case is sought. The organizationally different repositories can even compete against each other when a certain reward system is put in place for providing suitable help. Obviously such a knowledge and service marketplace will need an underlying agent infrastructure to work properly.

A second point is that traditional distributed systems are not designed to scale up for flexible interoperation on the Web. The agent community has always strived for developing a less structured way of communication and collaboration than the one offered with the traditional middleware standards basing on remote procedure calls. Consequently, agents are an interesting topic with respect to data integration. Furthermore, agent technology aims at the distributed execution of business logic that stands in contrast to the rather fixed and predefined workflows of most major IT solutions.

In this section, we will first briefly outline the Foundation for Intelligent Physical Agents (FIPA) agent framework. For our purposes, the major aspect is the logical inter-agent communication protocol. We will see that ontologies also play an important role here. Finally we take a brief excursion from agent technology and present Edutella, a peer-to-peer system developed for online learning.

Agent Frameworks One of the most popular agent platforms is the FIPA standard along with its reference implementation FIPA-OS. The FIPA architecture has three major layers. The framework provides the bottom two layers, namely the communication channel and the agent shell. The communication layer is able to use a variety of middleware protocols ranging from Web Services to the Voyager CORBA implementation. Directory and agent location services are also located in this core layer. The agent shell provides an interface to a pluggable knowledge base component. The FIPA reference implementation, for instance, can use JESS as a knowledge base. In this case the JESS specific commands are wrapped to conform to FIPA. The middle layer also provides session management capabilities that the agent can rely on. The actual agent implementation can leverage the framework and concentrate on the main tasks to be carried out. A developer can access the core functionality via a set of base classes and APIs.

Agent Messaging One major distinction between an agent system and a distributed system is the way the software components communicate. Whereas a distributed system is based on remote procedure calls and therefore requires a predefined interface, agents are more loosely coupled.

Agent frameworks typically cover several different communication patterns. Terms like brokering, mediators, message routers, discovery services, etc. are frequently found in the respective specifications. However, we believe those issues are more implementation specific. The crucial point in agent systems is the question of whether and how agents can communicate and collaborate with minimal a priori knowledge and tuning required by developers which is the main feature distinguishing agents from a distributed system or program.

One of the first specifications of an agent message format was the Knowledge Query and Manipulation Language (KQML). It is a language and protocol for exchanging information and knowledge and is part of the larger ARPA Knowledge Sharing Effort. The second major part of this effort is the Knowledge Interchange Format (KIF). KIF is an attempt to standardize a language for exchanging knowledge and provides for the expression of arbitrary sentences in first-order predicate calculus in a machine-readable format [62]. Naturally, KQML messages can use KIF as a query and assertion format, but other languages can be used as well [54]. Consider the following example using standard prolog:

```
(ask-all
:content "price(IBM, [?price, ?time])"
```

```
:receiver stock-server
:language standard_prolog
:ontology NYSE-TICKS)
```

This message is sent to a stock server. Since both the price and the time variables are unbound, it will obtain all available stock quotes of IBM. The receiver is specified using a generic name rather than a specific address, which is a common approach for all kinds of distributed systems. The ontology label will be discussed in the next section. The FIPA standard also defines a message format called Agent Communication Language (ACL), which is very similar to KQML. One FIPA specification suggests also using KIF as a knowledge content language within ACL messages [55].

Ontologies in Agent Systems Unless agents use natural language to communicate, an approach that would be completely impractical considering today's state of the technology, they need some shared formalism in order to be able to understand each other. If we again draw the analogy to a distributed system, it is the programmers that need to agree on the shared interface. With this common knowledge and the agreement on what a specific call will do, they can implement the client and server accordingly. However, this approach is limited, since a programmer needs to customize the system every time a new component is integrated. Agents claim to be interoperable without human intervention. A message like the one above, sent to an agent that the sender has never seen or dealt with before, should be understood. The last line of the message shows that, besides both agents supporting the KQML specification, the common ground is that both agents understand the NYSE-TICKS ontology. More specifically, they agree on the meaning of the price relationship and they also both agree that "IBM" denotes a stock symbol at the New York stock exchange. The agents can only communicate if this prerequisite is fulfilled. This view of an ontology being a key enabler for interoperable systems is supported by many authors [99, 16].

Ontolingua Ontolingua extends the basic KIF format with primitives for defining ontological constructs like classes and relations, allowing organizing knowledge in object-centered hierarchies. Ontolingua translates this generic representation to other languages such that the knowledge can be stored and reasoned about using existing engines for the target languages [71]. Furthermore, an API for accessing and manipulating a knowledge base, called Open Knowledge Base Connectivity, is defined [53]. Even though the Ontolingua system was exposed as a web application, the ontology library is not too broad. Nevertheless, one can argue that a lot of the ideas that are promoted in today's Semantic Web initiative originally came from the Ontolingua project.

Edutella An interesting alternative to traditional agent systems has been motivated by the recent success and stir caused by peer to peer (P2P) networking systems [57, 109] like Napster or Gnutella. The typical P2P application organizes the large base of participants' workstations to form a gigantic virtual

distributed fileserver. Usually, multimedia files like songs and movies are being shared in today's P2P networks. The Edutella project[4] aims at extending this paradigm to teaching content. Besides the P2P architecture, the handling of metadata is the primary focus. Edutella bases on existing metadata standards like SCORM, already mentioned in section 2.4.3, but uses RDF as an exchange format. In order to be able to access content within this distributed network, a metadata query facility is of central importance. Therefore, the RDF Query Exchange Language (RDF-QEL) has been developed and multiple implementation options have been suggested [116]. Edutella also aims at reusing existing software components in the P2P area, most notably Sun's JXTA project which provides a highly useful Java-based P2P construction framework.

3.4 Knowledge Management

The last aspect we want to investigate in the related work chapter is the area of knowledge management. Within this arguable quite wide description we shall look at the formal aspects that are more closely related to knowledge representation. Online learning and help systems definitely also cover issues of knowledge management. After all, such systems can be classified as knowledge management tools, i.e. systems that support the knowledge transfer from the knowledge engineer or expert to the end user. The following sections will describe two projects carried out at the German Research Center for Artificial Intelligence. The KnowMore and FRODO projects are closely related to our work since they emphasize proactive help features for users and focus on the distributed character of knowledge management. Finally, the Haystack project is introduced which originally introduced the idea of establishing a collection of so-called per user information environments and searching those within a community of users.

KnowMore The KnowMore system aims at supporting users during steps of a workflow by introducing a so-called organizational memory [1]. The paper gives the example of acquiring a customer project. During the different workflow phases from the initial contact via a phone call all the way to signing the contract, the KnowMore systems proactively suggests documents that might be helpful. An example of such functionality is a yellow page directory of competences within the organization being displayed when the offer to the customer is written. Another scenario could be a log of interactions with the customer being made available for reference upon the next interaction. The system is implemented as follows. The workflow steps' information needs are modeled as queries, together with processing rules on how the results of these queries should be displayed. The required input data for the queries comes from the workflow system context as well as many other data sources like documents, databases, or business process descriptions. Logic-based modeling of structure and metadata

[4] http://edutella.jxta.org/

using domain and enterprise ontologies is used to provide a uniform knowledge description. This enables the system to be able to perform intelligent information retrieval on the data. The example given is a search heuristic stating that people working on a project that uses a certain technology are likely to be competent in this technology.

Frodo The FRODO project refines the developments of the KnowMore project by emphasizing the distributed and heterogeneous nature of knowledge management. Rather than having a single organizational memory, a collection of independently designed organizational memories is connected using an agent platform. The agents perform deduction processes in a distributed fashion. At the core of the FRODO agents sits the TRIPLE inference engine, described in more detail in section 5.5.3. This allows a declarative representation of the agent's knowledge. The agent's reactive behavior in response to outside events is modeled via reaction rules. The inference engine and the declarative agent specification are then wrapped inside Jade, which is a FIPA compliant agent framework. Besides the design goal of basing FRODO on agent technology and Internet standards, the development of a toolkit for the development and maintenance of domain ontologies has also been identified as a key requirement. FRODO currently is work is progress. The information summarized here comes from the requirements analysis and system architecture document [142].

Haystack The Haystack project was started in 1997 at MIT [2]. The aim of the project is to design a digital information retrieval system that behaves less like a library and more like a personal bookshelf. On a very course level, the strategy works as follows. First the system gathers information and stores it in a very general datamodel. During the second step, the system tries to gather information about the user by observing how the user accesses the corpus. The collected material is also processed offline in order to further analyze it. Finally, the haystack adapts its data and retrieval processes trying to mimic actions observed from the user. The paper gives the following example. A user might query the system for a specific keyword trying to find a paper she or he has read recently. Once the paper is found by Haystack, the system returns a link to it as well as a link to a colleague who originally sent the paper via email.

The example outlines the value of cross-linking seemingly irrelevant information. Data is gathered by looking the user over the shoulder and recording which documents are opened or which email is sent to whom. Consequently, very little user effort is actually required. After a while, every user will have collected a large pile of data. This is where the literal name, a haystack of information comes from. The theory says that it will be possible to draw useful conclusions from the data gathered. Recently, the Haystack project was picked up again in the context of the Semantic Web [86] by introducing RDF as the storage model. The authors developed a new processing language for manipulating and creating semi-structured data called Adenine.

3.5 Similarities and Differences to our Approach

We will conclude this chapter by identifying the similarities and differences of the related work presented to our approaches and ideas.

The challenge commonly faced in information retrieval and question answering projects is the correct interpretation of natural language. This approach has advantages and disadvantages. On the one hand, it is obviously a great plus to be able to basically leverage the entire Internet indexed in a search engine like Google. The Mulder project is a representative having this advantage. With relatively little development and no user effort in terms of entering and maintaining metadata, a quite powerful system could be created. However, commercially these ideas are yet to be adopted. One can only speculate about the reasons. The most likely explanation is that by basing on natural language, too many errors and misinterpretations are introduced into the system, causing the added value for the user to decrease.

Tim Berners-Lee addresses this issue. He argues that "instead of asking machines to understand people's language" a solution should "involve asking people to make the extra effort" [10]. The extra effort mentioned refers to machine-readable metadata created by humans. This is the basic philosophy of the Semantic Web activity, explained in detail in chapter 5, which we base our work on. Nevertheless, the ideas presented by the question answering community are very important to get the Semantic Web jump-started and to get over the chicken and egg situation regarding the lack of data and the lack of Semantic Web applications we are witnessing at the moment. We will elaborate more on this issue in our RDF survey section 6.2.

Several ideas can be drawn from the Intelligent User Interface community, especially with respect to desirable system behavior. However, we believe that it is crucial which data and which intelligent algorithms are used to implement certain features. This point differentiates our work from the approaches mentioned, since we focus very much on ontologies and formal inferencing on available metadata.

The agent systems introduced provide on major insight for us; namely the importance of ontologies for system interoperability. This is only one of many aspects of platforms and standards like KQML and FIPA-OS. They also deal with several middleware can communication protocol issues, which we think are not too important on a conceptual level. The distributed query facility developed in the Edutella project is an important base, which can be reused in our applications. Our work focuses on one further aspect in particular, the declarative specification of software agents on the basis of the shared ontology. Therefore out work complements the pervious efforts on agent systems that were outlined in this chapter.

The projects listed in the knowledge management section arguable are closest to our ideas. The haystack system originally inspired us to look into distributed and more customized solutions. However, the technical realizations are quite different. The FRODO project is closely related in terms of technology. FRODO also bases on Semantic Web standards. Further similarities are also present on

an application level. In particular, FRODO and our work share the ideas of pro-active information presentation and the distributed character of the system. However, the FRODO project is still in progress and operates on a much larger scale with the focus being on workflow systems. On a technology level, FRODO bases on TRIPLE. Our solutions have a different scope in that they are less powerful and instead base completely on mainstream components in order to simplify the deployment.

Chapter 4

Requirements for a Unified Approach

The initial application scope of our research has been outlined in the background chapter. We also discussed which aspects are already covered by related work in the area, as well as what problems were faced. Using this information, in this chapter we want to list a series of major design goals we think are crucial for the successful development, maintenance, and most of all adoption of future artificial intelligence applications.

4.1 Sharing and Reuse of Content and Knowledge

A first requirement can be drawn from experiences in the online learning systems community. In the late 1990ies, a lot of research funding went into pushing online learning technology in the hope to successfully establish a solid information technology infrastructure for the knowledge society. Despite these enormous funds, the initiative suffered from a quite fundamental problem. Most research projects aimed at developing an online learning platform with document management, authoring, and online learning collaboration features. Unfortunately, the results were hundreds of mostly incompatible systems. Since most of the effort went into development of software, the content fed into the respective tool was typically very limited. This is quite natural, since authoring content within the framework of a research project is definitely a less interesting issue. On the other hand, a system with hardly any content in it is not of much use to the student.

The same argument can be made for just about any application. Consider expert systems. Figure 4.1 shows a screenshot of the printer troubleshooting expert system built into the Windows operating system. While this is a nice feature to have, applications like those are seldomly found in the IT world. We

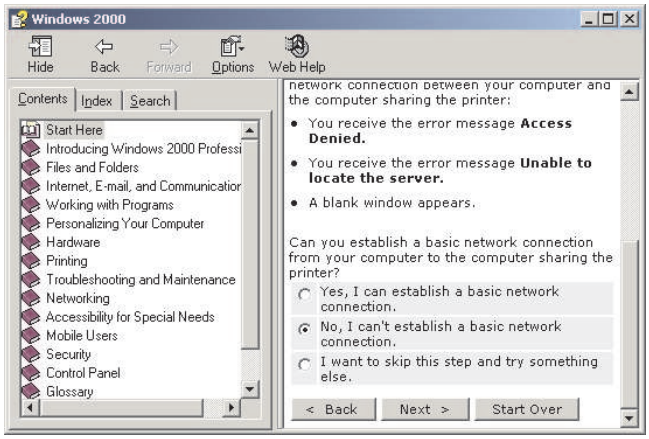

Figure 4.1: A simple printer troubleshooting expert system in Windows.

think the reason for this is that the limited usefulness of such a system usually does not justify the cost incurred by its development. It simply does not make sense to start from scratch and develop such an assistant for all kinds of problem cases.

Both examples, the lack of content and scope in online learning and in help systems, are an indication that any successful approach must be based on sharing and reusing content. It is quite obvious that a single group of authors and developers can never create a system large enough to be beneficial to many users with various backgrounds. The online learning community has also realized this key issue and addresses it with various standardization efforts such as the SCORM metadata standard. In essence, SCORM allows exchanging a predefined set of metadata between systems. Therefore, it is somewhat like a B2B standard for the online learning industry. We believe that this is a step in the right direction. However, more flexibility will be necessary. To us, the Semantic Web initiative is a prime candidate for enabling sharing and reusing not only documents and metadata, but also knowledge about the respective domains. Only if such knowledge can be encoded and exchanged, will applications be able to make smarter decisions, for example, when to present a piece of content or which sequence to present a group of learning modules in. Also, the issues regarding the meaning of classification information outlined in section 2.4.3 and the requirement to have an extensible metadata format naturally lead us to the Semantic Web and ontologies.

4.2 Ontology as the Basis

From the prerequisite of a system having to be based on sharing and reuse, we can directly infer the next requirement. Two parties can always exchange data if

they agree on a certain syntactical datastructure for the transfer. However, they can only really share the data if they have the same understanding of it. Like we outlined in section 2.1, an agreed upon ontology is a key enabler for sharing and reuse. We even want to go a step further. In our opinion, the shared ontology should be at the core of the entire system. This has two advantages. First of all, the use of the ontology is maximized, justifying the high development and maintenance cost. Secondly, it brings a high level of consistency to the overall system. Currently we often find a situation, where the individual components of a system are based on different knowledge bases. For instance, a natural language processing component might use Wordnet whereas the KQML-based agent communication middleware uses another knowledge terminology. The ideal situation would be to base all system modules on the same ontological basis. In a way, this can be compared to a large variety of applications, dealing with task ranging from online transaction processing to analytical data mining, all accessing the same logical database.

Having stated the requirement of an ontology forming the base of several intelligent applications, it also must be clear, that the development of such ontologies must be performed in a collaborative fashion. If the ontology should be a shared, agreed upon representation of a conceptualization, it is only natural to have the ontology established by a group of people. Section 6.4.1 briefly describes some of the currently available software solutions for collaborative ontology design.

4.2.1 Collaborative Ontology Engineering

Holsapple and Joshi provide five basic approaches for ontology design [85] that provide valuable insight on the best way on how to accomplish sharing and reuse on the ontological level. The *Inspirational Approach* characterizes an individual writing an ontology in an ad-hoc fashion. This approach usually lacks a theoretical underpinning and often causes problems since the personal views of the engineer strongly influence the outcome. The advantages are obviously the rapid development and the often innovative character of the result. Modeling the ontology according to observations is called the *Inductive Approach*. This yields ontologies suitable for a specific purpose. However, the results usually do not generalize. Establishing some general principles and applying them to an ontology geared to a specific case characterize the *Deduction Approach*. This approach seems quite useful from a theoretical perspective; however, identifying and selecting such principles will be an almost impossible task given the fact that the intuitive approach is already quite hard. If several partial ontologies about a domain are merged in an iterative fashion, this is called the *Synthesis Approach*. The base ontologies themselves will in turn be established using one of the other methods. Apart from some methods like the ones presented by Guarino [74, 75] and Gruber [72] not many quality assurance metrics have been defined for ontologies. Therefore, the merging task will again be quite hard to perform. Since a single person again does the merging, this approach also seems impractical. The authors strongly suggest the *Collaborative Approach*

to be employed. Rather than trying to merge ontologies after the fact of them being engineered, discussions and co-authoring of an ontology by a diverse group of editors is thought to be most promising.

Again, the Semantic Web appears to be the ideal foundation for this design goal. In section 8.6 for instance, we will demonstrate a rudimentary language processing algorithm that bases on an ontology which is augmented with lexical information. Furthermore, section 8.3 describes how agents can be specified in a declarative way using an ontology and a rule set.

4.2.2 Reusing Standard Software Components

Another aspect comes from the software engineering perspective. If a single ontology is the base for several applications, there must be a certain infrastructure for representing, structuring, storing, and querying knowledge. Such an infrastructure will speed up the development dramatically. A negative example for what happens without such tools can be observed in the Thought Treasure natural language processing system[1]. Thought treasure also bases on an ontology stating simple facts of life such as that soda is a drink. The following lines show the ontology representation format:

```
===soda.z//soft#A drink*.z/fizzy#A drink*.gz/
    carbonated#A soft#A drink*.z/
pop,soda# pop*,coke.z/tonic.oz/soda.My/
```

It is quite hard to read, let alone modify, the information given in these three lines. Certainly, a large fraction of the Thought Treasure code deals with basic tasks such as parsing this representation. Obviously, a base ontology stored in a standardized format with a set of freely available utilities and APIs would be very beneficial. An analogy, even though on a much lower level, can be drawn to the situation of everybody inventing their own data encoding formats with their own escape character handling and so on. Here, the invention of XML and readily available SAX and DOM parsers greatly improved the situation.

4.3 Mainstream Technology

During the design phase of a system, decisions need to be made on which third party components should be used. Especially in a more research-oriented environment, we often face a tradeoff between using proven mainstream technologies and more exotic software, which often is the result of very recent research activities. The reasons for adopting mainstream technology are quite obvious: we find well-functioning, polished, and well-documented solutions. The alternatives often offer much more functionality and one is usually able to find a solution that fits the specific needs of the project much better.

[1] http://www.signiform.com/tt/htm/tt.htm

We argue that in case of doubt, mainstream technology should be used. The reason is simple: it is better to have a system with more content than to have one with more features. The World Wide Web is the ultimate example for this argument. The HyperCard system available on the Macintosh platform offered more functionality compared to the simple hyperlink mechanism of today's Web. However, HyperCard suffered from a small content base and was never really a successful product.

Using mainstream technology we can make sure that systems scale on a performance, on a usability, and on an integration level. By usability we mean that people outside a research lab will be able to use the software. Integration refers to the ability to connect a system with other systems. When mainstream tools are being used, chances are that an appropriate off the shelf middleware or bridging software is available.

Even though this view might be controversial, other research groups are adopting the same strategy of sacrificing functionality for scalability and ease of use and engineering. An example is the Karlsruhe Ontology (KAON) tool suite [114].

4.4 Incorporating Various Information Sources

By investing in a shared ontology and an infrastructure for sharing and reuse, an important prerequisite for another requirement is established. We believe that intelligent systems should try to draw from as many information sources as possible. If agents and even legacy information systems speak the same middleware protocol and share an ontology, exchanging information becomes less difficult and developers and researchers can focus on what to do with the data obtained. We will provide two examples.

Incorporating Contextual Information As mentioned in a survey paper by Berzillon [18], leveraging contextual information has been a research issue for several decades already. We believe that contextual information, whether it is gathered from direct user input, an external data source, or by observing the user interactions over time, is crucial in adding value to an application. Among other things, it is the lack of context that makes a computerized help system much less efficient than a human expert.

Web Service Integration Web Services are an emerging middleware platform basing on Internet standards. Section 5.8 will outline the technical foundations in more detail. It enables programmatic access to information that previously was only available via a browser-based interface. Unfortunately not many commercial websites have implemented Web Service-based gateways so far. Therefore, many research prototypes currently wrap an existing website by parsing the HTML output and exposing a programmatic interface to the outside. This causes an engineering overhead and complicates maintenance, since the wrapper often has to be updated when the website design is changed.

The Mulder project homepage, for instance, has been turned off due to this maintenance issue.

With respect to the tool support, the developments in this area are very promising. Web Services are also gaining popularity as an enterprise application integration solution. Therefore, we predict that in the future it will be increasingly possible to query enterprise information systems from intelligent applications via a web service interface.

4.5 Agent Setting

From many of the previously stated requirements, such as sharing and reuse, it is already clear that a solution should be distributed in nature. The question remains whether the overall architecture should be a centrally managed, pure client server variant or whether it needs to be distributed among several organizationally independent entities. We believe that in the spirit of the Internet, the latter architecture should be chosen. The user or intelligent agent should be able to choose from a variety of content providers supporting different points of view and different learning styles, for instance.

4.6 Summary

In this chapter we listed key requirements, we think are necessary for building useful intelligent applications. Summarizing the arguments, the key requirements are sharing and reuse of content, the integration of external data sources, applying mainstream technology, as well as a distributed agent infrastructure. Ontologies play a central role in all of these points. The standards introduced within the Semantic Web initiative are prime candidates as the basis for our work. There definitely is a substantial amount of tool support, which allows us to avoid reinventing the wheel for each application.

Chapter 5

The Semantic Web Initiative

In 1998, Tim Berners Lee, the creator of the Web, introduced his vision of the next generation Internet, basing on recent research results such as the University of Maryland's Simple HTML Ontology Extensions [80]. Since then, the Semantic Web initiative has gotten a lot of exposure within the research community [11, 81, 10, 50] and even in many mainstream computer magazines [145].

This chapter will introduce concepts and the associated mark-up languages of the Semantic Web. We will also compare this technology to related standards from the eXtensible Markup Language (XML) and Web Services communities.

5.1 Overview

As Berners-Lee points out in a technical note[1], questions like "What is the Semantic Web, and how is it going to affect me?", or "When XML gives us interoperability, why do we need the Semantic Web?" are often asked. In this overview section we want to address and answer these questions, before the next section is diving into the technical details of the various mark-up languages and standards.

In the technical note a nice analogy is drawn between the problems one used to face in the pre web era when downloading a document and today's application integration problems. Getting a document from a remote site used to be a major problem. Users needed to telnet into remote systems and learn how to use proprietary library software. The Web changed all this by providing ubiquitous access to documents on just about any topic from anywhere in the world. In principle, the Web did nothing new but it made things a lot easier.

Consider an enterprise with its information systems. Inventory, payroll, or customer relationship management systems often grow over time and are

[1] http://www.w3.org/DesignIssues/Business

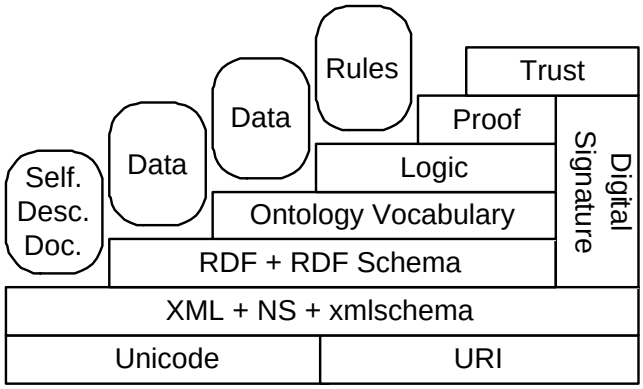

Figure 5.1: The Stack of Markup Languages proposed for the Semantic Web.

therefore poorly interconnected. A similar observation can be made with one's personal data being spread around a calendar, an email client, or a list of Internet favorites. We all have to do a lot of copying and pasting between these applications. On the enterprise level, custom glue code needs to be written for exporting and importing data. XML addresses this problem, however, it only makes it easier to integrate the applications by providing a uniform way to parse and serialize data and by allowing for convenient transformations between different datastructures with XSLT. The glue code still needs to be hand-written. The Semantic Web aims at doing to applications, what the Web did to documents. It provides a web enabled layer or information backbone, into which many diverse information sources can be plugged in. With every source representing information in the same conceptual way, structured queries over all applications are possible. This way, the various systems can cooperate and share information without the need for custom connectors.

As with most Web-related recommendations and standards, the World Wide Web Consortium manages the development of the Semantic Web languages. Figure 5.1 displays the proposed layered architecture. Even though the exact borders between the layers in terms of functionality and responsibilities depend on one's point of view, the figure is well suited to understand the coarse architecture. The W3C aims at working its way up the stack. The encoding layers and parts of the data layer are specified and working drafts exist for the ontology layer. Research groups are addressing the upper layers, however, no official W3C working group has been established for the logic and proof layers yet.

The foundation is built by well-established and accepted Internet technologies, namely Unicode, the Uniform Resource Identification (URI) scheme and of course XML, XML Namespaces, and XML Schema. All layers above make heavy use of these core technologies. For instance, all mark-up languages are subsets of XML.

The data layer allows representing information in an unambiguous way. An

interconnected graph of data is established by using URIs to denote concepts and instances. Different applications can use this data or publish own information via this methodology.

If such a Web enabled data representation approach is to be the basis of data integration, the meaning of globally referenced entities and concept must be specified. This is done in the ontology layer sitting on top of the data layer. As mentioned in section 2.1, ontologies formally represent a shared understanding about a domain. Therefore, they allow interpreting information from the data layer.

The logic layer contains domain knowledge in the form of rules allowing automated reasoning on available data. The idea is to be able to explicitly formalize knowledge, rather than embedding it in program code, which is hard to maintain. A government, for example, could specify and distribute tax laws in formal logic. This would enable an inference engine to establish a tax return based on the rule set and the user's tax data.

The proof layer is supposed to enable more complex agent interactions. Typically, an example from the authentication domain is given. Assume an agent wants to access a protected resource. At first, the document management agent denies the request stating that access is only granted to employees of company X and employees of partner companies. The requesting agent supplies a proof that it is entitled to view the document: it is acting on behalf of person P which is working for company C which is a partner of company X, therefore access must be granted. In conjunction with standard security features such as encryption, certificates, and digital signatures, mechanisms like these are thought to enable the Web of Trust. Note that the current research mostly focuses on topics dealing with issues up to the logic layer. The proof and trust layers are long-term research goals and therefore, they are not included in the following detailed descriptions.

5.2 The Encoding Layer

We start by describing the encoding layer and working our way up to the logic layer. This chapter concludes with an overview over the area of Web Services and its similarities and differences compared to Semantic Web technology.

5.2.1 Unicode

The American Standard Code for Information Interchange (ASCII) introduced in 1968, made it possible to exchange data between different computer systems by standardizing the mapping between characters and numbers. With seven bits available for this encoding, only the most basic characters used in the western world were included in the ACSII standard. This limitation causes serious problems when special characters like German umlauts or French accents have to be stored. A workaround was to use the eighth bit differently for region dependent encodings. The European Union alone requires several different encodings to

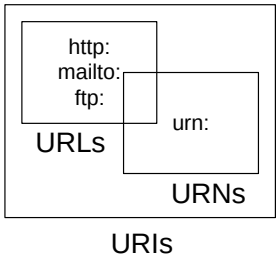

Figure 5.2: Addressing schemes on the Web.

cover all its languages, again causing several problems. A web browser, for example, needs to set the right encoding style when viewing pages from a different region. The Unicode standard is addressing this issue by providing a unique number for every character regardless of which platform, program, or language is used. Today, an increasing number of major software vendors are supporting the Unicode standard. XML also offers support for Unicode characters.

5.2.2 Universal Resource Identifiers

Every Web user is familiar with the Universal Resource Locator (URL) addressing scheme for web pages. The goal of this section is to briefly define the terms Universal Resource Identifiers (URI) and Universal Resource Name (URN) and to explain their relationship to URLs. Figure 5.2 shows that, according to RFC 2396[2], URIs comprise both URLs and URNs. There are also two types of URNs. The first type is location-independent, i.e. it does not point to a specific document. The URN urn:oracle-xsql, for example, is a unique identification of Oracle's XSQL technology. The second type has an institutional commitment to persistence and availability, i.e. the page's location is guaranteed to remain the same.

5.2.3 eXtensible Markup Language

From 1996 to 1998 a workgroup of the World Wide Web Consortium devoted itself to writing the XML recommendation. The key design goals for this new document format focused around the support for a broad range of applications, ensuring a high acceptance rate, and most importantly the simplicity for both users and software developers. This simplicity led to the broad user acceptance and tool support of XML today. Leading software manufacturers such as IBM, Oracle, Microsoft, Software AG, SAP, and others have subsequently created a wide range of XML tools or have integrated XML components in existing products. Today, "XML support" has become a key selling factor of the product. The fact that a whole range of other languages has been developed on top of

[2]http://www.ietf.org/rfc/rfc2396.txt

XML, clearly illustrates that XML has become an important base technology. In this section we will briefly explain the most important issues that are necessary for understanding the other languages. A more detailed introduction can be found in [44].

XML Documents XML documents have a hierarchical structure, much like the chapters, sections, and subsections of this document. This is done via so-called elements, which clearly delimit and label the individual document parts. In addition, attributes can be assigned to the elements. The following example contains contact information data:

```
<?xml version="1.0"?>
<contact type="business">
    <name>Peter Miller</name>
    <email>peter@company.com</email>
</contact>
```

Unlike browsers accepting sloppily written HTML pages, every XML document has to strictly obey the syntax rules in to be accepted by an XML parser.

The features of XML are extremely simple and on first sight, it is not clear how such a simple format can be beneficial. A study conducted by the Gartner Group estimated that 35-40% of the programming budget of a typical company are spent on proprietary solutions that format documents or create lists for exchanging data between different databases and applications. Obviously, a simple, globally used way of writing down information is a great help when dealing with such issues. XML relieves developers from tasks like writing parsers, dealing with special characters and escape sequences, and labeling data.

Namespaces The purpose of namespaces is to be able to use elements from different XML vocabularies in one document. Let us assume there is a document, which contains medical information about patients. All elements containing the patient's contact information can belong to the namespace contact, for example `<contact:telephone>`. Medical information is allocated to the namespace "med" such as `<med:organ>`. The accounting software can now read the document and pick out the contact tags alone, in order to address a letter. However, a search engine for doctors uses only the elements relevant to medicine. Namespaces make it easier to use simple XML software components in any possible scenario, such as the management of addresses. A namespace is defined within an attribute of the root element.

```
<x xmlns:edifact='http://www.edi.org/edifact#'>
    <edifact:price units='Euro'>32.18</edifact:price>
</x>
```

This example defines the edi-namespace. A URI identifies the namespace. Documentation could be found here, however, this is not necessary. The URI simply constitutes a string that identifies the namespace.

In a way, namespaces provide a mechanism for shorthand notations of element names. Consider `edifact:price` which is a shorthand notation for http://www.edi.org/edifact#price. Unfortunately this mechanism cannot be applied to values, a feature that would be nice to have especially in the Semantic Web context. As pointed out in [70], a possible workaround for this is using XML entities defined in a local Document Type Definition (DTD). An entity is denoted by `&entity-name;` and basically serves as a text module. Similar to the macro preprocessor facility well known in the C programming language, every occurrence of the entity within the document is replaced with the text assigned to the entity. Consider the following XML document:

```
<?xml version='1.0'?>
<!DOCTYPE edifact:Example [
    <!ENTITY edifact 'http://www.edi.org/edifact#'>
]>
<edifact:example xmlns:edifact="&edifact;">
    <edifact:element edifact:att="&edifact;att-value" />
    ...
</edifact:example>
```

The local DTD defines the entity `&edifact;` which is used in both the definition of the namespace and within the attribute value. The string `&edifact;att-value` is hereby expanded to `http://www.edi.org/edifact#att-value`. This method allows using the namespace prefix for element and attribute values also. The fact that this awkward workaround needs to be used in some situations is currently subject of active discussion in the RDF and Semantic Web standards efforts.

XML Schema XML Schema allows restricting the structure and vocabulary used in XML documents. The schemata themselves are written in XML as the following small example demonstrates.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="contact">
        <xsd:complexType>
            <xsd:element name="name" type="string" />
            <xsd:element name="email" type="uriReference" />
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

A so-called validator can then check if a document is valid with respect to a schema referenced from the document. This is an interesting feature that simplifies the exchange of documents. Participants of a B2B trading system could collaboratively define a schema. The integrity of incoming documents can then be checked by a standard piece of software. It is important to note that the development of the application is easier if, with a positive answer from the

XML validator, a certain degree of error checking is not necessary anymore. This cuts costs and at the same time makes the application essentially more robust. In this case a similarity to database applications can be seen. Any good database schema, using key references and integrity constraints, will guarantee that the data has a certain level of quality. Considering the example above, you can rely on a contact element having a name and an email field, for example. Technically a document specifies the URL of the schema it claims to be valid against in the `schemaLocation` attribute:

```
<contact xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="contact.xsd">
    ...
```

5.3 Data Layer

The Resource Description Framework (RDF) is at the core of the data layer. We also included an introduction to RDF Schema in this section, even though some people argue that RDF Schema is already an ontology language since it allows to provide webized and extensible definitions of shared concepts and their relationships. Staab et. al. for instance use a small extension to RDF Schema for modeling ontologies [133].

5.3.1 Resource Description Framework

RDF is a framework for metadata and the most basic mark-up language in the context of the Semantic Web [97]. The core idea is that everything is treated as a URI. A person, for instance, could be denoted by her or his homepage. When talking about Ora Lassila, we might use the site http://www.lassila.org; a desk in some office might be referenced via the company's inventory list as http://xyz.com/inventory#K4622-ERF. In the RDF terminology, these things are called resources. It is then possible to make statements about the resources. If Joe is Peter's brother, we could state this as the following subject, predicate, object triple:

```
Subject:   http://www.mit.edu/~joe/
Predicate: http://www.cogsci.princeton.edu/~wn/isBrotherOf
Object:    http://www.mit.edu/~peter/
```

In RDF, this subject, predicate, object triple is written as follows:

```
<?xml version='1.0'?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:wn="http://www.cogsci.princeton.edu/~wn/">
    <rdf:Description rdf:about="http://www.mit.edu/~joe/">
        <wn:isBrotherOf
            rdf:resource="http://www.mit.edu/~peter/" />
```

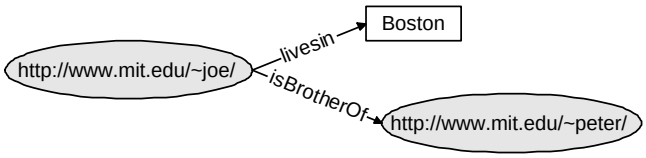

Figure 5.3: Every RDF triple is an arc in a directed labelled graph with resources as the nodes.

```
    </rdf:Description>
</rdf:RDF>
```

Note that the isBrotherOf predicate is a URI pointing to a version of the Princeton Wordnet lexical database project[3]. In Wordnet, a concept brother can be found. Different URIs can specify other meanings of brother, such as monk. Consequently, the pre- and postfix denotes the relationship of being someone's blood brother. Even though this is somewhat clumsy, there is a specific reason for choosing Wordnet as the predicate namespace. Due to the wide acceptance and popularity of Wordnet, we can assume that many agents can correctly interpret our statement.

Consider another example, the statement that Joe lives in Boston. Again we present the subject, predicate, object as well as the native RDF representation:

```
Subject:   http://www.mit.edu/~joe/
Predicate: http://www.schema.org/rdf/livesin
Object:    Boston

<?xml version='1.0'?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:schema="http://www.schema.org/rdf/">
    <rdf:Description rdf:about="http://www.mit.edu/~joe/"
        schema:livesin="Boston"
    />
</rdf:RDF>
```

The first difference is that the predicate comes from another namespace. Secondly, the object is a simple string or literal, rather than a resource. If another statement also uses the string `Boston` as its object, it would be up to the application to decide, if the city of Boston, or maybe a project with codename Boston is meant. Further technical aspects of RDF, for instance various forms of abbreviated XML syntax, can be found on the respective W3C portal site[4].

[3] http://www.cogsci.princeton.edu/∼wn/
[4] http://www.w3.org/RDF

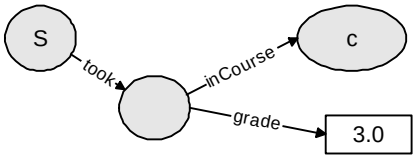

Figure 5.4: RDF uses anonymous intermediate nodes for representing relationships with an arity greater than two.

Since further statements about Joe, Peter, and other resources can be made, we eventually end up with a directed labeled graph. This graph is shown in figure 5.3. The resources are the graph nodes and the statements define the edges.

Non-binary Relations Mapping the binary relations to a directed labeled graph is quite intuitive. The question arises, however, how relations with higher arity are to be represented. This is a quite common case in relational data-modeling for instance. Consider the example of a student enrolling in a course and then getting a grade for it. In this case, enrollment is a ternary relationship, since the grade is neither assigned directly to the student nor directly to the course. The grade is associated to the student-course relation and a tuple $(S, C, 3.0)$ denotes that student S got a 3.0 in course C. RDF uses the so-called anonymous resources for this case:

```
<rdf:Description about="http://.../S">
    <uni:took rdf:parseType="Resource">
        <uni:grade>3.0</uni:grade>
        <uni:inCourse resource="http://.../C" />
    </uni:took>
</rdf:Description>
```

Figure 5.4 shows the RDF graph for this enrollment example. The RDF graph illustrates how the three nodes are connected by binary relations via the anonymous middle node. This modeling approach is also known from relational databases. The ER diagram from figure 5.5 contains an explicit enrollment table, even though this was not mentioned in the problem specification. Naturally, this relation contains the grade attribute. Just like more attributes can be added to this table, more resources can be linked from the anonymous RDF resource to model n-ary relations. There is no clean way to represent a unary relation in RDF. A simple workaround that can be applied is to attach a dummy object such that the unary relation becomes binary again.

Collections Besides resources and literals, RDF also supports collections as an object type. The RDF recommendation specifies bags, alternatives, and

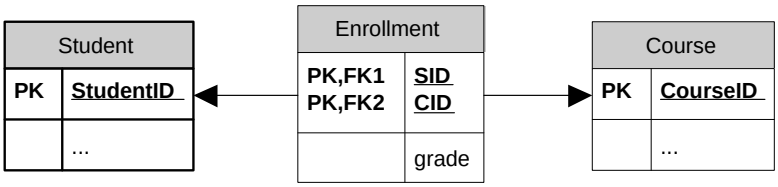

Figure 5.5: ER model corresponding to the RDF graph in figure 5.4.

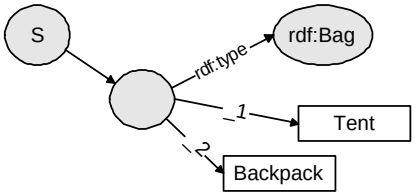

Figure 5.6: RDF addressing schema for the elements of a collection.

sequences. No order is defined in a bag whereas the order does matter in the sequence. If the object is an alternative, it means that the actual single value is one of the items listed in the alternative collection. The syntax is similar to the ternary relation example above. It also uses nested XML elements to denote an intermediate resource, i.e. the bag in the example below. The arcs connecting the items of the collection are assigned the pseudo labels _1, _2, and so on. The corresponding RDF graph is shown in figure 5.6.

```
<rdf:Description about="...">
    <s:bring>
        <rdf:Bag>
            <rdf:li>Tent</rdf:li>
            <rdf:li>Backpack</rdf:li>
        </rdf:Bag>
    </s:bring>
</rdf:Description>
```

Reification With reification, we can use an RDF statement itself as the object of another statement. This sounds somewhat strange at first, however this scheme allows for a powerful feature: one can make statements about statements as depicted in figure 5.7. For instance, consider the referenced statement to be: "Jack is the murderer", and the outer subject and predicate to be: "Detective believes". The entire compound would then be: "The Detective believes that Jack is the murderer". This is definitely an interesting feature, especially considering that RDF fragments might be collected from anywhere on the Web.

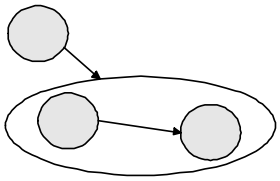

Figure 5.7: RDF Reification mechanism: a statement can itself be the object of another statement.

In these cases it is definitely a good idea to have a native mechanism for the representation of such meta statements. Denzinger provides some nice ideas on how to deal with contradiction, missing, and wrong information when reasoning is done on the Web [31]. Reification would definitely support many of the algorithms that are proposed there.

However, reification is a quite controversial issue. We believe that it is too advanced of a feature to be placed in a language, which is located within one of the basic levels in terms of functionality. Like any other RDF fact, reification statements are also triples. Consequently, they can also be stored in any RDF database. However, an RDF query language as well as its implementation in an RDF storage system should explicitly support reification without leaving the application to interpret the base and reification facts on a pure triple level. A recent survey on RDF storage systems and associated query languages [51] shows that only one out of six RDF query languages, namely TRIPLE, explicitly supports reification. A similar picture can be seen for storage and query engines that implement one of the query languages. Out of fourteen implementations, reification is also only supported by the TRIPLE engine. Detailed information on TRIPLE is provided in section 5.5.3.

Where is RDF stored? RDF is serialized using XML syntax. Like any XML document, RDF might only be a stream of bytes traveling from one application to another via the network. In this case, the stream will most likely be generated dynamically by a web application exposing some dataset in RDF format. This is comparable to SOAP messages containing information coming from a central database, for instance. RDF can also be stored in static files, separately or, if statements about an HTML website are being made, within the head tag of the website. Sean B. Palmer collected several approaches for embedding RDF in HTML on the Infomesh website[5].

Why not use XML? A frequently asked question on RDF is concerned with why RDF is needed on top of XML to begin with. After all, that statement that Joe lives in Boston can be encoded directly in XML, for instance within the following two examples:

[5] http://infomesh.net/2002/rdfinhtml/

```
<contact>
    <id>http://www.mit.edu/~joe/</id>
    <livesin>Boston</livesin>
</contact>

<livesin>
    <who>http://www.mit.edu/~joe/</who>
    <where>Boston</where>
</livesin>
```

One can easily come up with several other variants, all of them making sense to humans. However, to an application, they are just hierarchical datastructures and it is not really clear how the values are related to each other. The simple reason for inventing RDF was to provide a standard way of writing down such simple statements.

Another argument for RDF is the fact that data in any form can be broken down into a representation as a directed labeled graph, as we have seen in the paragraph on modeling non-binary relations. XML on the other hand, supports only hierarchical structures, clearly causing problems when n to m relations are to be represented. If students are enrolled in many courses and many students can in turn attend courses, clearly, a hierarchical representation must introduce redundancy, no matter which way one looks at it. Similar to the RDF addressing scheme of using URIs as identifiers for resources, XML also has the notion of ID attributes that would, in principle also allow to model graphs. This feature, however, is hardly used at all in the XML world.

Notation 3 Notation 3 (N3) is an increasingly popular alternative serialization syntax for RDF graphs[6]. It is motivated by the fact that the XML versions of these graphs are usually quite hard to read. While N3 is basically equivalent to RDF in its XML syntax, it is easier to read and write.

N3 files begin with the declaration of namespaces, identified by the prefix keyword. These are followed by a list of statement groups, which are terminated by periods. Within each group, all statements refer to the same subject specified at the beginning. Each subject s is followed by one or more semicolon-delimited predicate object pairs (p_i, o_i). These are then combined to the statements $(s, p_1, o_1), (s, p_2, o_2), ...$ Along the same lines, a subject predicate pair (s, p) can be followed by several comma-delimited objects $(o_1, o_2, ...)$. This then translates to $(s, p, o_1), (s, p, o_2), ...$ N3 distinguishes between URIs and literals by delimiting them with angle brackets and quotes respectively. The following lines show the N3 version of the example from the RDF section. The notation is more compact and also easier to read.

```
@prefix wn:  <http://www.cogsci.princeton.edu/~wn/> .
@prefix sc:  <http://www.schema.org/rdf/#> .
```

[6] http://www.w3.org/DesignIssues/Notation3.html

```
<http://www.mit.edu/~joe/>
    wn:isBrotherOf  <http://www.mit.edu/~peter/>;
    sc:livesin      "Boston".
```

N3 also has a rules part to it which will be outlined in section 5.6.2

5.3.2 RDF Schema

RDF Schema (RDFS) takes the logical next step and provides a type system for RDF graphs. It allows defining a class inheritance hierarchy, with multiple inheritance being explicitly supported. Following the principle that URIs represent everything from simple instances to complex concepts, they are also used to identify classes. RDFS also uses the subject, predicate, object triple notation. The following statements, for instance, 1) declare the class student as such, 2) declare the class student as a subclass of person and 3) identify Joe as an instance of the class student:

```
Subject:   http://www.mit.edu/types#Student
Predicate: http://www.w3.org/1999/02/22-rdf-syntax-ns#type
Object:    http://www.w3.org/2000/01/rdf-schema#Class

Subject:   http://www.mit.edu/types#Student
Predicate: http://www.w3.org/2000/01/rdf-schema#subClassOf
Object:    http://www.schema.org/rdfs/types#Person

Subject:   http://www.mit.edu/~joe/
Predicate: http://www.w3.org/1999/02/22-rdf-syntax-ns#type
Object:    http://www.mit.edu/types#Student
```

The strength of this mechanism is the ability to cross-reference other schemata by exploiting the possibility to point to any class defined in any other schema via a URI mechanism. The example is a fictious schema about students developed at MIT. The second statement defines the class student by `subclassing` person, which is defined in a general schema from a higher-level entity. As far as the syntax is concerned, RDF Schema is simply a set of RDF statements that use schema vocabulary like `type`, `subClassOf`, or `Class`. The explicit RDF serialization is therefore omitted since it works exactly like the example shown in the RDF section above.

Besides class and instance definitions, RDF Schema also allows for the definition of predicates. The example above uses predicates from the given W3C namespaces. The examples of the RDF section above, however, referenced predicates like livesIn. In RDF Schema, custom predicates are defined by specifying the predicate domain and range. The signatures of `livesIn` and `brotherOf` could be (`Person livesIn Literal`) and (`MalePerson isBrotherOf Person`). Note that literal denotes any kind of string, i.e. no URI. The following lines provide the definition of the `isBrotherOf` predicate in RDF notation. In order to abbreviate the URIs, assume that the entity &wn; is as-

signed to http://www.cogsci.princeton.edu/ wn/ and that the entity &rdf; is http://www.w3.org/1999/02/22-rdf-syntax-ns#.

```
<rdf:Description about="&wn;isBrotherOf">
    <rdf:type resource="&rdf;Property"/>
    <rdfs:subPropertyOf rdf:resource="&wn;isSiblingOf"/>
    <rdfs:domain rdf:resource="&wn;MalePerson"/>
    <rdfs:range rdf:resource="&wn;Person"/>
</rdf:Description>
```

The first line specifies the subject URI. In the second line, `isBrotherOf` is defined as an RDF property. The `subPropertyOf` keyword allows establishing a hierarchy of properties, just like `subClassOf` forms a hierarchy of classes. The semantics are quite simple. If p is a `subproperty` of q, then $q(A, B)$ implies $p(A, B)$. Finally, the last two lines within the description element defines that only male persons can be brothers of persons.

Possible Applications One can already think of several useful applications on the basis of RDF and RDF Schema. A frequently cited aspect is data integration. One simple prototype described on xml.com generates and integrates RDF metadata of email messages and an address book[7]. While this is definitely no breathtaking functionality, the article demonstrates that with a few lines of code and some readily available RDF tools, a working system can be created very quickly. Another application by Decker et. al. deals with Web searches[29]. RDF metadata is collected and stored in a central repository. Using a simple inference engine, searches can be made more effective by leveraging synonyms and including pages within a subcategory in a search for a broader category. Again, the system's architecture is quite simple and no major engineering effort was necessary for its implementation.

Concept Maps RDF graphs often remind people of concept maps or related topics, which are typically used to capture an expert's view of a domain. This is done in a very unformalized way by basically drawing a graph with important terms as the nodes and linking those with linguistic associations. Figure 5.8 shows an example.

In s strict sense, concept maps cannot be considered a form of knowledge representation, since they do not have defined semantics and no clear logical syntax. The application of these structures is mainly within classroom applications. By underpinning a set of documents with a concept map and by providing hyperlinks from the map to the individual documents, students were found to learn important concepts much faster [21]. Carr et. al. use similar methodology to generate hyperlinks for a given document [22]. In a more general sense they can be seen as a tool for aiding inter-personal collaboration.

Techniques similar to Concept maps are topic maps, concept spaces, or mind maps. What is interesting from a Semantic Web point of view is that there

[7] http://www.xml.com/pub/a/2000/08/09/rdfdb/

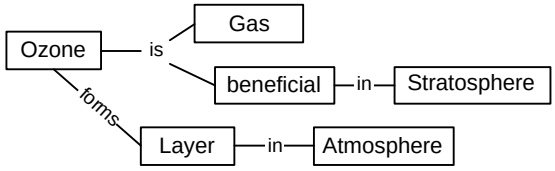

Figure 5.8: A concept map on Ozone.

actually is an ISO standard and an associated XML DTD for serializing topic maps[8]. However, concept or topic maps should not be confused with RDF. RDF can be a vehicle for representing them. This is, however, the only direct relationship.

5.4 Ontology Layer

The ontology layer arguably is the central and most important piece in the Semantic Web framework. It provides the vocabulary for the data layer and it is also the base for the logic layer in that rules refer to concepts and relations defined in the ontology. The goal is to provide a language that can be used for applications that need to understand the content of information instead of just understanding the human-readable presentation of content. Compared to RDF Schema, ontology languages provide additional vocabulary for describing concepts and relations.

Two major representatives can be found in this section, DAML+OIL and the recently published follow-up, the W3C Web Ontology Language.

5.4.1 DAML+OIL

As the name suggests, DAML+OIL[9] is the joint result of two originally separate projects. The DARPA Agent Mark-up Language (DAML) program formally began in August 2000. The goal was to develop a language on top of XML and RDF that would provide a powerful way to describe objects and their relationships to other objects [82]. The Ontology Interchange Language or Ontology Inference Layer (OIL) project[10] in turn is a similar effort funded by the European union [52]. Both projects quickly joined forces and released the first joint specification at the end of 2000.

Technically, DAML+OIL bases on description logics. A specific description logic is mainly characterized by a set of constructors that allow to define complex classes and roles from basic ones. The classes are hereby treated as sets of objects. The roles in turn are binary relations between objects.

[8] ISO 13250, http://www.topicmaps.org/
[9] http://www.daml.org
[10] http://www.ontoknowledge.org/oil/

Class Constructs DAML+OIL picks up some of the commonly used description logic constructors, such as `disjointWith`, which identifies that a class has no instance in common with another class. Consider the following example taken from the DAML+OIL website[11] where the class `Female` is defined to be a subclass of `Animal` and to be disjoint with the class `Male`:

```
<daml:Class rdf:ID="Female">
    <rdfs:subClassOf rdf:resource="#Animal"/>
    <daml:disjointWith rdf:resource="#Male"/>
</daml:Class>
```

Along the same lines, a class `TallMan` can be defined as the intersection of the classes `TallThing` and `Man`:

```
<daml:Class rdf:ID="TallMan">
    <daml:intersectionOf rdf:parseType="daml:collection">
        <daml:Class rdf:about="#TallThing"/>
        <daml:Class rdf:about="#Man"/>
    </daml:intersectionOf>
</daml:Class>
```

Note that the class inheritance mechanism is taken from RDF Schema, however, the new `disjointWith` and `intersectionOf` constructs come from the DAML namespace set to http://www.daml.org/2001/03/daml+oil#. Other constructs that define the relationship between two classes by making statements about their instances are `unionOf`, `disjointUnionOf`, `intersectionOf`, `complementOf`, and `sameClassAs`. Besides by referring to other classes, properties can also be used to construct more complex classes. Consider the class `TallThing` being defined as things whose `hasHeight` property has the value `tall`:

```
<daml:Class rdf:ID="TallThing">
    <daml:sameClassAs>
        <daml:Restriction>
            <daml:onProperty rdf:resource="#hasHeight"/>
            <daml:hasValue rdf:resource="#tall"/>
        </daml:Restriction>
    </daml:sameClassAs>
</daml:Class>
```

The example defines `TallThing` to be equivalent to an anonymous class defined by the restriction. This anonymous class is the class of all things that satisfy the restriction of having `tall` as the value of the `hasHeight` property. Since `TallThing` is equivalent to this anonymous class, the restriction will hold for all members of `TallThing`. Note that using the `sameClassAs` construct is semantically different from `subClassOf`. The `subClassOf` relationship, on

[11] http://www.daml.org/2001/03/daml+oil-walkthru

the one hand, allows the following conclusion: if $subClassOf(A, B) \wedge (a \in A)$, then $(a \in B)$, i.e. every `TallThing` must also have `tall` as its height, however, things with `tall` height do not necessarily have to be a `TallThing`. The `sameClassAs` construct, on the other hand, allows to defer the following statement: if $sameClassAs(A, B) \wedge (a \in A) \wedge (b \in B)$, then $(a \in B) \wedge (b \in A)$, i.e. the set of tall things is *exactly* the set of things with `tall` as the value of the `hasHeight` property.

Property Constructs DAML+OIL provides additional constructs to provide the ability to further describe properties defined in RDF Schema. The `ObjectProperty` and `DatatypeProperty` allow distinguishing between properties that have objects and primitive datatypes as their range. Rather than simply denoting all data values as literals as it is done in RDF, DAML+OIL references the rich set of primitive data types defined in XML Schema. For instance, the property `shoesize` would be defined as a `subPropertyOf DatatypeProperty`. Another construct, the `UniqueProperty` implicitly defines the cardinality to be exactly one. If P is a `UniqueProperty`, then $P(x, y)$ and $P(x, z)$ implies $y = z$. The `UnambiguousProperty` has a similar effect for the property's domain: if $P(x, y)$ and $P(z, y)$ then $x = z$. Like its name suggests, subclassing from the DAML+OIL `TransitiveProperty` marks a property as transitive, i.e. $P(x, y)$ and $P(y, z)$ implies $P(x, z)$. Finally, the `inverseOf` construct allows to define a property as the inverse of another property. `WorksFor`, for instance, could be defined as being the inverse of `employs`. Furthermore, a property's cardinality can be set to an exact range with `cardinality`, `minCardinality`, and `maxCardinality`.

Note that this section does by no means provide a comprehensive list of DAML+OIL features. For instance, it is possible to define custom data types by extending the basic XML Schema definitions and to define equivalence between classes. The most important concepts, however, have been outlined.

5.4.2 Web Ontology Language

The Web Ontology Language (OWL) is currently being designed by a W3C Web Ontology Working Group. It can be considered the successor of the DAML+OIL effort, since DAML+OIL was officially turned in as a submission to W3C in September 2001. Therefore, the goals of OWL remain the same. One feature of OWL is notable, however. It is the introduction of OWL Lite, a simplified version of the language. The idea behind OWL Lite is that the step from RDF and RDF Schema to OWL might be too big for tool developers. OWL Lite is thought to have a useful set of base features that are easy enough to implement. This is an important point, since the availability of tools is a necessary prerequisite for successful adoption of standards.

OWL Lite uses a subset of the full OWL language constructors. Classes can only be defined in terms of named superclasses, i.e. the use of anonymous classes defined by restrictions is not possible. Similarly, property restrictions in OWL

Lite use named classes. Furthermore, the cardinality mechanism is limited to cardinalities of zero and one.

5.5 Query Languages

Even though query languages are an integral part of any data processing environment, there are no established standards yet. Several approaches have been proposed up to now. According to Karvounarakis et. el.[88], these can be roughly grouped into the following categories: Some languages like RDFPath emphasize the graph structure of the RDF data and use mechanisms similar to XPath[12], a query language originally designed for selecting information from XML trees. Other alternative proposals like SquishQL borrow many syntax elements from the structured query language (SQL) and view RDF data as a collection of statements. Other approaches base on the object query language (OQL), Lisp (VERSA), and Frame Logic (TRIPLE). Work on a query language for DAML, the DAML Query Language (DQL), is currently work in progress. An initial abstract specification has been published[13], however no formal syntax is defined yet.

Since it is nearly impossible to list and explain all query approaches, the sections below focus on three representatives. We chose RDFPath, RQL, and TRIPLE for reasons of popularity and in order to show approaches that are quite different in nature.

5.5.1 RDFPath

RDFPath picks up the graph traversal ideas also found in the XPath XML query language, the Lore system[111], and related work in the area of semi structured databases. Starting from a so-called primary selection of resources in the RDF graph, location steps identify a set of resources, which are reachable via one step from the starting set. The location step `child()` denotes all children, i.e. resources linked via any arc. Similarly, `child(dc:creator)/child(vCard:FN)` would be the set of resources reachable from the context via an arc labeled with `dc:creator` and a second arc labeled with `vCard:FN`. Other location steps include constructs for identifying elements of RDF containers and the property labels themselves. Finally, filters can be used to restrict the possible paths that can be taken: `[child(dc:creator) = "Karl Mustermann"]` selects all resources of a given set of nodes that have a `dc:creator` child with the string representation "Karl Mustermann". The concepts and the syntax are very similar to XPath. Of course the difference is that instead of trees, which are hierarchical directed labeled graphs, general directed labeled graphs are being traversed.

[12] http://www.w3.org/TR/xpath
[13] http://www.daml.org/2002/08/dql/dql

5.5.2 RQL

RQL was developed within the On-To-Knowledge EU project [88]. It uses SQL-like query expressions like the following:

```
select T
from Painting{X}.title{T}, {Y}created{C}
where X=Y and C>"1930"
```

Several syntax elements are being displayed here. The term `Painting{X}.title{T}` denotes that the titles of all resources of type painting should be selected. The second term in the from clause `{Y}created{C}` is joined. Since `X` and `Y` must have the same value, this is an equi-join. The tuples can be filtered further by specifying conditions like `D>"1930"`.

RQL bases on RDF Schema semantics with respect to its type system. Hence, if a class oil painting were defined to be a subclass of painting, the oil paintings would also be returned by the query. A potential RQL implementer would have to take care of the transitivity of the binary predicate `subClassOf`, which RQL inherits from RDF Schema.

5.5.3 TRIPLE

TRIPLE is a very interesting project by Stefan Decker and Michael Sintek[14], since in a way it is a query language, a rule language, and also an inference engine. It is the successor of the Simple Logic-based RDF Interpreter (SiLRI) and therefore closely related to Frame- or F-Logic[90]. F-Logic combines the strengths of both the object oriented data model with its rich data modeling capabilities and deductive database languages with their clear semantics and expressiveness.

Similar to N3, namespaces and even abbreviations to lengthy names can be defined as follows:

```
rdf := "http://www.w3.org/1999/02/22-rdf-syntax-ns#".
isa := rdf:subClassOf.
```

Statements are written in the form `subject[predicate->object]`. The first feature that distinguishes TRIPLE from other approaches is the support for models. The plain statement notation can be augmented with a suffix to indicate the model it is in: `subject[predicate->object]@model`. A second unique feature of TRIPLE is its support for reification. The object can be replaced by another statement placed in angle brackets: `s1[p1-> <s2[p2->o2]>]`. The following example shows how F-Logic rules can be embedded. A couple of statements using Dublin core vocabulary are augmented by a rule stating that a document qualifies for a keyword search if it has a `dc:subject` relation to the keyword. The query `search(X, "RDF")` would then bind the resource `dfki:d_01_01` to `X`.

[14] http://triple.semanticweb.org

```
@dfki:documents {
    dfki:d_01_01 [
        dc:title -> "TRIPLE";
        dc:subject -> "RDF";
        ...
    ].
    FORALL S,D  search(S,D) <-
        D[dc:subject -> S].
}
```

Conventional approaches use built-in semantics. This means, a system that is based on the RDF Schema semantics, for instance, is therefore not suitable for Topic Maps or UML. TRIPLE allows different semantics to be adopted via parameterized models. In a parameterized model, the semantics can be specified in a declarative way using F-Logic rules. This mechanism allows the flexibility of adopting different semantics on the fly. The sample rule below defines the semantics of the RDF Schema `subPropertyOf` predicate in a declarative way by stating that if S is a `subPropertyOf` P and some resource O is connected to V via the property S, then they are also connected via the property P.

```
FORALL O,P,V O[P->V] <- EXISTS S S[subPropertyOf->P] AND O[S->V].
```

If the semantics cannot be entirely declared via rules, as it is the case for description logic-based languages like DAML+OIL, external engines can be attached to the TRIPLE system.

TRIPLE is both a query language and an inference system. Therefore, section 5.5.3 will outline the more implementation-oriented aspects.

5.6 Logic Layer

The logic layer is definitely related to query languages, since a query Q can be interpreted as the rule: if Q is true for variable assignment A, then print A. The latest representative in the query language section also being a rule system is a result of this fact.

The rule or logic layer sits on top of the RDF data and the RDFS / OWL ontology layers. Even though no W3C recommendation specifically deals with these aspects, we believe it is crucial for making Semantic Web technology useful. Several authors also support this view [76, 29, 133]. It is the rules that encode common knowledge about an ontology's concepts and actually allow for computations to happen. A simple example would be a rule stating that people living in an US city also live in the US itself. For humans, this is a trivial conclusion to make, however, it is not trivial at all for a computer program. The simple rule $livesIn(x, z) \leftarrow livesIn(x, y) \wedge partOf(y, z)$ captures such basic knowledge in a declarative way and allows the machine to defer this additional piece of information.

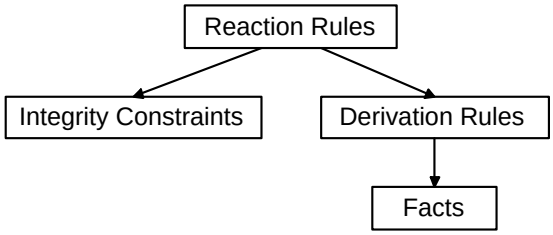

Figure 5.9: The top-level RuleML hierarchy.

In this section we will describe three representatives. First, the RuleML initiative is presented. It aims at developing an interchangeable XML-based syntax for rules. After this, the rule part of the popular Notation3 is shown. Finally, we outline the efforts of the Java Rules committee, which is working on bringing the logic world closer to the mainstream of iterative and object-oriented languages.

5.6.1 RuleML

One of the most promising candidates for rule mark-up standards is the RuleML[14] language. In the spirit of the Semantic Web and the view that an ontology is the *combination* of a backbone taxonomy, axioms, and rules, the RuleML initiative establishes an XML-based structure for exchanging rules. RuleML identifies the rule categories shown in Figure 5.9 and aims at building modular syntactical definitions for them.

- Reaction rules or Event Condition Action (ECA) rules get activated by a certain event, then check their condition, and possibly perform an action. Therefore, general reaction rules can only be executed in a forward fashion, i.e. the sequence of steps just outlined.
- Integrity constraints can be viewed as special reaction rules whose only action can be to flag an inconsistency in the data set. The execution of integrity constraints is the same as for general reaction rules.
- Derivation rules are also reaction rules in that their action is to assert new information if the rule premise is true. Note that RuleML does not prescribe whether this should be done in a forward chaining fashion, i.e. working upwards from base facts to the goal, or backward chaining fashion, i.e. working downward from the goal.
- Finally, simple facts can be viewed as a rule with an empty, i.e. true, premise.

The XML syntax is fairly straightforward to understand. The rule components like atoms, variables, and relations are embedded in the respective XML

elements. RuleML defines a DTD, which allows an application to check the syntactical correctness of the ruleset with a standard XML validation tool. The DTD version is 0.8. In this version, RuleML is similar to datalog. Consider the example given in the section above:

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE rulebase SYSTEM
    'http://www.dfki.uni-kl.de/ruleml/dtd/0.8/ruleml-datalog.dtd'>
<rulebase>
    <imp>
        <_head>
            <atom>
                <_opr><rel>livesIn</rel></_opr>
                    <var>x</var>
                    <var>z</var>
            </atom>
        </_head>
        <_body>
            <and>
                <atom>
                    <_opr><rel>livesIn</rel></_opr>
                        <var>x</var>
                        <var>y</var>
                </atom>
                <atom>
                    <_opr><rel>partOf</rel></_opr>
                        <var>y</var>
                        <var>z</var>
                </atom>
        </_body>
    </imp>
</rulebase>
```

It is planned to extend the RuleML syntax to Horn logic and to webize the language by allowing to reason over objects identified by URIs. This will also include the ability to introduce an ontology's type system into the rules by linking the rule relations, constants, and variables to the respective properties and classes found in the ontology. Grosof and Poon already suggest a possible syntax [70]. Grosof and Horrocks as well as Volz et. al. also present initial work on the DAML Rules project which aims at combining description logics and logic programming [68]. Outside the Semantic Web context, this approach as originally adopted by the CARIN system [100]. Our developments, which we will present in sections 8.1 and 8.2, regarding basing RuleML on RDF Schemata go in the same direction. Furthermore a RuleML working group is currently dedicated to developing a reaction rule extension. An interesting issue regarding the manageability of a rule set are courteous logic programs where a more

specific rule added at a later point in time can override more general rules [69]. Benjamin Grosof also did some interesting work in this area and provides a working system called SweetJess [67]. The work of Antoniou provides the basis of integrating prioritized rules with description logics [5]. These directions of research are likely to influence or to be included in future RuleML versions. RuleML is supported by a growing number of software tools. The TRIPLE and MANDRAX engines [84], for instance, provide inferencing support for RuleML rule bases.

5.6.2 N3 Rules

Besides giving an alternative notation for RDF graphs, N3 also includes a simple rule syntax. The predicates for implication, existence, and forall quantification are located in the namespace http://www.w3.org/2000/10/swap/log#. Similar to XML namespaces, a default namespace with the empty prefix can also be defined. As a basis for rules, N3 introduces syntactical constructs like {`:s :p :o`} which are called formulae. Note that this formula does not state that the triple (s, p, o) is true. Formulae are rather connected via predicates from the log namespace. Consider the following example:

```
@prefix log: <http://www.w3.org/2000/10/swap/log#> .
@prefix : <http://example.org/> .

this log:forAll :p, :q.

{ :p ont:inverse :q. } log:implies

    { this log:forAll :x, :y.
        { :x :p :y. } log:implies { :y :q :x. }
    } .
```

This can be read as: the following is true for all combinations of p and q: If p is the inverse of q, then for all x and y, (x, p, y) implies (y, q, x). N3 does not make any statements about how these rules will be executed. Tim Berners-Lee provides a reference implementation called CWM, which is described in the survey section 6.4.2 on inference engines.

5.6.3 Java Rules JSR

Java Specification Requests (JSRs) are the initial phase in a process of establishing a standardized API for the three Java distributions, the micro, standard, and enterprise editions. The, in the meantime, well-established Servlet specification, for example, also went through a JSR process. The Java Rules JSR 94[15] aims at easing the integration of the various rule engines into existing Java

[15] http://www.jcp.org/jsr/detail/094.jsp

code. Currently this is a major obstacle for the development of rule based enterprise systems for workflow management. The Platform for Privacy Preferences (P3P) project in browsers would also be a prime candidate for being developed in a rule-based fashion. The goal is to provide a generic adapter, which will allow various rule engines to be plugged into an application. Note that the Java Rules initiative is solely concerned with providing ways to interact with the rule engine's memory as well as administrative functions like loading and unloading a ruleset or starting and stopping the engine.

Java Rules are not concerned with rule syntax. Therefore, RuleML might complement Java Rules since both technologies together would allow an application not only to use different inference engines but also load different rulesets for execution.

In our opinion, Java Rules is very promising work in progress. Currently a prototype exists that connects JESS to a Java application via Java Rules.

5.7 Encryption and Digital Signatures

As we already stated in section 5.1, the proof and trust layers, as envisioned in [12] or [138], are long term research goals. Nevertheless, we want to briefly outline the status of the enabling technologies such as encryption and digital signatures. The current Internet security infrastructure is mostly based on HTTP basic authentication and the secure socket layer (SSL). With basic authentication, the client is challenged for a password upon requesting a restricted resource. The username and passwords are simply base64 encoded and can be read by anyone eavesdropping on the network traffic. Therefore, basic authentication only makes sense in conjunction with SSL. SSL uses the public key infrastructure to exchange a triple DES key which is then used for the further communication [130]. The use of the public key mechanism is restricted to the key exchange, since Triple DES can be processed faster. In conjunction with server and potentially even client certificates, these technologies enable the most basic security requirements of authentication, confidentiality, and integrity. Non-repudiation, i.e. the guarantee that a message really comes from the entity one thinks it is coming from, can only be supported by adding a digital signature.

In addition to this missing feature, especially for web services, it would be desirable to be able to integrate intermediary brokers in a message chain. With SSL, it is not possible to protect the message content from them, since a message must be decrypted and encrypted again in order to be forwarded via a different SSL session. There are several proposed standards looming in the XML and web service communities, namely XML Encryption, XML Signature, and SOAP Security extensions, which deal with mechanism for achieving transport independent authentication, encryption, and signatures. A nice overview of the current situation and tools is given in [65].

In any case, it is very likely that the W3C will publish a standard soon. Within the long-term timeframe for the Web of trust, the encryption and sig-

nature standardization issues will therefore definitely not be an obstacle.

5.8 Web Services

Even though Web Services are not directly related to the Semantic Web, their recent success made it the future de facto standard for Web-based remote procedure calls. This development prompted the Semantic Web community to investigate ways in which the two technologies can be combined in useful ways. Sections 5.8.4 and 5.8.5 look at the two most prominent developments in this area, namely DAML-S and WSMF. Before that we will outline the history of Web Services and briefly discuss some of the important technical issues.

5.8.1 The History of Web Services

After the Web's initial phase of being a medium for convenient reading and publishing static information, the popularity of Web applications has grown enormously. Today, there hardly is a service or a good that is not available online. Nevertheless, almost all of these services are geared towards human interaction. The electronic data interchange (EDI) community had quite some success in standardizing message formats for application integration; it is however impossible to develop a lightweight standard that serves a variety of application domains. Therefore, EDI solutions are typically very specific to a certain industry.

In 1998, the XML specification laid the foundation for more large-scale solutions by defining a generic syntax for semi-structured data. Even though XML is a very low level specification, the wide support is all kinds of software solutions allows for much simpler and faster development of B2B or EAI software. Currently, existing EDI standards are being mapped to the XML world. EbXML is one of the more prominent examples of this trend. Microsoft's BizTalk server also incorporates this approach. It acts as a message exchange hub that is able to map different message formats into others. Internally, all incoming files are converted to XML in order to be able to transform them using XSLT.

Apart from XML as being a data representation format, Web Services are XML's other major application area. Remote procedure calls basing on XML data encoding and standard Internet transport protocols are believed to be the silver bullet for lifting the Web to the next level: The programmatic exchange of information between technically and organizationally completely heterogeneous systems. This is a point where traditional middleware like DCOM and CORBA [118] failed already at a technical level. Despite some promising approaches on data integration like the one presented in [17], different vendors' solutions proved to be incompatible. Furthermore, the lack of a universally available infrastructure like naming services, security, and protocols, made a wide-scale adoption impossible. Therefore, these middleware solutions are typically found in implementations of a distributed system, which is deployed under the supervision of a single team of developers. This situation was disadvanta-

geous not only for customers, but also for core technology providers, since not too much money can be made in the middleware market. This is one of the main reasons for the previously unseen cooperation on XML middleware interoperability. The goal is to compete on the enterprise application level instead.

5.8.2 SOAP, WSDL, and UDDI

The W3C specified a stack of markup languages that cover all aspects necessary for a distributed, dynamic, and service-oriented architecture. XML in conjunction with XML Schema datatypes provides the basic marshalling and unmarshalling mechanism. On top of this, the Simple Object Access Protocol (SOAP) defines how messages can be packaged and sent to their destination. SOAP also defines the concept of message headers containing management information that is not directly related to the actual message contents in the body. Headers can include transaction or session information [128] for example. Like every remote procedure call framework, Web Services also need an interface description language. The Web Service Description Language[16] (WSDL) defines message types and error codes, but it also provides a binding mechanism that identifies the exact URL a message has to be sent to in order to be processed by the respective Web Service implementation. Finally the Universal Description, Discovery and Integration (UDDI) standard establishes a standard procedure to publish and query Web Services in a public repository[17]. In order to enable flexible searches, repositories are able to store rich service metadata. This includes basic contact information on the company hosting the service as well as a taxonomic classification of both the service and the company according to well-known classification systems such as NAICS, UNSPSC, or the ISO 3166 geographic taxonomies. From an ontological point of view, these well-known taxonomies again provide a standard vocabulary, which is important for publishing, searching, and discovering services. For instance a query could be formulated that searches for all services offered by companies in the NAICS education sector, which are located in the ISO 3166 region of North Dakota.

The Web Service standards stack of SOAP, WSDL, and UDDI allows for business partners' services being located and invoked at runtime. A supply chain management system, for instance, can easily register a new trading partner in a local UDDI repository. From there, the overall system can obtain the service location and query prices or the availability of items in stock. The seamless integration of these standards into development tools, most notably Visual Studio .NET, allows for rapid product development.

Despite the recent advances on technical interoperability, it is still a dream to invoke remote services in a completely automated fashion. In the supply chain management example above, there needs to be some sort of standardization body that specifies the interfaces for querying prices and availability. These standards will then be represented by UDDI tModels. A new supplier's

[16] http://www.w3.org/TR/wsdl
[17] http://www.uddi.org

services can only be invoked, if they follow the overall system's tModel specification. Commonly, this specification is given as a WSDL interface description. In contrast to searching for a certain service type at runtime, the other common pattern is to use the UDDI registry as a design time service repository. A developer can manually search for services, read the documentation, generate proxies, and write code to invoke the services. Neither of the two approaches allows for a completely automatic service invocation without a priori knowledge about a certain tModel or without a human writing custom code for unknown tModels.

Note that there is broad support in the industry for Web Services. As a matter of fact, similar technologies that have been developed before are being merged with the Web Service effort. An example are HP's e-Speak business to business technology tools that also provide an RPC infrastructure with a UDDI-like company-wide service repository that is searchable via taxonomies and keywords assigned to services. The current HP Web Services suite, formerly known as e-Speak, is now entirely devoted to Web Services[18].

5.8.3 WSFL and XLANG

Obviously the current language stack is mainly concerned with fairly basic features required for remote procedure invocation. Issues like behavioral aspects, quality of service, contractual issues, or business processes are completely uncovered. Consequently, yet another set of languages is being developed to address these issues. Many representatives can be named here, however we focus on two languages. XLANG is a Microsoft specification, and an implementation is available with Microsoft's BizTalk Server[19]. The second representative is the Web Service Flow Language (WSFL), which is an IBM specification. In turn, a WSFL implementation is available through the Web Services Process Management Toolkit[20].

A goal of both languages is to provide a means of specifying how a to use the functionality provided by a collection of Web Services in order to create a composite service. Such a composite service description will then contain the execution sequence or choreography of individual Web Services. This includes descriptions of tasks, loops, branching tasks, merging tasks, and so on. Special emphasis is devoted to the handling of transactions and exceptions.

The Web Service Endpoint Language (WSEL) is planned to complement WSDL and WSFL in that it allows to capture properties of the provider. For instance, it is possible to specify that potential callers must support a certain encryption mechanism or that the service operator guarantees a maximum response time of 20 seconds.

[18] hp web services platform: a comparison with hp e-speak: http://www.hpmiddleware.com/downloads/pdf/espeak_webservices.pdf

[19] http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm

[20] http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

5.8.4 DAML-S

Business process definitions are a step forward, but ultimately it is necessary to capture what a service does on a conceptual level. UDDI addresses this issue to some extent by allowing to classify services and companies according to standard industry, service, and geographic taxonomies such as UNSPEC. However, it would be preferable to use a widely accepted ontology for such tasks. Only this will allow a software agent to inspect the description and invoke the service on the fly, without any prior knowledge.

The DAML initiative described in section 5.4.1 created a subgroup working on the DAML-Service (DAML-S) specification. DAML-S comprises an upper-level ontology vocabulary for services [30]. It contains three major description categories.

- The service grounding contains technical information required for accessing the service. DAML-S is not restricted to be used with Web Services only, but can be used in conjunction with CORBA or KQML.
- The service model contains information on how the service works. This includes pre- and post-conditions as well as a description of input and output parameters. These descriptions link a parameter not only to its datatype but also to an ontological term.
- The service profile describes what the service does. This information is similar to XLANG and WSFL like process models.

McIlraith et. al. developed a small prototype of a travel portal that invokes semantically tagged Web Services [112]. Other than that, DAML-S is work in progress and no other prototypical implementations exist to our knowledge.

Many efforts aim at a further integration of Semantic Web and Web Service technologies. A nice survey and feature comparison of DAML-S and related technologies like e-Speak and UDDI is available on the DAML website[21]. Paolucci et. al. provide some initial ideas on semantically describing Web Service capabilities ontologies [120]. These efforts were implemented in a prototype which combines DAML-S and UDDI via a translation component [119].

5.8.5 WSMF

Within the Web Service Modeling Framework (WSMF), four major aspects are identified [49]. Like DAML-S, ontologies are used as an integral part to enable interoperability and a certain level of understanding among participating agents. Secondly, WSMF contains goal repositories that define the problems that should be solved by web services. A goal consists of pre- and post-conditions. For instance, asking for the nearest restaurant requires, as a pre-condition, the ability to specify the current location. In terms of the Web Service descriptions, the third point is that WSMF makes a clear distinction between the internal Web

[21] http://www.daml.org/services/daml-s/2001/10/survey-f-release.pdf

Service structure and its externally visible interface. Fensel and Bussler argue that a service, which is in a complex way composed of various other services, can have a simple external interface. Many Web Service description languages distinguish between elementary and complex Web Services, which implicitly confuses internal process structure and external signature. Finally, a mediator architecture is proposed to overcome interoperability issues which are, for instance, caused by different invocation patterns on the client and server sides. Similar to DAML-S, WSMF is also work in progress.

Chapter 6

Survey of Available Resources

After having explained the rationale for basing our work on Semantic Web standards and introducing the new technology in the previous two chapters, we evaluate the current state of the Semantic Web. This has two aspects to it. First of all, we have a look at the tools that have been and that are being developed by researchers around the world. Even though the Semantic Web has not yet reached the regular development departments of major IT companies, the amount of tools that has sprung up recently is quite amazing. Secondly, we also want to check how much data is marked up using languages like RDF and RDF Schema. We believe that the tools and the technology as such can only mature if there is a large enough body of data to operate against. Our survey with respect to sources of data is twofold. We look at big data- and knowledgebases that are frequently used and referred to from projects in the Semantic Web field. It is important to note here that in principle it does not matter whether a certain ontology or data collection is available in a Semantic Web mark-up language. The important aspects are that the collection is useful and publicly available. Converting the representation from one format to another usually is not the big problem. Apart from large dataset available from one organization, we also did crawling experiments to see if the average Internet user has already begun to use these new technologies. The results presented in section 6.2 show that this is not yet the case but that there are also promising developments. Finally, section 6.4 surveys the software tools available.

We want to point out that in this area, especially the tools are developing very fast. We can only provide a snapshot which will be outdated fairly soon. However, the information is useful to see where the strengths and weaknesses lie and in which direction future research needs to be directed.

6.1 Data and Ontologies

The title of this section "Data and Ontologies" again manifests the problem of when something should be called an ontology. All of the projects cited here qualify as ontologies in that they are well known and define a certain terminology that is used in a large user community. Wordnet, for instance, can be considered an ontology, even though the RDF Schema ontological part of it is very simple and small compared to the data portion carrying the important information. In turn, the UNSPEC product classification scheme is modeled as a huge taxonomy of product and service classes and contains no data at all.

We do not want to discuss the pros and cons of both representations here. This issue is addressed in section 9.1.1. Instead, we simply explain the most prominent projects and briefly outline their potential use within Semantic Web applications. Magkanaraki et. al. provide a more comprehensive list of related projects [104].

6.1.1 Wordnet

WordNet is a lexical reference system developed at Princeton[1]. Nouns, verbs, adjectives and adverbs of the English language are organized into synonym sets. Each of these sets corresponds to an underlying lexical concept. The synonym sets themselves are cross-linked via both lexical relations between word forms and semantic relations between meanings. The relations used are antonymity, hyponymity, and meronymity [113].

WordNet is a valuable resource due to its strong linguistic features and its broad coverage of the English language. As we pointed out in the related work chapter, it is used frequently in systems such as the Web Knowledgebase (WebKB) [107] for performing shallow lexical analysis. Wordnet terms can furthermore be used to provide a well-defined vocabulary for attribute ranges. For instance, a photo annotation tool, described in [127], uses WordNet definitions for possible color values. The Semantic Web community also realized WordNet's potential and an RDF version of it has been published on the SemanticWeb.org portal[2]. The WordNet relations are captured and described in a corresponding RDF Schema definition.

The site xmlns.com is devoted to providing fixed reference points acting as crystallization points for a Web of metadata forming a global RDF graph. Consequently, it provides URIs that can be used to refer to WordNet concepts in an RDF way[3]. Note that these names show up quite frequently in the following survey of today's RDF resources. Section 6.2.1 provides more information on this topic.

[1] http://www.cogsci.princeton.edu/∼wn/
[2] http://www.semanticweb.org/library/
[3] http://xmlns.com/2001/08/wordnet/

6.1.2 Open Directory

The Open Directory is the largest human-edited directory on the Web[4]. Unlike other Web portal sites like Yahoo! or Web.de, the Open Directory adopts an open source strategy. Currently over 50,000 volunteers manage the catalog, organize the categories and review pages before including them. The popularity of this project got a dramatic boost, when Google incorporated the Open Directory project into its site.

Due to its open source character, the base data about the category structure and the page classifications is freely available for download in RDF format. Currently this comprises about 3.8 million sites and over 460,000 categories making it the largest dataset available in RDF format. Obviously this is an extremely valuable resource that allows a user, for instance, to classify his or her own pages according to this popular scheme.

6.1.3 OpenCyc

Like the Open Directory project, OpenCyc is also an open source project and therefore a freely available version of the Cyc knowledge base[5]. The Cyc project aims at building a foundation of basic common sense knowledge via a huge collection of concepts and assertions. The effort is characterized by its top down ontology development approach, trying to establish a set of top-level concepts to be used by other, more fine-grained ontologies. This is often referred to as a standard upper ontology.

The usefulness of standard upper ontologies is discussed controversially. On the one hand combining smaller specific ontologies with a general upper ontology seems like a natural thing to do. On the other hand, the concepts are necessarily extremely abstract and it takes quite some effort to get into the thought process that went into their development, let alone trying to figure out how to use them for one's own application.

The OpenCyc distribution comes with a large ontology of 6,000 concepts and 60,000 assertions as well as browsing, editing, and inference tools. An API to connect OpenCyc with other applications is also provided.

6.1.4 Gene Ontology

The objective of the Gene Ontology project[6] is to provide uniform vocabularies for the description of gene products. It grew out of the desire to integrate the many biological databases available from various research groups via the Web today. Even though Gene Ontology does not solve the data integration problem, establishing a shared vocabulary is the first step towards this goal.

Currently, the Gene Ontology consists of three logical components, which model molecular functions, biological processes, and cellular components. These

[4] http://dmoz.org
[5] http://www.opencyc.org
[6] http://www.geneontology.org

can be viewed and edited via the DAG Edit software. This editor can be equipped with an export module allowing the ontology to be represented in RDF.

6.1.5 MusicBrainz

With the recent developments in Peer to Peer networking, especially the sharing of MP3 files, the MusicBrainz project got some attention. It offers a Web-enabled RDF version of the popular CD database (CDDB) [137]. CDDB is a huge collection of information on artists, titles, music genres, and CD's available on the market. It is used as an online discography as well as a backend system which MP3 players like WinAmp query in order to obtain metadata of the track or CD currently being played. Like some of the other projects mentioned, MusicBrainz adopts the open source approach and the information on close to one million tracks is freely available. Another key feature is that contributions are also open. Users can submit new information to MusicBrainz or correct errors in the data set.

6.1.6 MIT Process Handbook

The MIT Process Handbook project collects information on how different organizations perform similar processes and represents them in an online database. The goal is to provide a tool for redesigning existing organizational processes, inventing new IT supported organizational process structures, and, as a long term goal, generating business process software support automatically [106]. The first goal as been at least partly achieved since the handbook has been used by consulting firms. As part of the SweetDeal effort, Grosof et. al. describe how the MIT Process Handbook was converted to DAML+OIL [70]. Currently only a smaller fraction of the around 10,000 classes have been converted into Semantic Web format. Nevertheless, this is an interesting basis for marrying knowledge-based computing with the area of business processes and workflow management.

6.2 RDF Survey

Since RDF and the ability to state facts is the foundation of the Semantic Web, a survey of how much RDF data can be found is of interest. RDF is definitely used a lot by the Semantic Web research community; therefore, the survey is more an indication to what extent the general public is starting to adopt the technologies developed. It is also of interest to evaluate typically used predicate namespaces in order to draw conclusions about the application areas. The following text illustrates how the search was conducted and which tools were used. The results are then presented in section 6.2.2 before an evaluation and a summary are provided.

6.2.1 Collection of the Survey Data

We first did some initial experiments using the RDF crawler developed at the University of Karlsruhe[7]. Given a starting URL, RDF Crawler recursively traverses hyperlinks up to a specified search depth. RDF data found is stored in a file on the local system. The experiments quickly revealed that it is not easy to find RDF data on the Web. If the starting points for the search are not selected carefully, a pure crawling approach might require an extensive amount of URLs to be processed before any RDF data is found. Therefore we decided to pursue four different strategies, which are outlined in the following paragraphs.

The first experiment was performed in December of 2001. With the software and the search process in place, we reran the same experiment in August of 2002. The main intention for this second run was to see if a significant increase could be observed after the Semantic Web initiative got quite some public exposure at that time. We plan to repeat the experiments in the future as well.

Crawling According to a study by Lawrence and Giles in 1998 [98], even major search engines that continuously crawl the Web only achieve a coverage of at most 17% of the static Internet pages. Due to the massive growth of the Internet, this number is likely to have decreased even more [9]. With the limited bandwidth and computing resources available to our study, it was only possible to cover small islands of URLs. Nevertheless, we applied this approach, but chose popular sites within the RDF community as starting points. Table 6.1 shows these URLs that were used in both search runs. A total of 12507 pages within two hops of these URLs were processed in the first run. Two major RDF collections, namely the Open Directory Project structure and content dumps and the RDF version of the Wordnet lexical database project, were left out due to their large size. A site related to the RPM software packaging tool[8] also contains a large number of RDF files describing software distributions. These were only scanned in part. Since the choice of starting pages is very restrictive and quite arbitrary, we decided to include the Google directory page on RDF and its fifteen subordinate categories as well for the second run[9]. Similar to the www.semanticweb.org pages, these contain a very complete set of links to all sorts of RDF related sites. 31764 pages of this category were crawled during the second run.

Open Directory In order to make sure that the breadth of the Web is somewhat captured, we searched URLs from the Open Directory project. During the first run, 527408 URLs were extracted. Due to the massive growth of the Open Directory project, within eight months, this number increased to 2912434.

[7] http://ontobroker.semanticweb.org/rdfcrawl/

[8] http://rpmfind.net/linux/RDF/

[9] The Google RDF directory can be found at http://directory.google.com under Reference > Libraries > Library and Information Science > Technical Services > Cataloguing > Metadata > Resource Description Framework. Note that the Google directory bases on the Open Directory.

URL
http://www.w3.org/RDF/
http://wilbur-rdf.sourceforge.net/
http://www.daml.org/
http://www.lassila.org/
http://www-db.stanford.edu/~melnik/
http://www-db.stanford.edu/~melnik/rdf/api.html
http://www-db.stanford.edu/~stefan/
http://www-db.stanford.edu/
http://www.semanticweb.org/
http://protege.semanticweb.org/

Table 6.1: Popular sites within the RDF community were chosen as starting points for crawling

This comprises links from all categories, except for the adult pages. The URLs obtained from this source are typically entry- or homepages. Due to the large number of URLs, no more crawling was done from these sites. Obviously this approach will not find standalone RDF data, residing is a separate file. However, we expected to find information about the website encoded in the HTML page itself, as demonstrated by the following example:

```
<head>
  ...
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description rdf:about=""
    dc:title="Ora Lassila"
    dc:description="Ora Lassila's professional home page"
  </rdf:RDF>
</head>
```

Targeted search Since the initial experiments indicated that RDF data is hard to find, a more targeted search was conducted. Google allows searching for pages with a certain string in the URL. Obviously a URL containing RDF is more likely to contain some RDF data than a URL that does not. A small parser was developed that extracts the URLs out of a Google HTML result page. During the second run we used the newly available Web Service interface that allows Google to be queried programmatically from a Java or .NET client[10]. Leveraging the extensive Google database, a total of 1256 URLs were obtained during the first run. Table 6.2 summarizes the number of URLs in each of the three categories. There is a small overlap between the categories. Three URLs where RDF data was found appear in both the RDF community and

[10]See http://www.google.com/apis/ for details

the Open Directory categories, 63 URLs appear in the RDF community as well as the Google targeted search categories. For the second experiment, fewer pages containing RDF in the URL could be retrieved from Google and the number decreased to 1079. Note that this is by no means an indication that less information on RDF could be found. The most likely scenario is an internal change in the Google system. The search result site actually claims to have found 2,410,000 pages, however only the number specified could be obtained, both via the browser and the Web Service interface.

URLs found in the fact triples Since RDF subjects, predicates, and most objects can be URLs themselves, we assumed to be able to find RDF data at those URLs. Facts gained from the other categories were extracted first. When examining the collected facts, we only considered URLs that have not appeared in any other category. We chose to implement this restriction due to the expected large overlap with the other categories. After 124374 facts had been found via the other approaches, a first search process on the facts was started, yielding 365 new URLs. Note that URL anchors, i.e. the "#" character in the URL must be ignored, since anchors only identify a certain position within the same document. Therefore it is not necessary to scan such a URL again and consequently only the part left of this sign was considered. The facts of those URLs were loaded again and the process was repeated in the hope that one can follow the edges of the RDF graph to find new data. The 1923 new facts from the 365 new URLs yielded only 23 new websites and the process was stopped at this point.

This situation changed in the second run. 139288 facts were found at URLs from the other categories. The subjects, predicates, and resource objects form those facts pointed to 6037 previously unseen URLs. We loaded 54227 new facts from those URLs. This number is promisingly high, however, it turned out that almost all of the facts came from large data repositories that organized their data not within one large file accessible at a single URL, but rather made that data available via several URLs. One example is http://xmlns.com, where an RDF representation of the Wordnet database is hosted. The URL http://xmlns.com/wordnet/1.6/Survivor, for instance, contains several statements about other Wordnet resources located at similar URLs. Nevertheless, we were able to extract 697 new URLs from those new facts. At this point, hardly any URLs could be identified from facts from those sites that were not a simple derivation of previously seen URLs and the process was stopped.

Architecture of the RDF database In order to be able to perform further analysis of the data, we decided to load the facts into a relational database system. Figure 6.1 shows the table layout. The `facts` table stores subject, predicate, and object triples, along with the id of the URL they were found in. The primary key selection makes sure that data cannot be inserted twice from the same source in case the upload program needs to be run repeatedly. The `URLs` table has a further uniqueness constraint on the `URL` attribute to avoid

	Number of URLs scanned	
Category	**Dec 2001**	**Aug 2002**
RDF Community	12507	31764
URLs from Open Directory	527408	2912434
RDF appears in the URL	1256	1079
URLs from facts	365	6733

Table 6.2: URLs per category

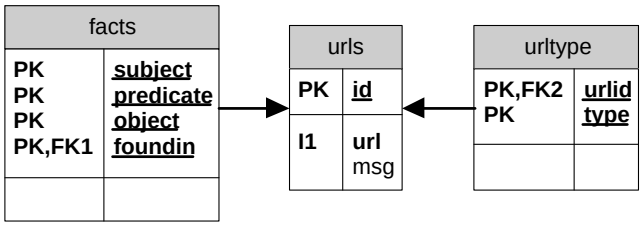

Figure 6.1: Design of the RDF database

duplicate URLs in the data set. Finally, the `URLtype` table indicates which of the four categories mentioned above the URL belongs to. The `msg` field in the `URLs` table records any error such as network errors, XML parsing errors, etc., that might occur while the data is accessed.

Figure 6.2 illustrates the overall software design. Any URL to be scanned is first inserted into the `URLs` table in the database. A different approach was used for each category: Data from the Open Directory dump was extracted with an XSLT style sheet. The program `GetGoogleURLs` obtains URLs containing the string RDF by running a query against Google. The `Crawler` program uses multiple threads to traverse the hyperlink structure from several starting points like Semantic Web portal sites. Finally, the last category's URLs are generated via the `ScanFacts` component, which bases mainly on an SQL statement that selects fact URLs that do not yet show up in the `URL` table.

After the URLs table is filled, the `RDFLoad` program can be started to scan the URLs for RDF data. It uses Sergey Melnik's RDF API[11] to upload the facts to the database. Any Java exception that is caught is written into the message field in order to trace, for instance how many pages contain syntactically incorrect RDF, or how many pages could not be reached because of a network outage. Since RDF API is widely used, a URL is considered to contain correct RDF if the RDF API in the version of Jan 19th 2001 parses it without error message and if the resulting RDF triple set is not empty. It is considered to contain incorrect RDF, if an `org.xml.sax.SAXParseException` is thrown, and it is considered to not contain any RDF if a `java.io.EOFException: no more`

[11] http://www-db.stanford.edu/∼melnik/rdf/api.html

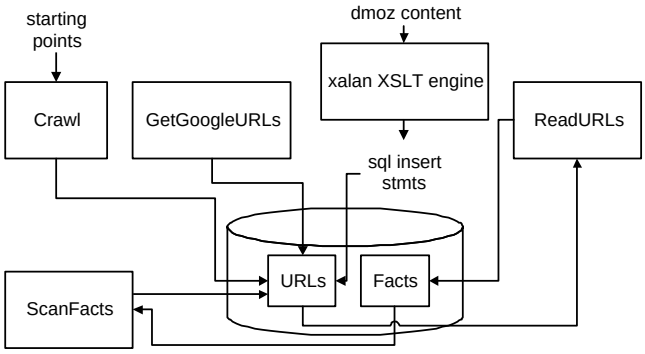

Figure 6.2: Overall software design

`input` exception occurs or if the resulting RDF triple set is empty. Any RDF data found is also written to a file `<num>.rdf` for further examination, with num being the id of the URL scanned. Finally, `ScanFacts` inserts URLs found in the `facts` table into the `URLs` table. Note that `ScanFacts` can only be run after `ReadURLs` inserted some facts. Furthermore, `ReadURLs` must be run again to load facts found in the newly inserted URLs.

The major advantage of this database-centric architecture is that the search process can be stopped and resumed without any problem. The database provides the necessary persistence and constraints to make sure that data cannot be inserted twice. Since this survey aims at evaluating a snapshot of the current use and acceptance level of RDF, the database only accumulates data and no mechanism for deleting or updating the information is implemented.

The application as well as the data sets can be downloaded at http://www.i-u.de/schools/eberhart/rdf/.

6.2.2 Search Results

This section outlines the results of the search conducted over 541536 web sites in the first, and 2952010 web sites in the second search.

How many pages contain RDF data? Figures 6.3 and 6.4 show how many pages contained RDF data by outlining the percentages of the following cases: a general error such as file not found occurred, page available but no RDF data found, syntactically incorrect, and correct RDF found. As expected, we see that there are strong variations between the categories. During the first experiment, RDF data was found in only sixteen out of over half a million pages from the Open Directory. This number increased to 180 out of 2.9 million pages in the second run. The density around Semantic Web portals is higher but still disappointing. About one percent of the URLs were found to contain RDF in

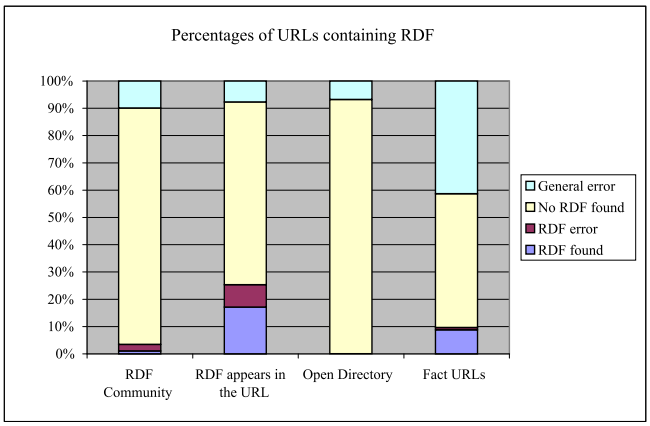

Figure 6.3: RDF data found per category during the first search Dec. 2001

both runs[12]. Finally, the highest success rate was found in pages with "rdf" in the URL, especially pages ending with ".rdf". These URLs contain RDF data with a probability of 17% and 10% in the first and second experiments respectively. Similar percentages occurred in the last URL category, where we followed RDF arcs that were present in the facts found in other categories. We found RDF in 9% and 13% of the pages during the first and second runs. Overall, with the categories combined, this translates to 1018 out of 541536 URLs containing RDF, 613 of them with correct and 405 with incorrect RDF for the first run. In the second run, out of 2952010 pages, 1479 contained valid and 2940 contained invalid RDF. Please note that the overall numbers are largely dominated by the Open Directory category, where the bulk of pages were scanned.

Error causes Figures 6.5 and 6.6 provide more detailed information about the error causes, i.e. the "General error" and "RDF error" sections from figures 6.3 and 6.4. Network errors and URLs that no longer point to any page were expected to be the most frequent sources of problems. In the first experiment, system errors are caused by the `ReadURLs` component and play a small role for the URLs gained from the targeted search. Five of these failures were recorded, four null pointer exceptions with unknown causes and one out of memory error, caused by a large binary file. During the second run we increased the number of threads used to search especially the large number of Open Directory pages. This resulted in a substantially higher number of 1147 out of memory errors in this category. Considering the total number of 2.9 million URLs scanned, this is definitely not a big concern for the quality of the results.

[12]Originally we also counted documents where the RDF API yielded an empty result, i.e. a set of zero RDF statements. The exclusion of these pages explains the higher number stated in [40]. Also refer to the updated version [41]

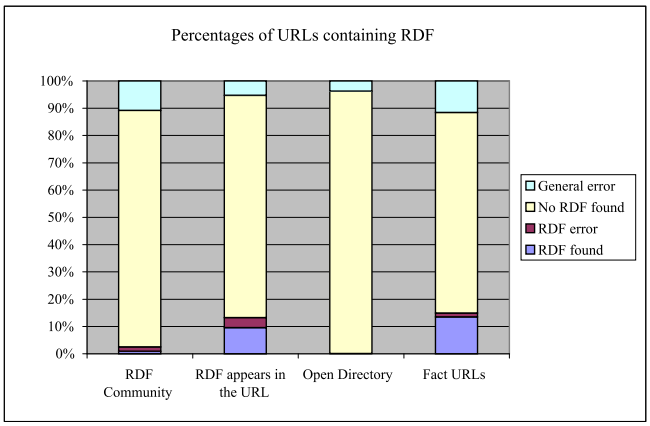

Figure 6.4: RDF data found per category during the second search Aug. 2002

An interesting issue are the 405 and 2940 RDF parsing errors that might reveal potential problems caused by the use of older RDF versions or frequently occurring mistakes made by RDF authors. In both experiments, over half of these errors are caused by unresolved entity declarations such as the non-breaking space entity ` ` defined in HTML. The remaining error causes are partly XML-related such as missing attribute quotes and partly RDF-related such as nested descriptions. We found that 14 and 24 URLs respectively omitted the RDF namespace and simply placed `<RDF>` tags in the document. This manifests itself in a "unresolved namespace prefix" error.

Size of the RDF data sets During the first search, a total of 125072 facts were extracted, 104580 came from the targeted search category, 19696 from the RDF community category, 1923 from facts URLs, and only 98 from Open Directory web sites. Figures 6.7 and 6.8 illustrate how much data was found at the different URLs. When analyzing the second run with respect to the category's contribution to the total number of 254783 facts, it can be seen that with 115495 facts, the last category contributed more than the RDF community pages with 107308 facts. 29168 facts come from the targeted search and 2812 from Open Directory pages. The first run yielded only three large files with more than 10000 facts, namely a list of airports from http://www.megginson.com, an excerpt from the CIA world fact book at http://www.ontoknowledge.org, and a category description file at http://w.moreover.com. The second search tapped into five large repositories, namely the OpenCyc project at http://opencyc.sourceforge.net, part of the WordNet database at http://www.semanticweb.org/library/wordnet/wordnet_hyponyms-20010201.rdf, two military ontologies at http://orlando.drc.com/, and again http://w.moreover.com. Overall, two changes can be observed between the experiments. First and foremost, the last category happened to tap into

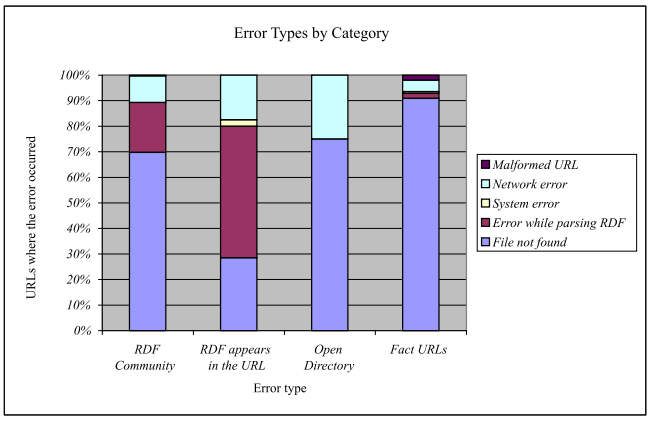

Figure 6.5: Error types during the first search Dec. 2001

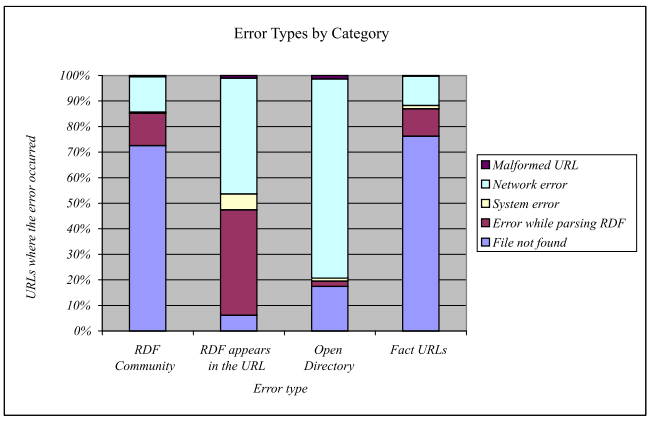

Figure 6.6: Error types during the second search Aug. 2002

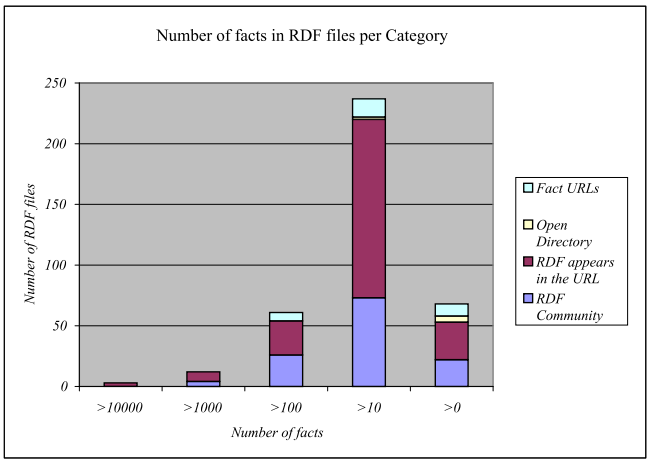

Figure 6.7: Distribution of the RDF data set sizes during the first search Dec. 2001

two highly connected datasets, the xmlns.com version of Wordnet, which is split into many files and the moreover.com directory. This resulted in the large overall increase. The remaining categories actually show little change except for the large number of URLs containing medium sized datasets in the Open Directory category.

A fairly large number of sites contain data in the rich site summary (RSS) format version 0.9. RSS is a format that originally has been proposed by Netscape as a lightweight syndication format for distributing news headlines on the Web, for example via Netcenter channels. An RSS example is shown in the following block:

```
<rdf:RDF xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
         xmlns="http://my.netscape.com/rdf/simple/0.9/">
<rss>
    <channel>
        <title>BBspot</title>
        <link>http://www.bbspot.com</link>
        <description>Your Spot for Tech Humor</description>
    </channel>
    ...
```

Typical namespaces used After the probability of finding RDF data and the typical data set sizes have been evaluated, we examine the facts further. In order to be able to correctly interpret data, it is crucial that an agent understands or can correctly interpret the predicate used in the triple. One of the

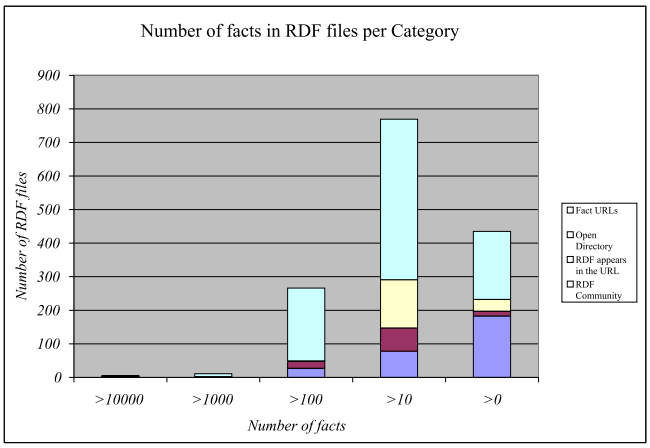

Figure 6.8: Distribution of the RDF data set sizes during the second search Aug. 2002

most prominent and frequently cited examples is the Dublin Core metadata vocabulary. Tables 6.3 and 6.4 show how often a certain namespace prefix occurs in the facts gathered, along with information at how many distinct URLs this namespace prefix was found.

In the data of the first experiment, we can see that some large data sets like the Ontoknowledge case study and David Megginson's airport example rank among the top namespaces but are only used by one web site. The largest number of distinct sites references the W3C and the Dublin Core namespaces, with the RDF type relationship occurring very often. We really should have counted the distinct hosts rather than distinct URLs, since a large data set being split into several files, like in the rpmfind.net example, would create the wrong impression. This was not done since the data model does not support this specific query. A manual check confirmed though, that the URLs really are located on a large number of different hosts. In contrast to our suggestion of using Wordnet or the Open Directory as a base for anchoring statements, no predicates that could be linked to these projects were found.

The second experiment shows a similar picture. Again, we can see some of the large datasets that contribute many facts but occur only within a very limited number of documents. Dublin Core remains the most frequently used non-W3C namespace, however, the Adobe namespace is a prominent newcomer in this list of namespaces found at several different sources. These pages follow the Adobe eXtensible Metadata Platform (XMP) [3]. XMP builds on top of RDF and is designed to embed metadata into application files. The fact that a major IT company embraces RDF is obviously a very encouraging sign for the Semantic Web community.

Like the predicates, commonly referenced objects, often referred to as RDF

Predicate namespace prefix	in # of URLs	in # of facts
http://www.ontoknowledge.org/oil/case-studies	1	23259
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	326	21011
http://www.w3.org/1999/02/22-rdf-syntax-ns#	326	17298
http://www.megginson.com/exp/ns/airports#	2	13589
http://alchemy.openjava.org/ocs/directory#	1	7014
http://www.w3.org/2000/01/rdf-schema#	62	6182
http://purl.org/	123	5198
http://interdataworking.com/vocabulary/	27	4698
http://www.trustix.net/schema/rdf/spi-0.0.1#	2	3012
http://my.netscape.com/rdf/simple/	93	2446
Other http://www.w3.org	331	2212
http://www.daml.org	27	2032
http://www.rpm.org	7	1716
http://metainfo.hauN.org	1	1351
http://home.netscape.com/	1	801
Other	164	13253

Table 6.3: Predicate namespace prefixes used by the RDF data found during the first search Dec. 2001

hubs, are also important for agents to understand RDF facts. Metadata of a web site pointing to an Open Directory category is an example. This would allow any agent aware of the Open Directory to draw conclusions about the content of the site for example. Table 6.5 shows the results of this test. In both experiments we found about 57% of the objects to be literals, mostly numbers and the frequently occurring strings "en", "text/plain". As the large number of RDF type predicates suggests, the objects are mostly RDFS classes. We could not find any non-class object that is referenced frequently from many different sites. Hardly any references to prominent repositories such as Wordnet or Open Directory objects were found.

6.2.3 Comparison of the Two Experiments

Before we will give an evaluation, we want to analyze if any trend can be observed when the runs from December of 2001 and August of 2002 are compared. Overall, the results do not show any drastic changes except for the much larger number of URLs and facts found in the last category, which comprises the sites referenced by the other facts found. This hints at a higher lever of interconnectivity among the RDF facts. However, a closer analysis shows that most of these come from a small set of sources. During the first run, URLs from 152 distinct hosts were added. This is within the same order of magnitude than the 269 hosts obtained during the second experiment.

Predicate namespace prefix	in # of URLs	in # of facts
http://www.cogsci.princeton.edu/	1	78445
http://www.w3.org/2000/01/rdf-schema	693	57132
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	1205	37926
http://orlando.drc.com/	19	27773
Other http://www.w3.org	435	11454
http://alchemy.openjava.org/	2	9793
http://purl.org/	463	9411
http://interdataworking.com/	16	5247
http://www.daml.org/	53	4490
http://ilrt.org/	9	2124
http://opencyc.sourceforge.net/	1	1630
http://ns.adobe.com/	152	1589
http://my.netscape.com/	34	902
http://www.rpm.org/	3	734
http://www.ontoknowledge.org/	2	645
http://dublincore.org/	82	544
http://www.omg.org/	3	523
http://www.semanticweb.org/	41	466
http://annotation.semanticweb.org/	5	375
http://xmlns.com/	48	351
http://example.org/	95	121
http://www.nesstar.org/	6	106
Other	129	3002

Table 6.4: Predicate namespace prefixes used by the RDF data found during the second search Aug. 2002

RDF Object	in number of Facts Dec. 2001	in number of facts Aug. 2002
Other literals	58949	237163
Other resources	44562	175110
http://www.w3.org/1999/ 02/22-rdf-syntax-ns#	7646	7947
Numbers	8115	9667
en	2414	3278
hourly	2361	3265
text/plain	1002	1410

Table 6.5: Overview over RDF Objects

Overall more pages were scanned, largely due to the big increase in the size of the Open Directory, and naturally also more pages containing RDF surfaced, although the percentage of pages containing RDF actually declined. The changes are not too big so we advise against trying to draw any conclusion from this.

Possible sources of errors Apart from the limited search that might not have revealed large amounts of RDF data, another source of error is not recognizing RDF data when a page is scanned. We tested several cases of incorrectly formatted RDF and missing RDF namespace definition. It turned out that the RDF API we used for the search reacts in a very robust manner by indicating the problem with a proper error message. If an XML or XHTML document is scanned by the RDF API, an empty dataset is returned. Some random samples of these cases were examined manually and no malfunction, i.e. RDF data that was omitted by the RDF API, could be detected there.

6.3 Summary

Some of the projects introduced in the beginning of this chapter are very promising. Especially the use of the open content approach seems to ensure high quality data, broad coverage within the target domain, and the involvement of a large user community. Nevertheless, there are not too many examples one can cite and we are nowhere near the situation where base data for any conceivable application is available online, as it is the case for HTML pages.

RDF The results of our RDF survey suggest that RDF has not yet caught on with a large user community. Obviously the search was not very extensive. Therefore it is possible, that some large RDF islands were not found. Much RDF data might also not be publicly available on the Web. In a way we are seeing a situation that is similar to the adoption of Web Services. There are millions of data sources that could easily be made available via both Web Service and RDF interfaces on a technical level. Without a doubt, Web Services have covered more ground in terms of public acceptance, however, except for some highly visible services like the Google Web Service API or Microsoft's Map Point service[13], most of the services that can be found in the UDDI registries today have a clear test prototype character. The most likely explanation for this situation is that the automation of the Web, be it through Web Services or the Semantic Web, brings about a radical change from a business perspective. Advertising largely finances today's Web with its free offerings. This must change when machines and no longer humans access the sites. Several payment methods such as micro payment and bulk subscriptions are being considered, however, it is too early to see clear trends or even standards in this direction. Once this shift towards a more automated Web begins, we will probably also see more data being exposed in RDF format.

[13] http://www.microsoft.com/mappoint/net/

The use of RDF as a simple metadata format for HTML pages does not make much sense at the moment, since HTML meta tags do the job just fine. The strengths of RDF, namely its extensibility and the possibility to refer to widely accepted standard vocabularies and global identifiers, are not being used. The very poor search results among regular web sites taken from the Open Directory clearly support this observation. Furthermore, the nature of facts found indicates that the level of interconnection is quite low, i.e. most objects are literals or belong to RDF schemata. Apart from the Dublin Core and the Adobe XMP namespaces, hardly any other non-W3C vocabulary is used by many different authors. Looking at the bright side, specifically Adobe's support of RDF is a very promising sign.

Despite the poor results, we believe that RDF has a lot of potential. The popularity of the NEC CiteSeer [15] research index for example, is a clear indication that there is a need for metadata and better, more targeted search on the Web. CiteSeer extracts the information which other papers are cited by a certain publication. The number of citations is used as an indication for the quality of a publication. If RDF publications metadata had been used, a system like this would be quite easy to implement. One can only imagine the various RDF applications that will be possible in the future.

We believe that it is crucial for the success of the Semantic Web that the research community starts working on some of these applications in order to get a large user community excited about the ideas and possibilities. Only then will it be possible to resolve what seems to be a chicken and egg problem: data will only be marked up if there is an application, and an application only is successful if it operates on a large data set.

RDF Schema A similar picture is drawn by a survey of RDF Schemata [104]. The authors observe that only a few projects examined provide a large body of classes for the backbone taxonomy. Furthermore, many syntax errors were found in the samples. The paper also supports our observation about the Dublin Core namespace being frequently used.

DAML+OIL and OWL With the core languages from the ontology layer being fairly new and not too many tools in place yet, one cannot expect to find a significant number of ontologies online at the moment. Our observation is that most of the ontologies that are currently available at sites like the DAML ontology library[14] are quite simple and mostly define concept taxonomies, a feature that is already available in RDF Schema.

Despite not wanting to raise unrealistic expectations, we believe that the future adoption of these languages has to be monitored very carefully. Hayes criticizes using Description Logics for representing knowledge on the Semantic Web and suggests that deficiencies of DAML+OIL and OWL are in part responsible for their slow adoption [79]. He argues that Description Logic is too concerned with efficiency and by doing so makes DAML+OIL users having "...to

[14] http://www.daml.org/library

take a course in how to say things in peculiar and unintuitive ways." Hayes suggests developing a content language whose sole function is to express, transmit and store propositions to be used by a large variety of inference engines. No safeguards should be placed on this language. Even if this causes an engine to time out during computation, Hayes argues that on the Web, systems need to deal with these kinds of failures anyway. The advantages of an expressive and easy to use language far outweigh these potential problems.

Outlook In order to predict the evolution of ontologies on the Semantic Web, Kim draws an analogy to paper-based business forms [91]. The form designer has the knowledge of the terms and processes related to the form. It is argued that the invention of the photocopier changed the situation by allowing the users to bootleg forms. This creates uncertainty, because tasks and processes might not have been defined for these bootlegs. In the Semantic Web, the analogous tool to the photocopier is the knowledge / ontology modeling tool. Again, the users, not just a single ontology engineer, can codify shared understanding. Consequently, Kim argues that in the long run, the use of ontologies will become a mainstream technique.

6.4 Software

This sections covers Semantic Web software tools. We will first look into graphical ontology editing tools, examine their usability, and how well they support the engineering cycle and collaborative editing of both data and the ontology itself. The second major part looks at inference engines. This section strongly draws on the query language and rules overview section 5.5 in the first part of the chapter. Inference engines also go hand in hand with the storage layer. Therefore, storage solutions are evaluated next. This section concludes with an overview of miscellaneous tools like basic application programming interfaces and some more specialized ontology tools.

6.4.1 Ontology and Data Editors

Graphical ontology editors are important tools that support the development process. In principle, these tools can be compared to HTML editors that allow the user to design a webpage without having to worry about the exact syntax of HTML. In the same way, ontology editors hide the details of the native ontology representation format and allow the user to graphically create and edit certain aspects of the ontology. Obviously, this is an appealing alternative to using a text editor to write the ontology directly in its representation format. However, sometimes this is necessary anyway if a certain feature of the ontology language is not controllable via the graphical interface.

There are several editing tools available today. A survey within the scope of the OntoWeb project tested eleven systems alone [51].

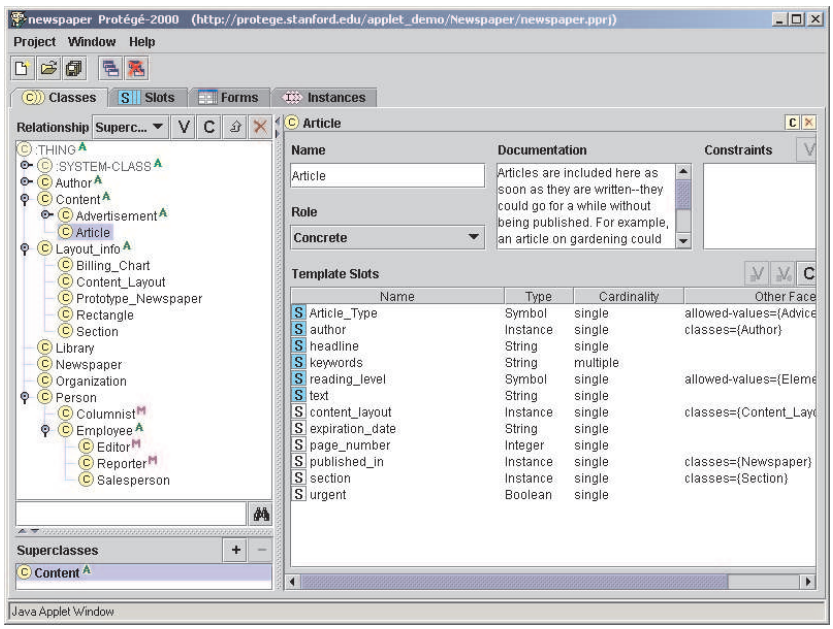

Figure 6.9: The Protégé ontology editor

Protégé The Protégé editor [117] was developed at the Stanford Medical Informatics department. It definitely is one of the most popular editing tools around. Figure 6.9 shows a screenshot of the tool. The left pane shows the backbone taxonomy, which can be conveniently edited by dragging and dropping, adding, and deleting concepts within the hierarchy. Concept hierarchies with multiple inheritance can also be designed using this tool. The large area on the lower right shows the relationships between the concepts. Constraints about a property's domain, range, or cardinality can be entered easily. Besides the ontology, Protégé is also used to enter data. An extensible entry form designer is very useful when non-expert users are supposed to perform the data entry.

The ontology can either be saved to an RDF Schema file or to a relational database. Unfortunately, the database schema used to save the information is not very straightforward and intuitive. Apart from this drawback, the overall architecture is open and allows for including custom plug-ins. This mechanism makes it is possible to develop own data entry widgets, to embed a reasoning engine, or to customize the data export.

Overall Protégé is a very polished and user-friendly tool with nice extensibility features. A weak point of the version we tested was the limited multi-user support, even when a database backend is used for storage. Changes made by one user are not propagated to the others. Protégé still has the notion of saving, even though new concepts are inserted into the database right away. Further-

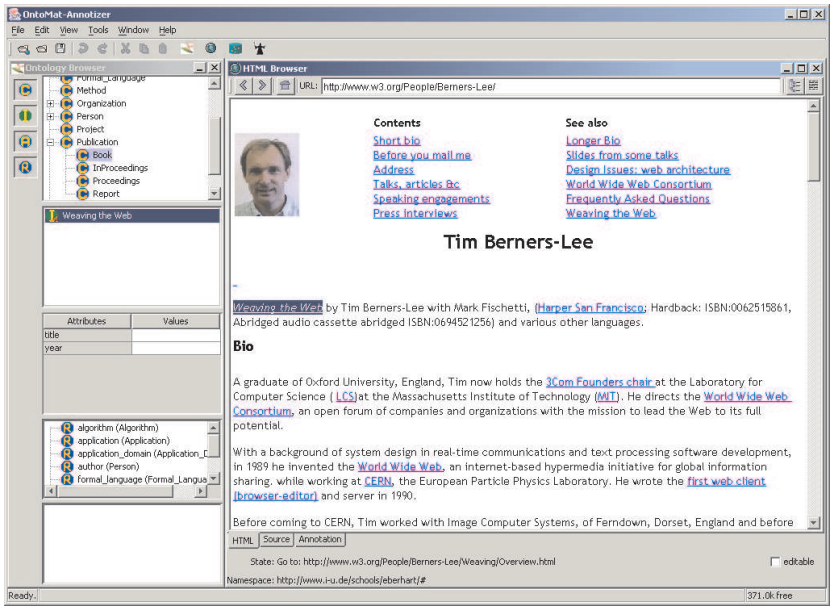

Figure 6.10: Webpage annotation with OntoMAT

more, the database backend table structure is very unintuitive from an RDF point of view. Rather than storing triples, we find a table with eight columns, several of them seemingly holding internal Protégé values. A triple representation including the full concept URIs would make it a lot easier to connect Protégé with other applications via the shared database.

KAON The Karlsruhe Ontology (KAON) and Semantic Web Tool Suite is the second representative to be described [114]. As the attribute "tool suite" indicates, KAON consists of a set of tools. Among them is a very simple editor plug-in for the OntoMAT workbench called Simple Ontology and Metadata Editor Plug-in (SOEP). We chose to present this simple editor since it is able to hook into a central server via the KAON-API. The server centrally stores the actual ontology and is implemented using a relational database. This design choice allows the server to perform concurrency control by locking structures that are being edited by a user. Therefore, this architecture supports the collaborative development of ontologies in the different phases of the ontology engineering cycle.

Important representatives of data editors are the so-called annotation tools. A document, typically a webpage, is loaded and certain text passages can be copied into a metadata set that is associated with the document. Currently this process is still quite cumbersome and ultimately, these annotation features must be included in web pages editors like FrontPage.

Summary As a conclusion, it can be said that there are several very similar ontology editing tools. They differ in certain handling aspects, import and export format support, and the availability of extensibility mechanisms. A weak point of almost all of the tools is the support for the ontology lifecycle phases of collaborative editing, testing, deployment, maintenance, and refinement. More research should also go into better visualization techniques, especially for large ontologies.

6.4.2 Inference Engines

Rules are an integral part of the Semantic Web since they provide an easy mechanism to formalize knowledge that is much richer than a simple class taxonomy. Consequently, rule engines are just as important. Due to the lack of a standard or de facto standard, the rule languages and the corresponding engines used in the Semantic Web context vary a lot. The systems presented in the remainder of this section arguably are the most important ones.

JESS The Java Expert System Shell (JESS) was developed at the Sandia National Laboratories[15]. It's language bases on the CLIPS production rules system [33]. JESS has grown to be one of the most frequently used inference engines. It is, for instance, used for the JSR 94 reference implementation and for the inference plug-in of the popular Protégé editor. The popularity of JESS bases mostly on its seamless integration with the Java language. JESS can be called from Java and in turn Java methods can be declared as JESS functions. There are several other Java engines available today. Another popular system, for instance, is the IBM Common Rules project[16].

XSB The XSB deductive database is often used by Semantic Web related projects[17]. XSB is a particular ordinary-logic-program (OLP) system. It evaluates rules in a backward fashion similar to Prolog. Due to its highly optimized tabled resolution algorithm, XSB is a leading academic OLP system. It supports well founded semantics, which solve Prolog's problem of entering infinite loops, for example in the case of Russel's Barber Paradox [125]. In university development for over 10 years, it is a popular choice for ontology-based software engines such as the Triple system, introduced in section 5.5.3, or the OntologyWorks suite of tools[18]. The respective languages offered by these systems are compiled into XSB prolog for efficient execution. The XSB grounding is hidden from the user via the TRIPLE shell and, in case of the OntologyWorks system, a Java API. XSB is also able to load facts from a relational database via an ODBC interface, allowing to easily access large amounts of data stored in commercial enterprise information systems.

[15] http://herzberg.ca.sandia.gov/jess/
[16] http://alphaworks.ibm.com/commonrules
[17] A typical example is outlined at http://ilrt.org/discovery/2000/10/swsql/
[18] http://www.ontologyworks.com/

CWM CWM is a forward-chaining reasoner specifically designed for the Semantic Web. It currently is available as a prototype version from the W3C website[19]. It reads RDF in XML or N3 serializations and is able to process rules written in the N3 rule syntax. CWM is designed to work like simple command line tools such as awk and sed. The N3 ruleset can also contain queries, which basically are rules with only a body. The result of queries is generated as an RDF XML or N3 output stream. Simple mathematical and string operations are built in via the namespace mechanism shown in the previous chapter's section on the N3 language.

FaCT and Racer The Fast Classification of Terminologies (FaCT) system was written by Ian Horrocks[20] and is a Description Logic (DL) classifier using a highly optimized tableaux subsumption algorithm. Given a set of DL statements, FaCT will compute the class taxonomy and test the satisfyability by trying to establish a model of the concepts. Consequently, FaCT operates solely on the class definitions, not on instance data. It is a tool to detect errors in the definitions at design time. Other systems operating on instance data usually rely on rule engines such as JESS or XSB such as the DAMLJessKB[21] system, for instance. An exception is the Racer Description Logic classifier[22]. Recently, it has gained popularity since it is able to reason over ground facts as well. The TRIPLE system, for instance, uses Racer for the visualization of ontologies.

6.4.3 Storage Systems

According to an OntoWeb survey [103], the storage servers currently available mainly base on either relational databases or XSB in conjunction with the Berkeley DB embedded database system. Almost every system implements a query language, typically RQL, RDQL, or SquishQL. Popular systems are TRIPLE, KAON, Redland, Sesame, and RDFDB. Even though combining the storage system with an inference engine on top of it would be a natural choice, most solutions either do not provide inferencing at all or rely on external software, causing the query and rule functionality to be poorly integrated. TRIPLE is the only exception here. Furthermore, RDF Schema is also not commonly supported. RDFDB is reported to have been tested with 20 million triples [103]. Nevertheless, the systems mentioned still need to be proven in terms of scalability.

6.4.4 Miscellaneous Tools

There is much more ontology related software that could be mentioned in this section. We conclude this list of relevant software tools with a quick overview of

[19] http://www.w3.org/2000/10/swap/doc/cwm.html
[20] http://www.cs.man.ac.uk/ horrocks/FaCT/
[21] http://plan.mcs.drexel.edu/projects/legorobots/design/software/DAMLJessKB/
[22] http://www.fh-wedel.de/~mo/racer/

basic application programming interfaces and ontology merging and validation tools.

Application Programming Interfaces The instant availability of easy to use application programming interfaces like the Document Object Model (DOM) and the Simple Api for XML (SAX) played a big role in XML's success. Similar APIs for RDF, RDF Schema, and DAML have been around for quite some time now. Sergey Melnik's RDF API is a prominent example as well as HP's ARP RDF parser included in the Jena suite. Jena, for instance, also provides RDF Schema and DAML APIs allowing to parse, update, and serialize the respective formats. There is definitely not a lack of systems, however, a standard API, which all parsers should be conform to is still work in progress.

Ontology Merging Tools In the previous sections we saw that the term ontology is used in several different contexts for quite different things. Let alone the complex task of trying to formally describe an application domain with an ontology, we believe that ontology merging will always be a semi-automatic task where software can only support a human knowledge engineer. We also believe that merging ontologies only makes sense if the ontologies are relatively non-formal, i.e. have more of a vocabulary then a logic program character. This view is supported by the fact that none of the three tools, FCA-Merge [135], PROMPT[23], and ODEMerge[24], evaluated by the OntoWeb report [51] can merge ontological rules and axioms yet. All of the systems also require manual input by the user. Formal translation approaches do exist. For example, Ciocoiu and Nau propose ontology based models, which allow a formalized translation from one ontology to another [23]. However, no corresponding system implementing those approaches is available to our knowledge.

The bottom-up and top-down approaches are possible for ontology merging. In the bottom-up method, a set of documents is first marked up using both ontologies. In a second step, a pruned concept lattice is automatically computed from this data. The user can then extract the merged ontology from this lattice and the sets of relation names from the two original ontologies.

The top-down approach of PROMPT and ODEMerge starts by analyzing similarities in the concepts and making suggestions on which of them should be merged. During the merging process, the system tries to match class hierarchies and relationships. In case a dangling reference results from a concept being dropped or renamed, the user is notified and prompted for the appropriate correction.

Ontology Validation Tools Ontology validation tools currently base on very different methods. The OntoClean system, already described in section 2.1.2, employs a formal approach where the consistency of the backbone taxonomy

[23] http://protege.stanford.edu/plugins/prompt/prompt.html
[24] http://delicias.dia.fi.upm.es/webODE/index.html

is evaluated via meta properties such as rigidity and identity. Ontoprise developed two tools called OntoGenerator and OntoAnalyser. The OntoAnalyser system allows simple checks on ontologies using F-Logic queries. Finding ontological names that do not conform to a predefined naming convention is an example. OntoGenerator aims at evaluating the performance of ontology-based applications. Synthetic values are inserted into the ontology to be tested and the runtime of sample queries is determined. Finally, the ONE-T tool checks for formal errors in an ontology. For instance, inheritance cycles or type errors in subclass and instance-of relationships are detected. These are very basic features that are also often checked by the ontology editor directly.

Summary The availability of mature APIs for several programming languages is an important base for the Semantic Web. However, the ontology management is still very immature. Holsapple and Joshi suggest that ontology merging might not be a promising solution since the merging is likely to produce both poor results and high cost for the merging process [85]. Except for the OntoClean and OntoGenerator systems, the approaches for ontology validation are still quite primitive.

Chapter 7

Design Choices

This chapter will first give an architectural overview of our suite of software components for building intelligent applications based on ontologies and Semantic Web standards. The overview in section 7.1 is followed by the detailed design of the data storage, ontology, and rules layers which form the core of our solution. After this, section 7.3 shows how editing tools are interfaced with the core layer in order to author ontologies and base content. Since we made a strong point about data reuse, section 7.4 outlines three approaches on how we can tap into external datasources. The last two sections focus on the flow of input and output between the user and the system. Furthermore, a feature list of the SmartGuide and SmartAPI applications is given. We focus on describing the major design decisions, the choices on which existing solutions and data are integrated, as well as the purpose of the integrated component in the overall picture.

7.1 Overall Architecture

Clearly, the Semantic Web languages have been designed with a very distributed architecture in mind. We identified this as a key requirement in chapter 4. Consequently, our overall design builds on the idea of every user having a personal assistance agent running on a desktop at work, a laptop, or a small portable device such as a personal organizer or a cellular phone. Additionally, agents are available on the corporate Intranet or the Internet taking over similar functionality of web portal sites, search engines, or document repositories as, for instance, the solution presented in [45]. The difference is that a human user does not contact these agents directly, but via other agents which might act on behalf of a human user.

This ecosystem of agents will function and interact like a group of humans. Different views and logical organization of content are explicitly supported by the ability to customize agents. Agents can also have areas of specialization. Such agents will only be dedicated to serving a small set of other agents. In

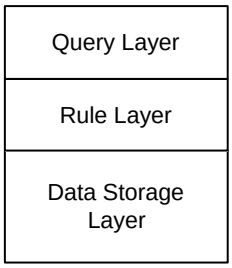

Figure 7.1: The data, rule, and query layers are the foundation of our architecture.

turn, there can be agents with a very general focus, which can be accessed by the entire agent population. In the following, we explain the architecture from the point of view of a single agent. We then go on and explain how the agents can be interfaced.

7.2 Core Data, Rule, and Query Layers

The first question that has to be addressed is which interpretation of ontology should be used. Since our goal is the ontology-based construction of software, it is clear that a more formal approach needs to be taken. A simple vocabulary can be the basis for data integration but not for actually constructing software. We therefore define an ontology to be a backbone taxonomy, a set of relationships between the classes of the taxonomy, and a set of rules representing additional domain knowledge.

Clearly RDF will be used for data representation since the tree structure of pure XML is not sufficient to model a metadata network of connected resources. RDF Schema provides the mechanism for establishing the class taxonomy and typed relations among them. RDF Schema is also the base of other higher-level languages such as OWL, indicating its high level of acceptance. We decided not to use description logic based languages like OWL or OIL. The reasons for this are twofold. Firstly, as we pointed out in the software survey, the tool support for these languages is relatively poor at the moment. Secondly, we agree with Hayes' argument that the more powerful description logic constructs are somewhat awkward to use [79]. As a matter of fact, most of today's description logic ontologies mostly only use the simple features that are comparable to what is offered by RDF Schema.

Instead of using description logics, we opted for including simple recursive datalog rules for enriching the ontology with more expressive domain knowledge. We opted against more expressive logics such as first order logic, since we believe they are not necessary for our general application area of intelligent document retrieval and help systems [34]. Shallow reasoning performed on a

wide range of semantically enriched metadata will be enough to solve the problems in these areas [35]. Instead, we allow both procedural attachments in rules and reactive rule behavior. The rationale behind this decision is to be able to specify the software solely on a rule level, relieving much of the burden from the developer. Furthermore, using RuleML as a rule representation format is an obvious choice. The current version supports datalog. The reactive rule and procedural attachment syntax mentioned before are currently being specified.

Figure 7.1 shows the hierarchy of the data, rule, and query layers. The base data repository stores statements on a triple basis. It is the responsibility for the datastore to enforce the restrictions specified in the underlying RDF Schema definition. RuleML plays a role in all three layers. The rule layer deducts information from the given rules and data. Therefore it lies between the data and the query layers. Queries can be formulated as rules without rule head. Integrity constraints are realized as rules with a special procedural attachment, i.e. raising an error.

Following the design requirement of using mainstream technology, we provide the OntoJava and OntoSQL implementations of this core specification. The details are given in the next chapter in sections 8.1 and 8.2.

7.3 Data and Ontology Editing

After the conceptual architecture of the core components has been illustrated, we explain how data and ontology are edited and deployed. Figure 7.2 shows the core components on the right and the three editors on the left side, one for each RDF, RDF Schema, and RuleML. For reasons of simplicity, we chose to maintain ontologies in files in their native RDF Schema and RuleML formats, rather than choosing a more collaboration-friendly approach as the ones outlined in section 4.2.1. The files are deployed to the core system by a set of converters. This has two advantages, the first being that we can reuse Protégé as an editor for both RDF and RDF Schema. Secondly, a conversion procedure is necessary anyway once third-party rulesets and schemata are to be downloaded and used.

Besides Protégé, we also use a classical database client server setup for data entry. Specifically, Microsoft Access serves as an RDF editor in conjunction with a SQL Server database created by OntoSQL. Like any XML language, RuleML is also not suitable to be edited manually. We developed the Prolog2RuleML system, presented in section 8.4.5, which allows to use the much more convenient Prolog rule notation with any text editor.

7.4 Interfaces to External Data Sources

As we pointed out in the beginning of this chapter, one agent might only have a very limited set of resources stored locally. Consequently, the tight integration with other peers and agents is crucial. We opted for three major ways of interfacing a single agent with other data sources. The most obvious way is

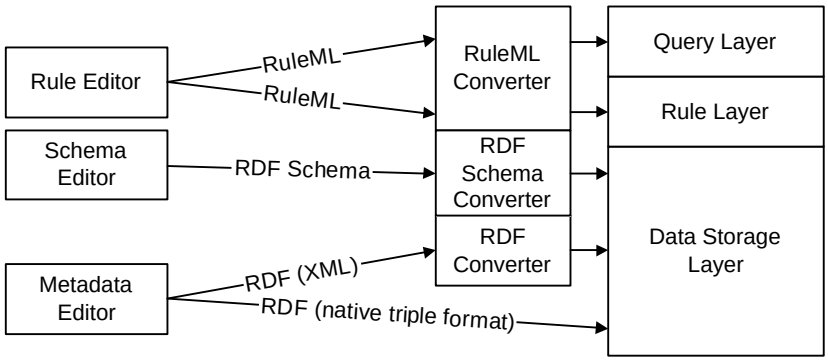

Figure 7.2: A set of editors is used to create the ontology and the content. The XML-based markup, generated locally or received from a third party, is loaded into the system using converters.

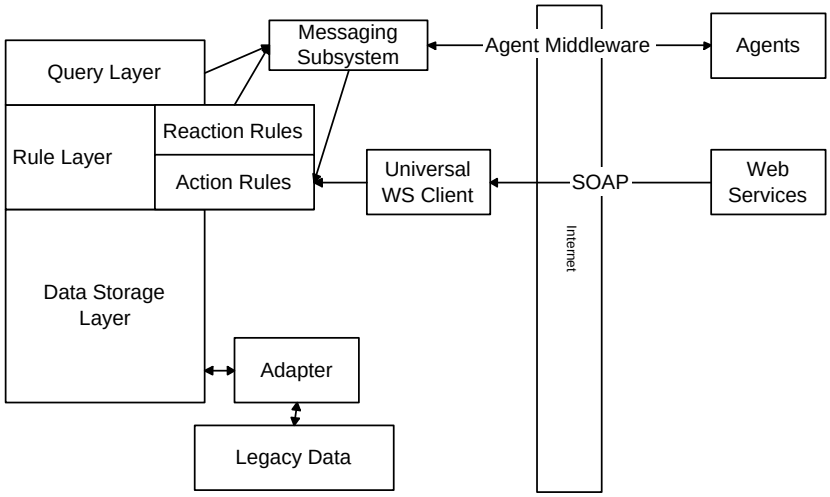

Figure 7.3: Agents deployed on other peers, Web Services, and legacy systems represent the three kinds of external information sources.

message exchange with other agents. Figure 7.3 shows this channel on the top right. The agent middleware can use arbitrary transport protocols like SOAP over HTTP or Java RMI. On a message level, we use two KQML-style message types that refer to a shared ontology. A `query` message triggers a response back to the calling agent with results obtained from the query layer. The purpose of `tell` messages is to spread information around to other agents.

While this message-passing paradigm is a standard feature of any agent system, our interface between the local knowledge base and the messaging system is innovative. It bases on the observation of Boley et. al., that an agent can be entirely specified in a declarative way [14]. The figure shows a reaction rule component, which handles incoming messages as well as an action rule part, which can actively send messages. This means that we use rules to specify the behavior of the agent rather than following the traditional approach of writing code in an iterative programming language to do so. For our OntoAgent implementation, described in section 8.3, we opted against a full-blown agent infrastructure like FIPA-OS. In the spirit of Web Services, our agent middleware bases on simple and well-established Internet standards.

While this inter-agent messaging might be the prime way of communicating in the future, currently this is only an option within a local test bed since no public agents are deployed. Therefore, other gateways are necessary. Even though Web Services have yet to take off for applications other than enterprise application integration, they are a promising technology likely to have a major impact. In the light of this observation, we developed a universal knowledge base to Web Service gateway called Web Service Description Framework (WSDF), which we explain in detail in section 8.4. Simply by enriching the existing interface description with semantic information, WSDF allows our agents to dynamically invoke Web Services without previous knowledge of its API. Finally, probably the most important sources of information are existing applications. This can range from ones personal calendar to a corporate information system. We suggest an easy extension of our OntoSQL engine, which allows existing information of a relational database to be viewed as Semantic Web data.

7.5 Gathering Information from the User

Besides obtaining structured data from other applications, the user is the other prime source of information. Figure 7.4 illustrates that a model of the user is kept in the core datastore. This allows us to integrate the user model with the other available information and rules. Consequently, the data is logically represented as RDF triples that are again conform to the core ontology. Physically, these applications most likely will load the data directly via the native datastore interface, rather than using the XML serialization. We envision two ways of gathering information. First of all, the user should be able to make simple statements using natural language. In order to translate these statements into RDF, we developed a small solution called OntoLang (see section 8.6). In the spirit of our ontology-based software engineering method, OntoLang uses the

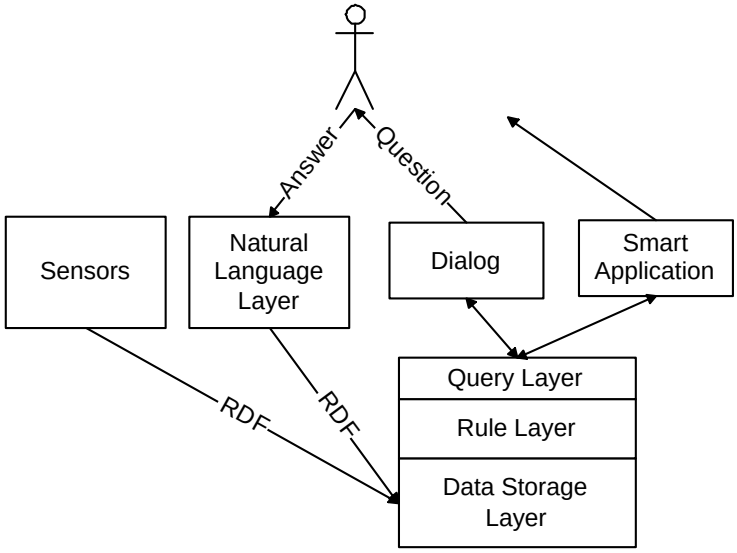

Figure 7.4: A smart application can present selected information to the user. In turn, the user can feed information into the system by initiating a natural language dialog or simply via controls placed on the user interface.

ontology for this translation process.

Other, simpler but not less effective ways are special user interface controls. A button might be labeled "I am interested in topic X". The corresponding action of asserting $interestedIn(user, X)$ into the knowledge base can simply be hard coded in the event handler. Dynamically placing such controls on the user interface is done by the SmartGuide application, introduced in the chapter 9. Finally, special sensors can feed information such as a GPA position or, in the case of a helper application, the local system's configuration parameters, for instance.

7.6 Applications

We developed two intelligent applications, SmartGuide and SmartAPI. SmartGuide addresses document retrieval while attempting to incorporate as much contextual knowledge as possible. The knowledge base combines a user model with domain knowledge. In conjunction with rich content metadata, SmartGuide leverages this information in order to suggest documents, which are likely to be useful, or which provide the answer to a problem the user is facing. The natural language statement parser, mentioned in the previous section, is augmented with SmartDialog, a clarification dialog system. SmartDialog analyses the user model and the current suggestions in order to optimize the recommen-

dations. The details of this dialog system are given in section 9.3.

A second application, SmartAPI, serves a more specific problem domain, i.e. assisting programmers with their job. Rather than documents being tagged with metainformation, SmartAPI bases on semantically enriched APIs. In combination with an ontology on frequent programming tasks, comparable to the MIT Process Handbook, SmartAPI can then automatically generate small program sequences, saving the developer from having to study the documentation and find the respective useful parts. SmartAPI is discussed in section 9.4.

Chapter 8

Core Technology

After the coarse architecture has been introduced in the previous chapter, we will now present our individual technological contributions in detail before providing application examples in the following chapter. First, sections 8.1 and 8.2 introduce the OntoJava and OntoSQL inference engines. Both solutions base on mainstream technology, however, they are very different in that OntoJava is a solution for executing rules on a small scale, possibly on a portable devide such as a cellular phone or a personal organizer. Basing on relational databases, OntoSQL is targeted more towards the backend server area. The OntoAgent framework, explained in section 8.3, provides a solid engineering basis for the development of agent-enabled applications. An application of OntoAgent from the area of online learing will be demonstrated in the following chapter. The integration of external data sources is at the center of attention in section 8.4, which illustrates our Web Service Description Framework. We conclude with OntoLang, a small system that shows how Semantic Web technology can also be leveraged in yet another context, namely language understanding.

8.1 OntoJava

The core idea behind OntoJava is that a directed, labeled RDF graph lends itself to being modeled using objects and object references of an object oriented programming language like Java. One of the main benefits is that many aspects of the RDF Schema semantics are actually enforced by the Java compiler and interpreter. An object located in a main memory database represents every resource with its URI. We will examine to what extent the restrictions on the graph's arcs imposed by RDF Schema can be enforced by the language's type system and which restrictions are necessary. Apart form this discussion, we introduce the OntoJava cross compiler, that automatically converts RDF Schema, and RuleML into a set of Java Classes that act as a combined main memory object database with a built in forward chaining rule engine[1]. Figure 8.1 illus-

[1]Initially, the forward-chaining approach was introduced by the OPS5 system [26]

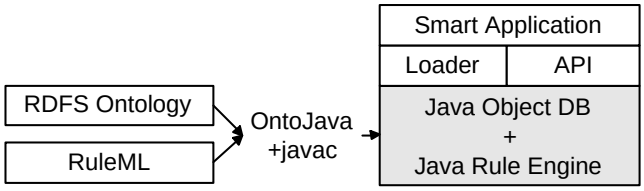

Figure 8.1: The object database/inference engine generated by OntoJava can be loaded with RDF data and accessed by an application

trates this process and how a smart application can interface with the object database. An object-oriented language like Java offers many advantages and therefore the section concludes by proposing Java-enabled reaction rules and ways to customize the generated classes.

Conceptually, OntoJava consists of three major parts. The first part in section 8.1.1 is the mapping of RDF Schema classes to classes in an object oriented sense. This was pioneered by Cranefield [27] who suggests a mapping of UML to both Java classes and RDF Schema ontologies as the layering of Semantic Web languages from figure 5.1 suggests. This approach is also supported by Grosof [70], who is also our co-lead in the RuleML ontology combination subgroup. The second part in section 8.1.2 is the explicit association of predicates in rules with properties of the ontology. Finally, the third part in section 8.1.3 deal with rule compilation. This work was inspired by previous approaches of compiling Prolog into C execution engines [24, 124].

8.1.1 Mapping the Class Taxonomy

A class taxonomy, which can be expressed in RDF Schema, is the backbone of an ontology. Domain concepts are represented as classes with a hierarchy being imposed by the `subClassOf` property. Consider the following example:

```
<rdfs:Class rdf:about="&pre;Person">
    <rdfs:subClassOf rdf:resource="&pre;Animal"/>
</rdfs:Class>
```

OntoJava maps every RDFS class into a Java class. The `subClassOf` property is similar to inheritance in object oriented systems with respect to both being transitive relationships defining the class hierarchy. However, the RDF Schema version is more flexible than its Java or C# counterpart since it allows multiple inheritance. OntoJava is therefore not able to handle multiple inheritance in the RDFS source at the moment. Section 8.1.6 outlines a simple solution for this limitation where RDF Schema classes are converted to interfaces which are then implemented. The RDF Schema example above is mapped to a class `Person` that inherits from `Animal`.

```
public class Person extends Animal
```

8.1.2 Mapping Properties

RDF Schema properties are defined with a domain and a range. Using the directed labeled graph metaphor, an arc's label identifies which property it refers to. The types of the resources connected by the arc must be the same class that's listed as the property's range or domain, or a subclass of it. The range is decisive for the type of the arc's source, the domain for its destination's type. Besides a class, the range of a property can also be a literal. Properties of classes can be defined as follows:

```
<rdf:Property rdf:about="&pre;isParentOf"
    <rdfs:domain rdf:resource="&pre;Person"/>
    <rdfs:range rdf:resource="&pre;Person"/>
</rdf:Property>

<rdf:Property rdf:about="&pre;name"
    <rdfs:domain rdf:resource="&pre;Person"/>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
```

Properties with a literal as the range are mapped directly into instance variables, which is quite straightforward. Note that it is easy to change this to get/set access methods. This becomes necessary if changes need to be tracked to support an undo operation or if multiple inheritance basing on interfaces is implemented. Relations to other instances are represented by a collection of references to other Java objects. This is the natural representation of a directed labeled graph in Java. The RDF Schema example above causes to following methods and variables to be defined in the class `Person`:

```
public class Person extends Animal {
    public String name;

    private HashSet isParentOf = new HashSet();
    public void putIsParentOf(Person p) {
        has.add(p);
    }
    public boolean getIsParentOf(Person p) {
        return has.contains(p);
    }
    public HashSet getAllIsParentOf() {
        return isParentOf;
    }
}
```

As sections 8.1.3 and 8.1.5 will explain further, the get/put access methods play an important role for the rule mechanism or when restrictions on the variables are to be checked. The corresponding get and put methods ensure that

the appropriate data types, as defined in the RDF Schema property definition, are used. Note that the Java compiler enforces this. RDF Schema allows a property to have multiple domains, but only a single range. Multiple domains would result in several Java classes having variables and methods with the same name, which is no problem.

In terms of logic, the assertion $isParentOf(a, b)$ corresponds to the invocation of `a.putIsParentOf(b)`. The query $isParentOf(a, b)$ can be answered by calling `a.getIsParentOf(b)` whereas `a.getAllIsParentOf()` yields the answer to $isParentOf(a, X)$. Finally, $isParentOf(X, Y)$ is answered by iterating over all person instances and calling `getAllIsParentOf` again.

Ternary relations are not part of RDF and are handled by a workaround of using an intermediate pseudo resource. A similar scheme can be applied to OntoJava by promoting a relation from a simple object reference to having its own object, which can then point to more than two constituents.

8.1.3 Mapping Rules

The two basic rule evaluation strategies are forward and backward chaining. Obviously a imperative language environment lends itself to forward chaining, where the rules are executed in an if-then fashion. Rule implications are simply asserted into the fact base as new facts. The popular Java Expert System Shell (JESS)[2] is another representative for this evaluation style, implementing the RETE [56] algorithm to ensure efficient rule execution.

Restrictions on the rules' expressiveness, namely disallowing negation, make sure, that exactly one minimal model exists for the set of rules [46]. A minimal model describes the smallest possible fact base, where for any variable assignment on any rule body yielding true, the rule's head is also true. Therefore, if the minimal model is contained in the current fact base, no more rules will fire and assert new facts. The order in which the rules are evaluated does not matter, since the fact base cannot shrink. If stratified negation is allowed, rules containing the not operator must be executed last. This feature is not yet implemented in OntoJava.

In OntoJava, each rule from the RuleML base is converted into a static method. The brute force approach would be to check the right side for all possible bindings of the free variables for each rule until no new assertion occurs, which would be quite inefficient. OntoJava implements the following optimizations. Every time an update takes place on a specific property of an object, all rules are evaluated, that contain that property in their right side, i.e. the rules that are potentially affected by the change. This means that the rules are checked incrementally. Consider the rule $isUncleOf(A, C) \leftarrow isBrotherOf(A, B) \wedge isParentOf(B, C)$ which has some label, say rule5. Further assume that this is the only rule, which has the `isBrotherOf` predicate on the right side. This causes the following call to be placed in the `putIsBrotherOf` access method in the person class:

[2] http://herzberg.ca.sandia.gov/jess/

```
public void putIsBrotherOf(Person p) {
    if (!(isBrotherOf.add(p)))
        return;
    Rule.rule5(this, p, null);
}
```

First, the new relationship is stored by inserting it into the set datastructure. If the object was already in there, the add method returns false and the call returns. Otherwise, the rule is activated. Since the rule has three variables, three parameters are passed. We also know that the added relation is the only change to the database. Therefore, A and B, the first and second variable, must be bound to `this`, the current object, and `p`, the object inserted into the datastructure. Only C must be bound to all persons. Here, we can use the free variables' type information. A free variable appearing with a `isParentOf` predicate can only be bound to persons. Thus, instead of iterating over all objects, we only need to iterate over persons. The following shows the rule's code:

```
public static void rule5(Person a, Person b, Person c) {
    if a==null, iterate a over all persons
    if b==null, iterate b over all persons
    if c==null, iterate c over all persons
    with combinations of (a, b, c) {
        if (a.getIsBrotherOf(b) && b.getIsParentOf(c))
            a.putIsUncleOf(c);
    }
}
```

Note that the call to `putIsUncleOf` can again trigger other rules that are activated in the respective put method.

Obviously this approach will not perform efficiently for rules with many free variables, since all but two of the combinations of free variables need to be tested by nested loops. We are planning another optimization here by performing short circuit evaluation of parts of the condition at the outer loops. This should drastically reduce the number of combinations that need to be checked.

Rules with large amounts of free variables being checked in nested loops resemble a relational join very much. This leads to section 8.2 where we examine how rules can be evaluated top-down by an SQL engine.

Obviously OntoJava currently implements only a brute force algorithm. Many optimizations have been suggested in the past. Section 8.2.8 will outline some of the most common approaches using in deductive and even relational databases.

8.1.4 Property Inheritance

RDF Schema defines the core property `subPropertyOf` with the following semantics:

$$parent(a,b) \leftarrow subPropertyOf(father, parent) \wedge father(a,b) \tag{8.1}$$

This rule can be rewritten into a set of rules:

$$\{p(a,b) \leftarrow c(a,b) \mid subPropertyOf(c,p))\} \tag{8.2}$$

Rather than implementing special handling for subproperties in the generated code of the access methods, we decided to reuse the built-in rule mechanism and include a rule for each subproperty relation defined in the RDF Schema source according to equation 8.2.

The transitivity of `subPropertyOf` is handled by the fact that asserting the parent property through the rule will in turn trigger another rule asserting the grandparent property, and so on.

8.1.5 Constraints

RDF Schema currently has no mechanism for defining further constraints. However, it is clear that this would be a valuable addition. DAML+OIL, for instance, offers a construct like `daml:maxCardinality` to restrict the number of outgoing arcs from an object. Dealing with a constraint violation includes detecting the violation, notifying the user, and finally reversing changes made to the data repository. Constraints cannot be checked incrementally, since partially inserted data might reflect an inconsistent state. Once the user issues some sort of commit, the conditions must be checked. It is definitely possible to generate constraint checks into each class. Violations could be flagged using Java exception mechanism. To make sure the operations of a transaction can be reversed, each object might clone the internal datastructures and retain a copy of the old values until a successful commit is issued.

8.1.6 Multiple Inheritance

Unlike C++, Java does not allow a class to have more than one superclass. The reason lies in ambiguities in which implementation of an inherited method `m` is to be called, if both super classes implement `m`. Java solves this ambiguity by disallowing multiple inheritance for classes and offering multiple inheritance for interfaces only. Since an interface only contains the method signatures and not the implementations, it is always clear which method a caller refers to. Consequently, a Java interface instead of a class will be generated for each RDF Schema class. A class `StudentWorker` that is derived from both `Student` and `Worker`, results in the following Java interfaces:

```
public interface Student extends Person { ... }
public interface Worker extends Person { ... }
public interface StudentWorker extends Student, Worker { ... }
```

An implementation class is generated for each interface. Note that the complete implementations of `StudentImpl` and `WorkerImpl` need to be repeated in `StudentWorkerImpl`. This is not really a problem since the implementation is generated automatically anyway.

```
public class StudentWorkerImpl extends PersonImpl
                               implements StudentWorker {
    ... implement Student, Worker, and StudentWorker methods
}
```

Finally, the following code then creates an instance of the class:

```
StudentWorker sw = new StudentWorkerImpl();
```

8.1.7 Defining Instances

RDF Schema defines the `type` property. A resource can be declared to be an instance of a class via a (ResourceURI, type, ClassName) triple. Unlike regular triples that are represented by storing a reference to the object in the respective data structure of the subject, the OntoJava framework handles this triple by instantiating an object of type ClassName:

```
ClassName obj = DB.createClassName(ResourceURI);
```

Thus, the triple is represented by the Java expression `obj instanceof ClassName` being true. Using a factory method offers further flexibility when custom behavior is to be added to the generated classes. Section 8.1.8 describes this mechanism in more detail. OntoJava also maintains global data structures allowing convenient access to the objects stored in the main memory database. Every object that is created by a factory method is immediately inserted in a global hash set using the resource's URI as the key allowing for efficient retrieval. Furthermore, a set data structure is maintained for each RDF Schema class to be able to quickly answer a query asking for all Person instances in the database.

Since the type predicate is treated in a special way, consequently, checking if a resource is of a specific type is not preformed by searching a type-predicate data structure but via the Java `instanceof` operator. The respective special handling for type predicates is built into the OntoJava software.

This behavior reveals a restriction of the OntoJava system. While in RDF, a class can be defined to be an instance of several classes, this is not possible in an object oriented programming environment. An instance can be viewed or cast to the interfaces and super classes related to the instance's class, but never to a completely unrelated class.

A solution for this limitation would be to implement the typing mechanism with own code instead of having it (partly) handled by the programming language environment. However, this would greatly complicate the code and make

the application programming interfaces less descriptive, reversing major advantages of the solution. Furthermore, the Java compiler would no longer be able to catch RDF Schema violations at compile time. Since these points are important advantages of OntoJava, it seems reasonable to not opt for such a workaround and accept this limitation.

A workaround suggested for handling multi-class membership in classes $C_1, ..., C_n$ with Protégé is to create a new class C which inherits from $C_1, ..., C_n$.[3] An instance of this new class has the multi-class property, however, this would require creating, compiling, and loading new classes into the virtual machine at runtime.

8.1.8 Extending the Generated Classes

Extending the generated classes with own functionality can customize the system. As was mentioned in section 8.1.7, we use factory methods to create new objects in the database. Using the abstract factory design pattern [61], the user can replace the object factory with an own version that creates the customized classes instead of the original ones. The inference functionality remains since the new classes only extend the existing ones.

8.1.9 Namespaces

OntoJava's handling of namespaces is fairly rudimentary at the moment. One option, which was used in the examples shown so far, is to omit a certain namespace prefix from the RDF Schema source. Alternatively, the entire URI is used in class names. Here, we replace characters that cannot appear in variable names with an underscore. Method names can still be given without the namespace prefix, since they do not have to have unique names like the classes do. The URL http://www.w3c.org/onto/Concept would become the class `http__www_w3c_org_onto_Concept`. The drawback of this simple approach is that it results in very lengthy class names and makes the API quite unreadable. Nevertheless, this allows different RDF Schema sources to be combined into a single application.

Alternatively, a namespaces can be associated with Java packages. The naming convention for packages is already based on Internet DNS names. For instance, the Document Object Model interface specified by the W3C is located in the package org.w3c.dom. If the folders given in the URL are converted into further subpackages, the class name would only have to contain the URL's last part. The URL above would then become the class `org.w3c.onto.Concept`. This is definitely more natural, and with the proper import statement, the class can even be referred to only as `Concept`.

[3] http://smi-web.stanford.edu/projects/protege/protege-rdf/protege-rdf.html

8.1.10 Reaction Rules

Rather than asserting new facts, reaction rules perform an operation like sending mail or printing a message to the console. Usually the inference system defines a set of commands like `print` that can be used in reaction rules. Since we are dealing with a Java environment, it seems quite natural to allow Java statements to be embedded in reaction rule heads. The following example calls a web application, using a free variable as a parameter:

```
<_head><java>
    runtime.Loader.load("http://host/servlet/SearchGate?flight="
                        + <var>F</var>.name);
</java></_head>
```

The web application's RDF output is then loaded into the database using a runtime library. The implementation is quite simple. Instead of generating the call to the assertion method as shown in section 8.1.3, the code from the rule is printed. Only the variable references need to be replaced.

This solution is flexible, but it also seems fairly proprietary. After all, rules are supposed to be exchanged across any platform and system. However, any kind of reaction rule command is proprietary. It seems natural to reuse an existing platform. Furthermore, applets apply the same concept. With Java being a cross-platform language and a virtual machine being installed on a large fraction of hosts, it seems fairly reasonable to load not just RuleML, but also some accompanying Java libraries for sending email etc.

8.1.11 Further Features

In addition to the features presented so far, OntoJava has two more aspects. Since the active database is only held in main memory, we provided for a serialization and deserialization facility which base on the mechanisms build into the Java APIs. This way a snapshot of the current state of the object database can be stored on disc. Besides the predicates coming from the ontology, OntoJava has some simple type comparators for numbers, strings, and dates built in.

8.1.12 OntoJava Implementation Architecture

The design and implementation of the OntoJava cross-compiler strictly follows object-oriented principles. Figure 8.2 shows the class inheritance graph. OntoJava bases on both an RDF and a RuleML parser. The RDF parser is not shown in the figure, since it is provided from an external library. The RuleML parser classes can be seen on the left with the RuleML package prefix. Note that the class names correspond directly to the XML element names found in the RuleML syntax DTD. The generic RuleML parser classes are now extended within the `convert` package. This is enabled via an abstract node factory, which produces instances of the desired RuleML parser extension. Our converter extension adds functionality allowing the nodes to transform themselves into Java

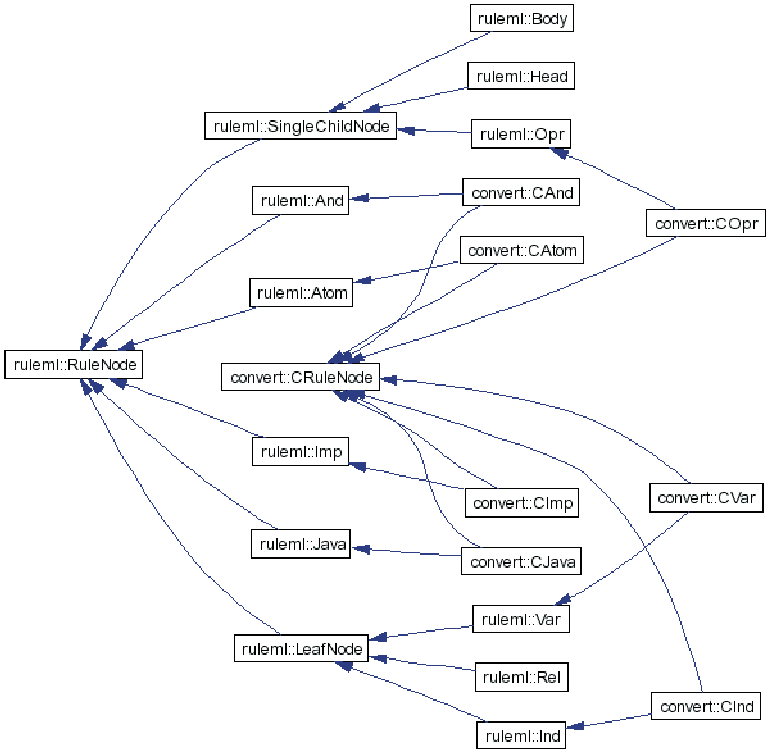

Figure 8.2: Class inheritance diagram of the OntoJava package

statements. The methods required for doing this include collecting rule body variables or the actual print order. These methods are specified in the central `CRuleNode` interface, which is implemented by all converter classes. This architecture proved to be very useful once we exchanged the Java converter with the SQL converter for the OntoSQL system.

Figure 8.3 shows a class collaboration graph from the RuleML parser part. The solid arrows denote subclass relationships; the dashed lines indicate that instances of one class have references to instances of another class. A rule has a head and a body. These are both `SingleChildNodes`, which in turn reference one node each.

Figure 8.4 shows a class collaboration diagram from the converter module. A rule variable references a class, which specifies its type information. The class holds three references to the predefined types object, class, and string. The object and class types were introduced to allow an `instanceOf(object, class)` built-in predicate.

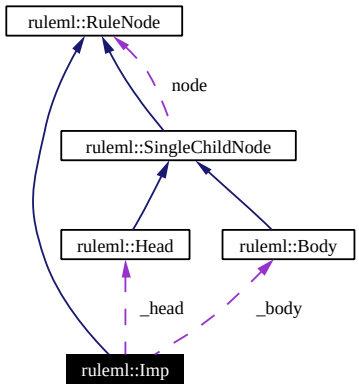

Figure 8.3: Class collaboration diagram .

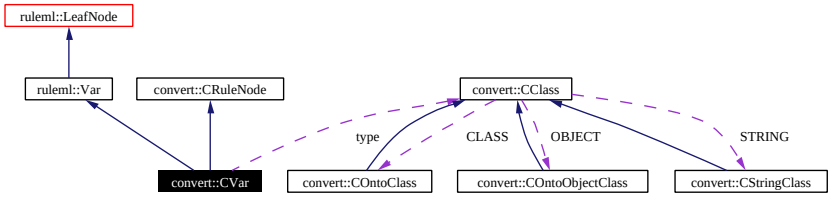

Figure 8.4: Class collaboration diagram .

8.2 OntoSQL

While OntoJava's goal is to provide simple inferencing support for mobile and small devices, OntoSQL aims at backend server systems. Databases can be found in virtually every enterprise. Jim Gray even calls them "the bricks of cyberspace". Furthermore, databases have already been used as logic engines. The XSB deductive database, for instance, provides an ODBC backend into relational database systems [126] where database cursors are used to retrieve the base facts which are normally loaded from the XSB fact base. Rather than simply providing a storage repository for facts, the database engine can also execute SQL queries which a Prolog to SQL compiler [32] generates from queries passed into the `odbc_query` built-in predicate. Therefore, it seems natural to reuse database technology for Semantic Web solutions. Since our design goals only require simple reasoning, we decided against an approach where a powerful reasoner like XSB is attached to the database. Some initial test with the XSB database backend also revealed serious performance problems caused by multiple open cursors, general failures due to ODBC errors, and the tendency of XSB to load the entire database tables into main memory. Another reason for sticking with a basic database is the fact that we want to be able to use the database's SQL interface directly.

This section first compares the similarities and differences between the datalog-oriented RuleML version 0.8 and SQL. We pick up and elaborate ideas presented in [46] and the Edutella white paper [116], where a mapping of the RDF-QEL query language to SQL is discussed briefly. The second part introduces the OntoSQL system, which is able to automatically generate the necessary tables and views in a relational database system[4], enabling it to act as a RuleML engine.

8.2.1 Mapping Datalog Queries to SQL

For this discussion we assume that the information is stored in a single table containing fact triples. We have three string columns, subject, predicate, and object, which are all part of the composite primary key, preventing the application to insert the same triple twice.

```
create table fact (
    subject   char(255),
    predicate char(255),
    object    char(255),
        primary key (subject, predicate, object)
);
```

This storage schema is obviously very simplistic, yet sufficient for our analysis. Possible alternatives deal with models and optimize space requirements by

[4] In SQL-type relational database systems a view is actually defined by a set of rules [140].

introducing namespaces and URIs as entities in the schema.[5]

We distinguish between implication rules and queries. A rule $A \leftarrow B$ states that A_{var} is true if B_{var} is found to be true for a certain variable assignment. A query finds all variable assignments for which the search condition is true. We can generalize a query to be an implication rule with an empty left side: $\leftarrow B$.

We examine the mapping of queries first. Handling rules is complicated by the fact that the right side can depend on other rules. Section 8.2.2 explains how this is handled using SQL views.

Queries on a single predicate are mapped as follows:

```
isFatherOf(X, Y)

select * from fact where predicate = 'isFatherOf'
```

Conjunctions are translated to self-joins on the fact table. The join condition is determined by the occurrences of the variables in the query. Here, the object of the `isParentOf` triple must be the same resource as the subject of the `isBrotherOf` triple.

```
isParentOf(X, Y) and isBrotherOf(Y, Z)

select * from fact a, fact b where
    a.object = b.subject and
    a.predicate = 'isParentOf' and
    b.predicate = 'isBrotherOf'
```

The SQL union operator can handle disjunctions. Note that generally, a disjunction like $A \leftarrow B$ *or* C can be written as two rules or queries $A \leftarrow B$ and $A \leftarrow C$, and vice versa.

```
isBrotherOf(X, Y) or isSisterOf(X, Y)

select * from fact where predicate = 'isBrotherOf'
union
select * from fact where predicate = 'isSisterOf'
```

8.2.2 Mapping Datalog Rules to SQL

As mentioned before, rules are similar to queries, except for the fact that implication results can influence other rules. Consider the following simple example consisting of two rules:

```
isSiblingOf(X, Y) <- isSisterOf(X, Y)
isRelatedTo(X, Y) <- isSiblingOf(X, Y)
```

[5] Sergey Melnik collected a list of proposals about ways of storing RDF data in relational databases at http://www-db.stanford.edu/~melnik/rdf/db.html

The second rule depends on the first rule via the `isSiblingOf` property. To answer a query asking for all siblings, we could use the following SQL query that is embedded in a view:

```
create view isSiblingOf as
    select * from fact where predicate = 'isSiblingOf'
    union
    select subject, 'isSiblingOf', object from fact
        where predicate = 'isSisterOf'
```

The first subquery gets all `isSiblingOf` facts from the database. This is necessary since the user might assert a sibling relationship, if the gender of the sibling is not known. The second subquery processes the first rule. Except for the select clause, that explicitly states the predicate used on the rule's left side, the query corresponds directly to the datalog query `isSisterOf(Y, Z)`. Wrapping the query in a view allows us to treat the view's name `isSiblingOf` as a table. The DBMS internally resolves the underlying SQL statement.

If a query for all related resources is posed, the same mechanism can be applied. Note that the SQL query references the view defined above:

```
create view isRelatedTo as
    select * from fact where predicate = 'isRelatedTo'
    union
    select subject, 'isRelatedTo', object from isSiblingOf
```

Since even this simple example results in quite elaborate queries, it makes a lot of sense to encapsulate the queries retrieving every triple with a given predicate in a separate SQL view. These views can then be reused in other views or queries, as demonstrated in the last example.

Now the user can run the SQL query `select * from isRelatedTo`. The SQL engine of the DBMS handles all rules. While this is very convenient, we must rely on the DBMS's optimizer to efficiently handle the range of joins and union operations triggered by a simple query like the one above.

8.2.3 Recursive Rules

Recursive rules, i.e. rules where the predicate on the left side also appears on the right side, cannot be handled with the methodology presented above. In order to make sure that the rule set can be converted, a predicate dependency graph is established. It contains a node for each predicate. Whenever a predicate A appears in the body of a rule which has the predicate B in its head, we define B to be dependent on A and we draw an arc from A to B. The rule set can be converted if the dependency graph is free of cycles, with the exception of a cycle caused by a linear recursion of a predicate with itself. These cases can be handled by recursive queries, which are defined in the SQL-99 standard:

```
create view isAncestorOf as
    with rec(subject, 'isAncestorOf', object, level) AS (
        select * from fact where predicate = 'isAncestorOf'
        union all
        select subject, 'isAncestorOf', object from Parent
        union all
        select a.subject, 'isAncestorOf', b.object, level+1
            from rec a, Parent b
            where a.object = b.subject and level < 9
    )
    select * from ancestor;
```

This view corresponds to $isAncestorOf(A, C) \leftarrow isAncestorOf(A, B) \wedge isParentOf(B, C)$. The first and the second subquery get all existing ancestor information and combine it with the parent information. This forms the basis for the recursive third subquery. DB2 does not terminate the query, if the data contains a cycle. The `level` parameter registers the depth of the recursion and prevents an endless loop by restricting the search to a certain depth.

We model every predicate as an SQL view. From the dependency graph we can conclude, that, with the exception of the case above, no view definition will be recursive. This allows us to run SQL queries similar to the ones shown in section 8.2.1 from arbitrary database clients. The rules will then be executed transparently by the system.

Note that even the fact table might actually be a view of a regular ER schema. This way, up to date information can be used without any need to upload data from an enterprise information system.

8.2.4 Further Mapping Possibilities

Besides the mappings presented so far, two more possibilities have been suggested [32]. These are not yet included in the OntoSQL system mainly because RuleML does not support these primitives yet. Note that we use the conventional notion of arbitrary n-ary relations in the Prolog and SQL sense here. Negation can be handled by using a negated exists subquery, as shown in the following example:

```
<- employee(X) and not manager(X)

select id from employee where not exists
    (select * from manager where id = employee.id)
```

Furthermore, it is suggested to group by free variables. Consider wanting to compute the average salary for each employee role, given an employee relation with name, role, and salary attributes. The underscore used in place of the employee name denotes a "do not care":

```
<- avg(S, employee(_,R,S))
```

```
select avg(salary) from employee group by role
```

8.2.5 Building Applications with OntoSQL

OntoSQL is used to define the tables and views necessary. Once this process is complete, an application program can simply query the views using the generic DBMS SQL interfaces. Since we found DB2[6] to be the only DBMS currently supporting SQL 99 recursive queries, OntoSQL contains the following workaround in order to support other DBMSs as well. Rather than a recursive query, a sequence of self-joins on the fact table is preformed and the results combined by the union operator. This yields the correct results if the longest transitive closure sequence is smaller than the maximum number of self joins performed on the fact table.

8.2.6 OntoSQL Architecture

Large parts of the OntoJava cross-compiler architecture were reused for OntoSQL. Figure 8.5 shows that in the class hierarchy, the `ontosql` package takes the place of the Java converter seen in figure 8.2. A slight change manifests itself in the interface `SQLAble`, which is implemented by the classes representing constants and variables. This interface deals with establishing the SQL query's where clause, which is mainly influenced by these two components. This is also documented by the class collaboration graph in figure 8.6. The variable class contains a reference to a list of `SQLAbles`, which are the conditions that need to appear in the where clause.

8.2.7 Performance Results

Obviously, the performance of large rulesets is a concern due to the multiple joins and large union statements which have to be evaluated when data from a view is requested. We ran three series of performance tests to analyze this problem and compare the performance of relational database to traditional inference engines.

Software and Hardware Configuration Each test compares the performance of Microsoft SQL Server 2000, IBM UDB 7.2, and XSB 2.5. The tests for both relational databases were performed in a client server setting with a Java program connecting via JDBC-ODBC middleware. XSB on the other hand was evaluated by using the Prolog `findall` predicate. This predicate collects all solutions of a goal, such as $ancestor(X, Y)$ and stores them in a list. However, this happens inside of the Prolog engine and no results need to be pushed through elaborate middleware layers. Therefore, the comparison of the database servers to XBS should only be interpreted in a qualitative way. The database server had an Intel Pentium III processor with 866MHz and 512MB RAM. The same machine was also used for the XSB test runs.

[6]The personal developer edition of DB2 version 7.2 is available at the IBM website.

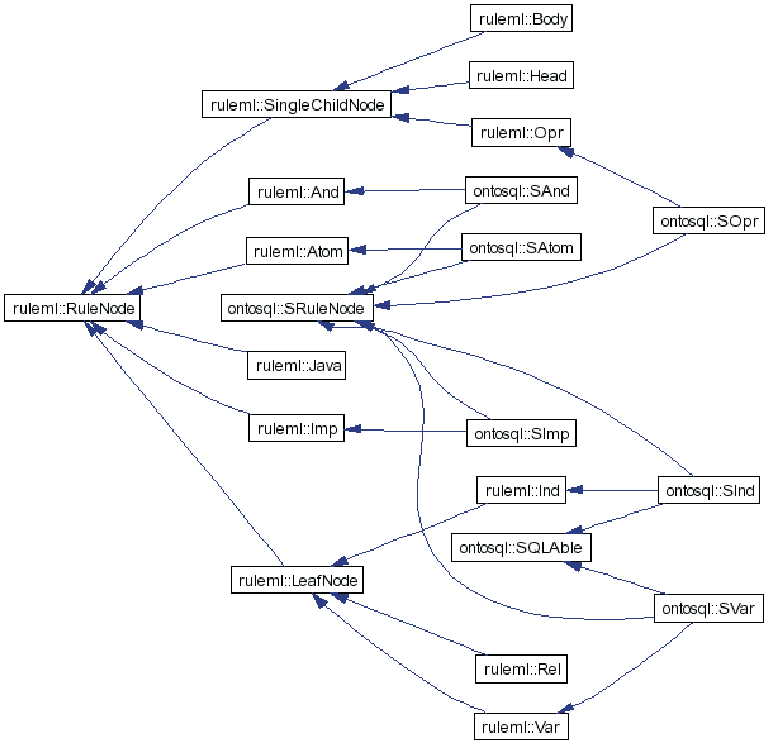

Figure 8.5: Class inheritance diagram of the OntoSQL package

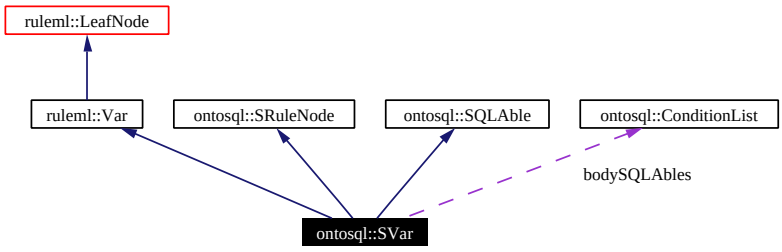

Figure 8.6: Class collaboration diagram .

Test 1 The first test is supposed to evaluate how a large number of rules, which are chained together by the choice of head and body predicates, affect the performance. We define the following rules:

```
p1(X,Y) <- p0(X,Y).
p2(X,Y) <- p1(X,Y).
...
```

Consequently, a fact table was created for each predicate along with the corresponding view definition, which calculates the union of the base facts and the additional information obtained by via the rule:

```
create view p2 as
    select * from p2fact
    union
    select * from p1
```

All fact tables were filled with 1000 tuples each having the values (1,1), (2,2), ... as subject object pairs. The experiment then performs queries on all of the views and measures the time it takes from sending the query to receiving the last result row. In turn, the measurements for XSB were taken by timing the command `findall([X,Y], f0(X,Y), L)` from the input prompt. Figure 8.7 illustrates how the database engine executes this chain of rules. We end up with a linear sequence of union operators, each being fed with the previous result and the values from a full scan of another fact table.

Figure 8.8 shows the average execution times of four test runs for querying the different predicates in the chain. From the execution plan, one can conclude that the increase with a growing number of rules would ideally be linear. The performance of DB2 and SQL Server increases slightly faster than linear with DB2 performing much worse overall. While loading the 38th view, DB2 aborted with an out of memory error. The execution time of SQL Server is only slightly slower compared to XSB, even though SQL Server was configured in the client server setting. Both observations indicate that the queries are well optimized in SQL Server. The overall response times for computing a chain of 100 rules, which causes the database to merge 100000 tuples is less than 400 milliseconds. Clearly, this is an indication that SQL Server, like XSB, holds all tables in main memory. The performance of DB2 compared to XSB is what we expected.

Test 2 The second test aims at creating rules resulting in several join operations. Therefore the rule base was changed to a Fibonacci-like pattern with a predicate depending on its two predecessors. The fact tables again contained 1000 tuples each.

```
p1(X,Y) <- p0(X,Y).
p2(X,Z) <- p1(X,Y), p0(Y,Z).
p3(X,Z) <- p2(X,Y), p1(Y,Z).
...
```

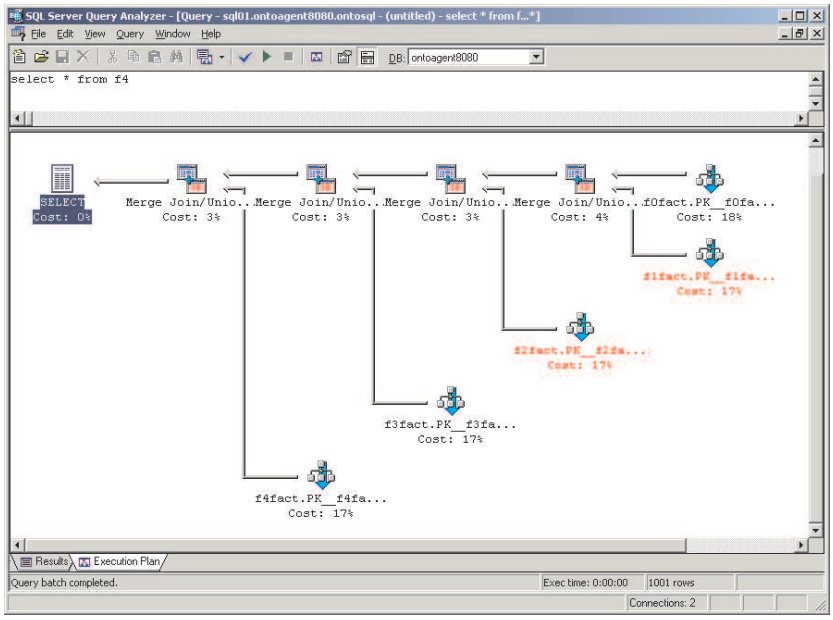

Figure 8.7: SQL Server's execution plan for a simple chain of implication rules.

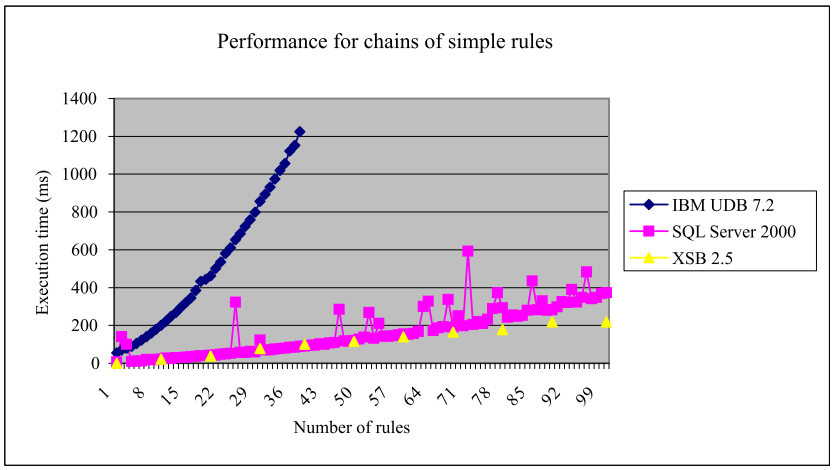

Figure 8.8: Performance of a chain of simple rules. SQL Server almost reaches the performance level of XSB.

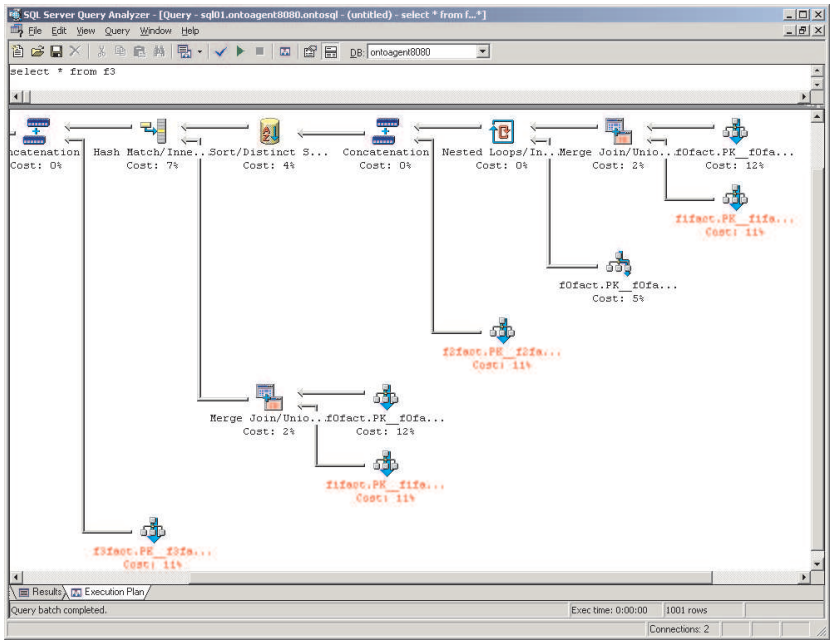

Figure 8.9: SQL Server's execution plan for a chain of complex rules.

This causes execution plans like the one shown in figure 8.9. Due to the fact that two views need to be joined to compute the next view, the operator tree looks like a binary tree. As it can be seen in the exponential character of the curves in figure 8.10, none of the three candidates comes up with the much faster strategy of iteratively materializing the predicates. This exponential growth in complexity causes DB2 and XSB to run out of memory starting from level eight. SQL Server is able to compute up to level ten before exceeding the limit of 256 tables in a join. Up to level six, which already causes massive computations to take place, the response times of less than two seconds are very satisfactory. Again, the performance of SQL Server compared to XSB is outstanding.

Test 3 The third test run was supposed to evaluate the performance of computing with recursive rules. The test bases on a single table filled with the values (1,2), (2,3), ..., (1000,1001). The underlying predicate is assumed to be transitive. Depending on how many recursion levels are computed, a maximum of 1000 + 999 + 998 + ... + 1 values should be retrieved.

This test run required different setups for each database software. We used the SQL 99 recursive query facility of DB2. Figure 8.12 shows that the execution time increases almost linearly. DB2 was actually able to compute the full recursion depth. Over time, the increase slowed down, probably due to the number of new results being added getting smaller and smaller. The full transi-

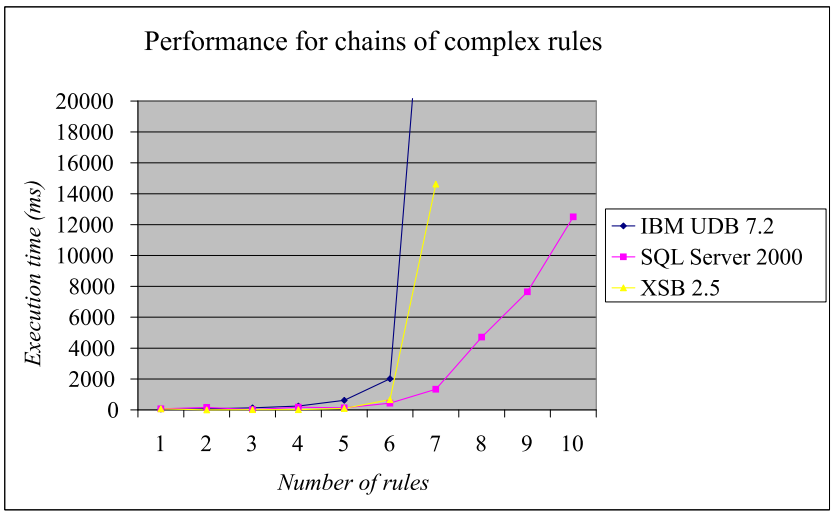

Figure 8.10: The performance of computing with an exponentially growing operator tree.

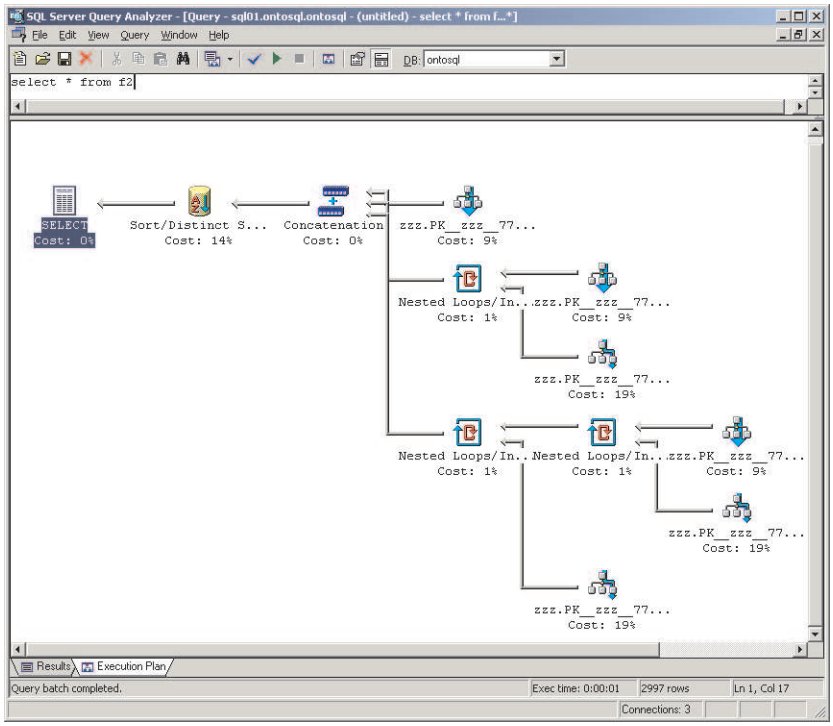

Figure 8.11: SQL Server's execution plan for recursive rules.

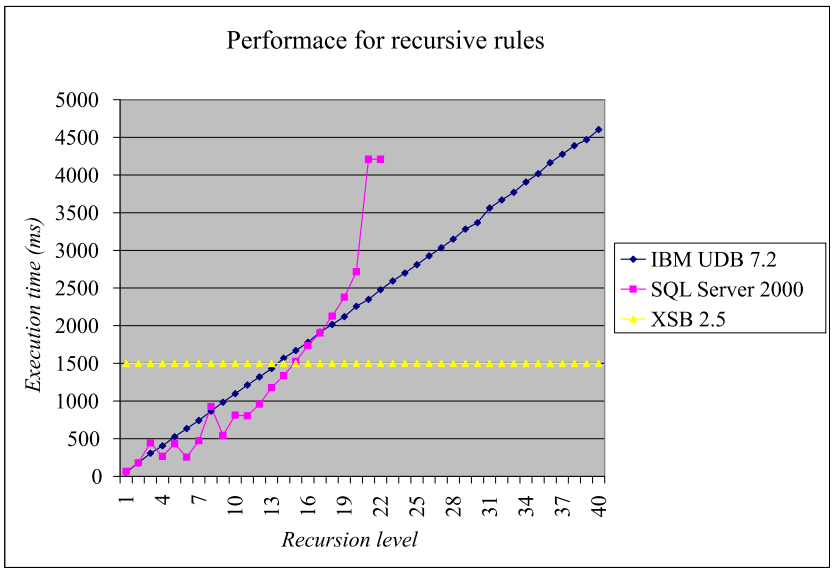

Figure 8.12: The performance of evaluating recursive rules.

tive set was computed and transferred within less than 60 seconds. Since SQL Server cannot handle recursive rules, we used the self-join workaround. This yields execution plans like the one shown in figure 8.11. The performance is comparable to DB2, however, the table join limit was again hit starting from recursion depth 21 onwards. In the case of XSB, we tried to mimic recursion depth with an addition level value being passed, similar to the level parameter used in the recursive SQL 99 query. The search is stopped once a specified maximum level is exceeded. It turns out that this is much slower than computing the full recursion to begin with. Obviously, XSB is highly optimized for recursion with its tabling algorithm. Consequently, we only computed the full recursion and the impressive time of 1.5 seconds is not dependent on the recursion depth.

Evaluation The performance measurements are encouraging due to the fast response times. Furthermore, SQL Server handles huge queries that result from the combination of different views without problems or serious performance penalty and even outperforms XSB in many cases. DB2 on the other hand, performs well for recursive queries. However, XSB exposes unparalleled performance in this area. The situation encountered in the second test run is an extreme case and all databases have problems with it. We believe that such a chain of complex rules will hardly ever appear in a normal rule base. In general, database servers definitely seem up for the challenge. Test run one showed, that dozens of simple rules can easily be chained together. However, it is definitely advisable to avoid the table self-join workaround for recursive predicates. The

IBM implementation of SQL 99 recursive queries provides both scalability and good performance results. Hence, DB2 should be preferred over SQL Server even though SQL Server performed better in tests one and two.

8.2.8 Optimization Strategies Employed

When recursive queries are evaluated using the naive Prolog-style top-down or backward chaining strategy, many facts are computed over and over again since the system returns all possible proofs for a certain derived fact. This is an unnecessary overhead, since a user is usually only interested in the data, not the proof. The bottom-up or forward chaining strategy does not have this problem but suffers from another deficiency. It will compute many facts, which are completely unnecessary for answering the query. This paragraph will briefly outline some possible optimizations, which are partly implemented in logic programming environments and database management systems.

Tabled Resolution XSB uses a mechanism called tabled resolution, where intermediate results are cached in memory [125]. The user can explicitly mark a predicate to be tabled or let the system decide. However, this decision can be tricky. Tabling improves performance dramatically for certain cases such as the ancestor predicate. However, tabling the *append* predicate, for instance, would lead to so-called over tabling, i.e. all intermediate sub lists being stored.

Magic Sets Magic Sets are another methodology, which can be found frequently. This technique, first introduced by Bancilhon et. al. [7], establishes sets of facts that might contribute to answering a query. If a fact is not in this Magic Set, it does not have to be considered. Introducing artificial predicates, which correspond to the sets, then enforces membership in the sets. The logic program is automatically transformed in such a way that the artificial predicates are included in the rule body conjunctions. Therefore, this compile time technique avoids deriving unnecessary facts, which lie outside of the Magic Sets.

Deductive databases such as LDL were the first systems to employ Magic Sets [125]. Mumick and Pirahesh report on their implementation of Magic Sets transformations in the Starburst relational database system [115]. This technique replaces part of the traditional optimization techniques in that it is used to push down all equality and non-equality predicates into the operator tree. Seshadri et. al. describe a cost-based implementation of Magic Sets in DB2 [129]. Both publications report great performance improvements for the new algorithms. However, these results refer to decision support type of queries. We were not able to determine if, and to what extent the current versions of DB2 and SQL Server used for our experiments implement Magic Sets. We believe that DB2's recursive queries use this optimization and that the other queries were optimized with the traditional methods. In general, deductive databases can definitely benefit from Magic Set optimizations. Currently, a Magic Set based rule engine is being implemented for KAON. This work was still ongoing, at the time this thesis was written.

Iterative Fixpoint Evaluation In contrast to compile time optimizations like Magic Sets rewriting, Iterative Fixpoint Evaluation is a method applied at runtime. It bases on a relatively simple observation, which can be made during the goal evaluation. If a fact f is discovered during a specific rule evaluation iteration, then it must be based on a fact which was newly found in the previous round. Otherwise, f would have been found before. The classical example is the same generation problem.

$$sg(X, Y) \leftarrow flat(X, Y)$$

$$sg(X, Y) \leftarrow up(X, A) \wedge sg(A, B) \wedge down(B, Y)$$

Ramakrishnan and Ullman describe, how Iterative Fixpoint Evaluation works in this case [123]. The first rule never changes after the first iteration, since we only find ground facts in the rule's body. In the second rule, new values of *sg* are computed as we go along. Therefore, the rule engine would work with the full relations of *up* and *down*, however, only those values of *sg* are considered, which were newly concluded in the previous iteration. This decreases the execution time dramatically, since the change in the values of *sg* is relatively small. Many deductive database projects such as LOLA [8] or Aditi [6] adopted this approach. This strategy is actually also implemented by OntoJava, since further rule evaluation depends on new facts being asserted during the process.

8.3 OntoAgent

Apart from the integration of external services, a second trend in enterprise applications can be observed. Especially in the supply chain management domain, traditional centralistic approaches fail to realize the full potential of information technology. Pioneers in this area are definitely logistics companies like UPS, which offer web-based packet tracking services for their customers. Nevertheless, the vision of parts carrying sensors and information requires a more decentralized approach, where decisions can be made locally by smart components and agents rather than simply propagating every piece of information to an overloaded server and waiting for instructions.

While this vision of agent-based enterprise systems is very appealing, the development effort is quite daunting. The complexity of existing systems, especially in the area of EAI, is already causing major headaches for project leaders. Therefore, a new design paradigm for agent technology, which focuses on ease of development, is necessary. Boley et. al. suggest to declaratively specify agents using standard Semantic Web markup languages [14]. Again, an ontology providing a shared conceptual representation is a prerequisite for this paradigm. Outside of AI and Semantic Web communities, ontologies are often viewed as purely academic and not really relevant to everyday problems. However, it is important to note that ontologies do have a large overlap with the various standardization efforts such as RosettaNet or ebXML. Both need to define a standardized vocabulary for a domain. For instance, one can leverage the

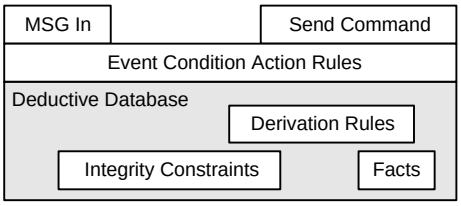

Figure 8.13: Components of a generic agent.

broad shard understanding of the RosettaNet concept of a buyer and define one's own ontology class by extending the resource http://rosettanet.org/roles/Buyer. This observation is supported by a recent report of the Gartner group, predicting that ontologies will be a core component in 75% of all EAI projects [87].

In the recent years, several definitions for the term agent have been given. Franklin and Graesser provide a nice overview of the various points of view [60]. They go on and define an autonomous agent as "... a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future". Maes provides a similar description for the term intelligent agent: "An intelligent agent is a long-lived software process that runs with little or no human supervision. It can cope with unforeseen events. It senses changes in its environment and acts upon them."

8.3.1 A Generic Agent Architecture

Boley et al.'s definition refines the statements above by identifying the five major components illustrated in figure 8.13. The following paragraphs explain each component's function as well as how it can be represented using Semantic Web mark-up languages.

Mental State Every agent has a "mental state", which is a set of facts it believes to be true. While there are a large variety of knowledge representation techniques available, simple directed labeled graphs have become the method of choice in the Semantic Web community. Objects or resources, represented by graph nodes, are interconnected via labeled relationships. The Resource Description Framework (RDF) can be used to serialize such graphs.

Schema for the Mental State Shared ontologies are thought to be the key enabling technology for agent interoperation. A class taxonomy along with properties defined for the classes are the basic components of such an ontology. Therefore, ontology representation languages like RDF Schema and DAML+OIL provide syntax for the definition of classes and properties and they can be used to model a schema of an agent's mental state. Before agents can

successfully exchange und understand each other's messages, they should agree on a common ontology to use. Otherwise it is not guaranteed that the data is interpreted according to the shared domain conceptualization.

Integrity constraints Rules play a pivotal role as they appear in the following three agent components. Integrity constraints such as `IF condition-not-fulfilled THEN error` can be viewed as logic statements used to exclude illegal mental states. The RuleML initiative is currently planning to include integrity constraints in one of the next versions. Currently, they appear for example in SQL (check, create assertion, referential integrity, etc.) or UML's Object Constraint Language (OCL).

Derivation rules The classical form of rules are derivation rules such as `IF condition THEN conclusion`. They specify the agent's terminological and heuristic knowledge and allow deriving new information from the basic set of facts known to the agent.

Reaction rules Reaction rules define the agent's behavior in response to events and messages. Reaction rules are often referred to as Event Condition Action (ECA) rules and have the following form: `UPON message RECEIVED: IF condition THEN action`. While derivation rules influence the agent's reasoning by establishing conclusions, reaction rules can trigger actions such as sending email, printing a message, or sending a message to another agent.

8.3.2 Rationale for Rule Extensions

In addition to these three rule types that are used to model these basic agent building blocks, we introduce the following two categories.

Queries In many applications it is desirable for some sort of intelligent application to query an agent's mental state. We therefore consider queries, which can be viewed as a derivation rule without rule head. Rather than deriving conclusions, a query yields the variable assignments for which the query's condition is true: `IF condition THEN yield-result`. These variable assignments are then returned to the caller.

Action Rules Finally, we consider the situation where an agent not only reacts to external events, but can also act spontaneously. Therefore, we define action rules to be a special case of reaction rules where only the condition is necessary to trigger the action part. Table 8.1 summarizes the three basic rule types and our two additional rule classes.

The main argument for our extension comes from the question of how the message flow between agents is initiated. Our standpoint is that an agent should also be able to actively examine, i.e. query, its environment. In certain situations, these queries should be activated without an external stimulus. We think

Rule Type	Description	Invocation	Example
Derivation Rules	Define derived concepts on top of base concepts	Other rules	IF condition THEN conclusion
Action Rules	Like derivation rules but can contain commands like send email, print message, or assert new fact in the rule head	After update	IF condition THEN action
Reaction Rules (ECA)	Like action rules but in addition to a condition, an external event or message is needed for the rule to fire	Incoming message	UPON message RECEIVED: IF condition THEN action
Queries	Obtain base data and derived data from derivation rules	Application	IF condition THEN yield-result
Integrity Constraints	Make sure that the agent's internal state is legal with respect to the application domain	After update	IF condition-not-fulfilled THEN error

Table 8.1: OntoAgent rule types

that agents will often try to obtain information from a legacy system via an RDF wrapper interface. The alternative standpoint is a more workflow-oriented view. Such a scenario might have sensors outside the agent that actively inform agents via the proposed event interface. Even though it makes the implementation more complicated, we choose the first approach in order to solve the message initiation problem.

8.3.3 Rule Execution

The rule classes expose important differences with respect to their execution behavior. Queries are initiated by an external component like an application or another agent. The local agent then services this request. The search condition is evaluated against the base facts and the computed facts. Therefore, derivation rules will be triggered if the search condition tests the predicates that appear in the rule heads of derivation rules. All these operations are read only due to the backward-chaining nature of our system. Action rules and integrity constraints, however, react to changes to the base facts. Integrity constraints need to be check after each update operation in order to make sure the new state is consistent. Similarly, action rules can only be activated after an update. If this were not the case, they would fire permanently. Thus, both types of rules can be activated by a one-time event of a fact being inserted. ECA rules

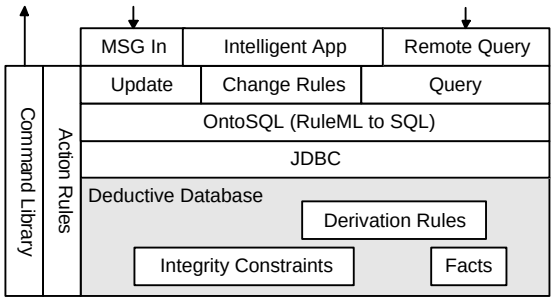

Figure 8.14: Components of an agent running on the OntoAgent platform.

also react to such an event. Rather than an update, an incoming message or event is the deciding trigger for these kinds of rules. These observations play an important role for the implementation of our OntoAgent platform [36, 38].

8.3.4 Implementation of the Agent Framework

There are several rule engines available today, many of them even supporting RuleML. SweetRules [66] and the TRIPLE system [131] are arguably is the most prominent among those. We chose to base the OntoAgent platform mainly on our OntoSQL platform. OntoSQL's main advantage is the fact that it bases on mainstream relational database technology. Executing such an agent specification requires a data store, an inference engine operating on top of it, as well as a messaging system for incoming and outgoing communication. This section will pick up the generic agent components mentioned in the previous section and describe the underlying major design decisions as well as the implementation basing on the OntoSQL [37] system.

8.3.5 Deductive Database

As illustrated in Figure 8.14, the fact base along with the derivation rules build the agent's foundation. Derivation rule engines are often referred to as deductive databases. Both the OntoJava and OntoSQL systems, introduced in the previous sections, are good candidates for the implementation. We opted for OntoSQL, since the application of database technology such as triggers seems like an interesting approach for handling the different rule types.

Derivation Rules Consequently, derivation rules are translated into SQL statements as explain earlier. No special extension of the OntoSQL framework is necessary in order to fulfill this agent rule requirement.

Integrity Constraints An important means to keep the database clean and in a correct state are the so-called integrity constraints. It is desirable to perform

as many checks directly inside the database as possible, rather than pushing this duty up to an application. The SQL check mechanism allows restricting the range of a certain attribute. Referential integrity and uniqueness constraints are usually used for clean modeling of database schemata. However, the traditional database integrity constraint mechanisms are not really applicable in our case since the schema used is very generic. It can essentially be reduced to a single subject, predicate, object table. This approach has clear advantages in dealing with semi-structured data; however, it requires more effort to be put on the definition of constraints.

The create assertion construct would solve this problem, since it allows a constraint to be defined as a condition including several tables. In contrast, the check construct only allows referring to other attributes of the same table. Unfortunately none of the major database vendors currently implements this feature.

We propose to use SQL triggers in this case. Consider the following example of a trigger ensuring that the domain of the predicate `fatherOf` is the class of all males.

```
create trigger fatherOfSignature
on fatherOfFact, typeFact
for insert, update, delete
as
    if exists (
        select * from fatherOf where subject not in
            (select subject from type where object = 'Male')
    )
        begin
            raiserror ('Fathers must be Male', 16, 1)
            rollback transaction
        end
```

Triggers basically consist of an SQL statement that is executed upon an event - typically an update. A trigger must be defined for a certain table. It seems natural to define this trigger on the `fatherOfFact` and `typeFact` tables since their views appear in the condition's SQL statement. However, other predicates such as `isParentOf` can imply fatherhood via a derivation rule. The respective fact tables, `isParentOfFact` in this case, would therefore also need to have this trigger defined on them. The trigger syntax allows using the pseudo tables deleted and inserted in the body. These tables have exactly the same structure as the base table, but only contain the deleted or the changed and inserted tuples of the table. This feature can be used to do incremental checks only on the tuples affected in the current transaction.

The question arises, whether we can use these pseudotables rather than recomputing the entire views again. While this is possible for simple cases, for example if the predicate does not appear in any derivation rule, the general case seems to get quite complex. One would have to rewrite the queries described in

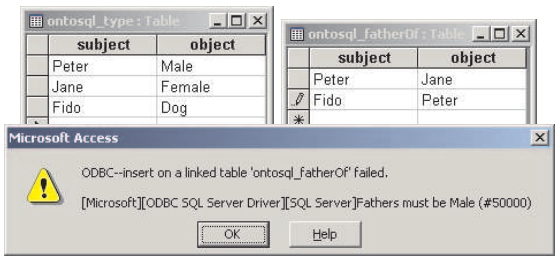

Figure 8.15: Triggers can enforce integrity constraints at the database level. The dog Fido cannot be Peter's father and the transaction is rolled back.

the previous section by replacing `[PredicateName]` with `inserted`. Furthermore, the delete case would require different logic. If the triple $(Pat, type, Male)$ is deleted from the type base facts, the integrity constraint is only violated if Pat is a father and there are no other facts, such as $(Pat, type, TallMale)$, that allow us to conclude that Pat is male.

In case of a constraint violation, the trigger causes the current transaction to be rolled back, undoing all changes that lead to the violation. Figure 8.15 shows an example of the above trigger in action. It shows Microsoft Access which is used as a graphical front-end for SQL Server. The "Fathers must be Male" error shows up since Fido the dog cannot be Peter's father.

The RuleML initiative is currently working on extending their rule language to include syntax for constraints as well.

Currently, OntoAgent provides no support for the specification of integrity constraints. This quite non-trivial task is entirely left up to the application designer. At least applying the brute-force strategy of defining the trigger for all tables can solve the problem of deciding which tables need to be associated with a certain trigger. Another complication is that constraints often require existence and forall quantification. The example above could be read as: raise an error if there exists a father who is not in the set of male persons.

8.3.6 Agent Actions

The previous section laid the agent's foundation by providing the fact store, inferencing, and the capability to exclude illegal mental states. This section will now explain how agent's can perform actions in order to interact with their environment.

Command Library We chose to implement the command library in Java. Java offers a rich array of build-in functionality such as threading capabilities and a large selection of abstract data types. Furthermore, an extensive variety of external libraries are available as Java archives for sending email or SOAP

`print(X)`	prints X to the console
`assert(triple)`	permanently asserts new fact
`delete(triple)`	permanently deletes fact
`email(X,Y)`	sends X the email with text Y
`load(URL)`	loads and asserts RDF triples
`send(X,triple-1,...,triple-n)`	sends X a message consisting of triples

Table 8.2: OntoAgent command library

RPC functionality. Table 8.2 shows the most important commands that can be triggered from rules.

The `send`, `email`, and `load` commands are executed within their own thread, since these operations can potentially take a long time and could stall the engine's execution. The load command allows the agent to interface with any RDF-enabled information source. This could be an enterprise information system as well as another agent's remote query interface.

Asserting new facts has to be used with care. Nevertheless, we believe that the assert and delete functionality is very important. Assume an agent wants to maintain a history of important events in order to use them for making further decisions. Data about the events could be stored and deleted by action rules. The decision-making rules could be designed with the temporary nature of the data in mind. For example, these rules could have a heuristic character.

Action Rules As shown in table 8.1, action rules, like integrity constraints, get evaluated upon updates to the fact base. Again, triggers seem to be the natural choice, especially since both the Oracle and IBM database servers support Java stored procedures. This would make it extremely convenient to combine Java's flexibility with the reuse of an existing trigger mechanism. However, there is a complication. Integrity constraints must hold all the time. Therefore, it does not matter if they are evaluated when it is not really necessary, i.e. due to the problems we described with the triggers' incremental strategy.

Action rules, however, must only fire, if a variable binding makes the condition true when it was not beforehand. The following example illustrates the issue: If all parents receive a congratulation email, we only want them to get the email if their child was just born. If we'd simply check the condition, every parent would get the mail after any child was born.

Due to this problem we opted for a solution where action rules are triggered from outside the database. After each update, the action rules' conditions are checked via the JDBC interface. The resulting tables containing the variable assignments are stored in a Java abstract data type, along with the command they trigger. This information is then compared to the previous state and calling the appropriate methods in the command library performs all new actions:

```
set currentState = EMPTY
forever
    upon update:
    check action rule conditions
    store results in variable newState
    for each action in newState and action not in currentState
        perform action
    currentState = newstate
next
```

8.3.7 Communication Subsystem

In our system, we distinguish between two basic types of messages: queries and information messages. Figure 8.14 shows that the message input and remote query components handle the respective incoming events and messages, whereas outgoing messages and queries originate from the command library. This section describes the structure of the messages, their effect on the agent, as well as the implementation basing on the modules previously introduced.

Queries from Remote Agents An agent sends queries in a synchronous manner,[7], in order to obtain data from another agent. Since most RDF parsers support reading data from URLs, it seemed natural to package the query inside an HTTP GET request. This simple mechanism can easily be replaced by using SOAP middleware. We are working on an implementation using the axis web service engine[8] in conjunction with the tomcat web server. The answer obtained is then an RDF/XML document. We support very simple queries retrieving all outgoing arcs from an RDF resource $(subject, ?, ?)$ or all outgoing arcs from an RDF resource with a specified label $(subject, predicate, ?)$. A query sent to the agent at `host` could look as follows:

```
http://host/servlet/Query?subject=...&predicate=...&object=?
```

The RDF result is then added to the querying agent's fact base via the update interface. Figure 8.14 shows that messages or queries from the agent are initiated from the command library which is in turn activated by the action rule component. The following action rule causes the agent to query some information host for a customer's preferences, which are then also asserted into the local database:

$$queryAndAssert(\text{"}http://infohost/servlet/\text{"}, Cust, \text{"}hasPreference\text{"}, \text{"?"}) \leftarrow isCustomerOf(Cust, Comp)$$

The implementation of the query servlet only requires reading the requested predicate view and formatting the result in RDF. Future versions of OntoAgent

[7]Synchronous meaning that the calling thread is blocked. Note that, as described in section 8.3.6, the caller specifically starts a new thread to be able to resume its operation.

[8]http://xml.apache.org/axis/

might incorporate an RDF query language or the recently published RuleML query specification, in order to allow for more flexibility in the queries.

Query Result Lifetime and Result Caching Various strategies are possible on how the results from other agents are to be treated. Currently OntoAgent asserts them until the agent knowledge base is reset. Alternatively, it would be possible to assert facts temporarily. For instance, a customer preference might be valid for a week, before it is deleted.

Along the same lines, it might make sense to cache query results. The easiest and more elegant way would be to do this in the communication subsystem. If a method to query for customer preferences is called repeatedly, only the first invocation will actually query a remote agent. Technically we can implement caching via a hashtable, which uses the parameters as keys and the results as values. Note that these features are not implemented.

Reaction (ECA) Rules Obviously reaction rules are quite similar to action rules. Consider the following example:[9]

```
ON RECEIVE requestReservation(?CarGrp, ?Period) FROM ?Customer
IF hasCapacity(?CarGrp, ?Period)
THEN SEND askIf( blacklisted(?Customer)) TO Headquarter
```

The first challenge is to correctly associate an incoming message with a certain ECA rule. The condition will again be computed via an SQL view. Therefore, the second task is to pass the incoming values into the view. One approach would be to use triggers again, which are themselves ECA rules. For a number of reasons given in [38], we opted against this method and decided to leverage our action rule functionality. We treat the predicates that appear in the message as regular RDF Schema predicates. The handler for the incoming messages *temporarily* inserts the contents of the message into the database. Note that we use the intermediate resource reservation $?R$ to represent the ternary relationship between $Customer$, $CarGrp$, and $Period$. The rule is rewritten by treating the `ON RECEIVE` part as a normal condition:

```
send(askIfBlacklisted, ?Customer, Headquarter) <-
                requestReservation(?Customer, ?R) and
                hasCarGrp(?R, ?CarGrp) and
                hasPeriod(?R, ?Period)

create trigger HandleRequestReservation
on MsgIn
for insert
as
    for all tuples in (
        select hc.sender from hasCapacity hc, inserted i
```

[9] The example is taken from http://tmitwww.tm.tue.nl/staff/gwagner/AORML/

```
          where i.msgType = 'requestReservation'
            and hc.subject = i.par1
            and hc.object = i.par2
  )
      call send(askIfBlacklisted, hc.sender, Headquarter)
      remove message
```

Compared to the action rule case, this trigger only needs to be defined to react upon inserts in the message table, since only an incoming message can trigger an action. Nevertheless, we decided against this approach for the following reasons. First and foremost, this approach requires quite a lot of additional functionality in OntoSQL. Secondly, triggers and trigger actions tend to be fairly dependent on the implementation of the database server. It would be quite hard to support the major vendors. Finally the following alternative turns out to nicely leverage our action rule functionality. We treat the predicates that appear in the message as regular RDF Schema predicates. The handler for the incoming messages temporarily inserts the contents of the message into the database. Note that we use the intermediate resource *Reservation* to represent the ternary relationship between *Customer*, *CarGrp*, and *Period*:

$$requestReservation(Customer, Reservation)$$
$$hasCarGrp(Reservation, CarGrp)$$
$$hasPeriod(Reservation, Period)$$

The rule is rewritten by treating the `ON RECEIVE` part as a normal condition. If the entire condition is met, the rule fires which mimics the desired reaction rule behavior. After the insert, which triggers the ECA rules (which actually become action rules), the message facts are deleted again:

```
receive message M(p1, p2, ..., pn)
insert parameters (p1, p2, ..., pn) into fact base
  (this can trigger certain actions of ECA rules)
remove parameters (p1, p2, ..., pn)
```

If the entire condition is met, the rule fires which mimics the desired reaction rule behavior. Figure 8.16 depicts how the incoming message is processed. First, the reservation request pseudo fact is inserted via the regular update component. After the update, the view send is queried to retrieve the parameter values of all outgoing messages. If the respective parameter constellation has not been encountered before, then the call is performed. Finally, the initial insertion is simply rolled back. Note that this is the only difference to the regular assertion of a fact via the update component.

8.3.8 Intelligent Application

Agents become intelligent agents if they are able to learn. Since machine learning techniques usually base on a wide variety of computational algorithms, it

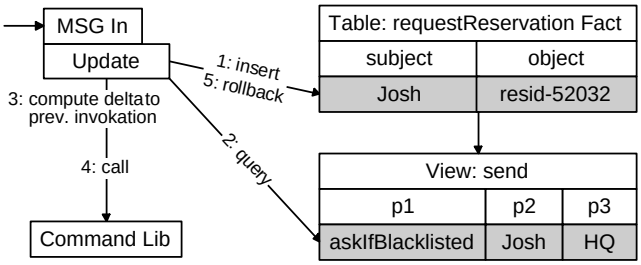

Figure 8.16: A history of actions is kept in order to determine which rule to fire.

seems awkward to try and implement rule-based learning algorithms. We propose a different setting. Figure 8.14 shows an intelligent application component that interfaces with the basic agent framework. In such as layered architecture, the learning algorithms can be implemented in any language or system using traditional techniques. The base data, however, can come from the deductive database via the query interface. The intelligent application can influence the basic framework in two ways. It can assert and delete base facts altering the inference results and ultimately the agent's behavior. More importantly, though, the rules themselves can be modified over time. Since we operate on top of a relational database, this would only require drop/create trigger/view statements that can even be performed while the agent is running.

We believe that the declarative specification is a great tool to let researchers and developers focus on the learning and behavioral aspects of agent technology located in the top layer of the architecture shown in figure 8.14. We are currently using OntoAgent in the development of a collaborative agent system for document retrieval [36]. The idea is that a community of users all shares their personal collection of links to relevant online reading material. Rules are used to determine whom to ask in a given situation. The idea of an intelligent application might even be something as simple as a feedback system, where the user, over time, tells its document retrieval agent who provided the best links.

8.4 Web Service Description Framework

When looking at the current Web Services stack, it is clear that a more detailed description of services that goes beyond the simple method signatures that can be found in WSDL specifications in needed [122]. It is necessary to capture what a service does on a conceptual level. UDDI addresses this issue to some extent by allowing to classify services and companies according to standard industry, service, and geographic taxonomies such as UNSPEC. In this paper we extend this idea by applying some of the concepts and ideas from the Semantic Web community. Standard mark-up languages like the Resource Description Framework Schema (RDF Schema) or the DARPA Agent Mark-up Language

(DAML) allow the specification of ontologies, which can be viewed as a powerful extension to taxonomies.

The idea of semantically describing Web Services is not new. Within the DAML project, the DAML-Service (DAML-S) group defined an ontology for describing complex Web Services as well as business processes. Their work provides a vocabulary for marking up services [4]. In Europe, a large research group is working on the Web Service Modeling Framework [49]. It proposes a complex mediator architecture for brokering between different data models and service invocation styles. Both projects propose good concepts and ideas, however, due to the extremely complex nature of the problem and the wide scope of both projects, the results and suggestions are currently mostly limited to how services should be marked-up. The actual service consumption is often left out. McIlraith et. al. provide an approach where user profiles can be fed into a ConGolog program which locates and executes services [112].

With the Web Service Description Framework (WSDF) we take a different approach by reducing the scope to calling function-like services without side effects and omitting the problem of business process integration with its workflow aspects [42]. Our philosophy is to provide proof of concept that is applicable today on a smaller scope.

8.4.1 Semantics of Parameters and Return Types

Consider the following example: Let's assume we need to find out the courses a student is currently enrolled in. The following service is applicable:

```
[WebMethod]
public string[] getCurrentCourses(string studentID)
```

This illustrates the first challenge. A WSDL specification only provides us with the raw data types, string and string array in this case. The fact that the parameter actually denotes a student ID that is given out by the university's registrar and that course IDs identify the courses taught at the university is completely beyond the scope of WSDL. Solely the information that both kinds of IDs happen to have the same data type string is given. Of course one could wrap the simple types in custom data types such as `StudentIDType`. Nevertheless, a client is still left guessing and needs to linguistically analyze the type's name. The WSDL message types look like this:

```
<message name="getCurrentCoursesRequest">
    <part name="studentID" type="xsd:string">
</message>
```

At this point, we use an ontology about universities to supplement the WSDL specification with conceptual information. The ontology will provide `courseID` and `studentID` properties linking the instances to the literal values. The WSDF then simply adds another type information to each message part. In the example above, `studentID` would not only be a string, but also a literal, which is connected to the ontology's student class.

8.4.2 Semantics of the Method

Let's assume the required student ID is not known. Instead, we have the user's first name, last name, and birthday. Before the `getCurrentCourses` method can be invoked, the following service must be called in order to obtain the student ID:

```
[WebMethod]
public string getStudentID(string fn, string ln, Date bd)
```

The same argument about the parameters and the return type applies here. However, we want to point out another issue. Obviously the parameters supplied must belong to the same person, as well as the student ID which is returned, is the student ID of the same person. Again, this is quite intuitive for a programmer reading the interface, but quite hard for a program to figure out. In WSDF we use rules to describe this behavior. We define ontological classes `WebService` and `WebServiceCall`. A `WebServiceCall` has parameters and results. The `WebService` class carries relevant technical information about an attached stub for instance. The following rule captures the method's semantics. Note that the use of the variable S on both sides of the rule ensures that the parameters and results belong to the same student.

$$\begin{aligned} studentID(S, I) \leftarrow \quad & hasMethod(WSC, "getStudentID") \wedge \\ & returnValueOf(I, WSC) \wedge \\ & hasParameter(WSC, FN, LN, BD) \wedge \\ & firstName(S, FN) \wedge \\ & lastName(S, LN) \wedge \\ & birthday(S, BD) \end{aligned}$$

The rule already describes how returned vales should be interpreted. In a logical sense, the new fact $studentID(S, I)$ is asserted upon I being returned. The next section will provide insight into what is actually happening in a running system in this case.

In principle, a service can be invoked once all parameters are available. The following second rule specifies, that the Web Service instance `getStudentID` can be called once the three arguments from the same student are available.

$$\begin{aligned} callable(getStudentID, FN, LN, BD) \leftarrow \quad & firstName(S, FN) \wedge \\ & lastName(S, LN) \wedge \\ & birthday(S, BD) \end{aligned}$$

8.4.3 When to Invoke a Service?

Of course not any service that theoretically could be invoked should actually be invoked. The client application can use the following two approaches. In the goal-driven approach, we use backtracking to determine a calling sequence, which will answer the goal query. In the previous example, the goal could be to

find out whether Joe is enrolled in IT200. This can be answered by calling the `getCurrentCourses` service. Having established this, the new sub goal becomes finding out a student ID. This process is repeated until a subsequent sub goal is fulfilled by the current knowledge state.

The second approach is more heuristic in nature. Here, a service is called if it looks promising. In the example, the rationale might be to try to find out as much information about the user as possible. Therefore, both services would also be called.

8.4.4 WSDF Syntax

The WSDF syntax complements existing WSDL information about a service. This implies that the Web Service can of course still be invoked in the traditional way. The WSDF file contains several links to the parameter and method descriptions. The rules are given in RuleML syntax.

```
<?xml version="1.0"?>
<wsdf xmlns="http://www.i-u.de/schools/eberhart/wsdf"
      xmlns:ruleml="http://www.dfki.de/ruleml"
      targetWSDL="http://www.mit.edu/services/CourseInfo.wsdl">
   <type part="studentID" message="getCurrentCoursesRequest"
         ontotype="http://www.mit.edu/properties#studentID" />
   ...
   <method name="getCurrentCourses">
      <semantics>
         <ruleml:imp>
            <ruleml:head>
               <ruleml:atom>
                  <ruleml:rel>
                     http://www.mit.edu/properties#studentID
                  </ruleml:rel>
                  <ruleml:var>S</ruleml:var>
                  <ruleml:var>I</ruleml:var>
               </ruleml:atom>
            </ruleml:head>
            <ruleml:head>
               <ruleml:atom>
                  <ruleml:rel>
                     http://www.i-u.de/schools/eberhart/wsdf#returnValueOf
                  </ruleml:rel>
                  <ruleml:var>S</ruleml:var>
                  <ruleml:var>I</ruleml:var>
               </ruleml:atom>
            </ruleml:head>
            ...
         </ruleml:imp>
```

```
      </semantics>
      ...
   </method>
</wsdf>
```

Note that the rules consist of predicates from both the target ontology that is used to describe the services as well as the WSDF mini-ontology on Web Services, return types, etc.

8.4.5 java2wsdf and prolog2ruleml

Like any XML dialect, WSDF and RuleML are not meant to be edited manually. Following java2wsdl's example of generating a service's WSDL description directly from the public methods, we developed the java2wsdf tool. It constructs the WSDF markup from information the programmer places in the javadoc formatted comments preceding a method, which will be exposed as a Web Service:

```
/**
 * Returns the studentID of the student with a given
 * name, firstname, and birthday
 *
 *@param fn    The student's first name
 *             wsdf:ontotype="http://www.mit.edu/types#firstname"
 ...
 *@return      The list of courses takes by the student
 *             wsdf:ontotype="http://www.mit.edu/types#studentID"
 *@rules       callable(?getStudentID, FN, LN, BD)
 *                                     <- firstName(?S, ?FN) and
 *                                        lastName(?S, ?LN) and
 *                                        birthday(?S, ? BD)
 ...
 */
public String studentID getStudentID(String fn, String ln, Date bd)
```

Since the naming conventions of java2wsdl are known, the conversion is fairly straightforward. The implementation bases on a doclet, which can read the values of the existing tags in order to extract the ontological tagging as well as the custom tags, like @rules. The rules themselves are denoted in a prolog-like syntax which allows :- or <- as the rule symbol and interprets both commas and the "and" keyword as logical conjunctions. The actual prolog2ruleml parser, which is then called from java2wsdf, was constructed with the JavaCC compiler compiler.

JavaCC allows to formally establish a grammar, in this case for prolog implications, as enrich it with Java commands which are executed upon the respective parsing events. Consider the example for parsing the rule base, which is a carriage return delimited stream containing prolog style rules:

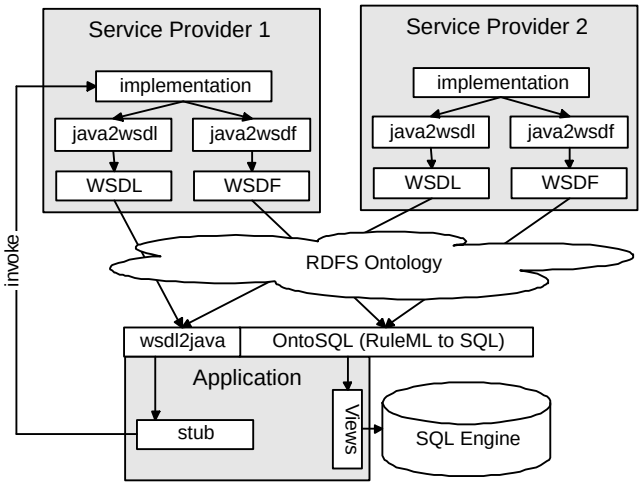

Figure 8.17: The overall WSDF system architecture.

```
String rulebase() :
{
    String rulebase = "";
    String imp = null;
}
{
    ( imp = imp() { rulebase = rulebase + imp; } )+
    { return "<rulebase>" + rulebase + "</rulebase>"; }
}
```

In essence, this defines a rule base to consist of one or more implications: rulebase := (imp)+. JavaCC allows defining the grammar rule to return a string. In our case, this is the RuleML representation of the ruleset. The RuleML representation of each individual implication is therefore returned and simply appended to the local variable `ruleset`.

8.4.6 System Architecture and Invocation Sequence

Figure 8.17 shows the overall system. At runtime, the WSDL and WSDF descriptions are downloaded from the service providers. The necessary base is provided by the fact, that both the client application and the providers share the same RDFS ontology. The OntoSQL toolkit and the WSDL compiler will then generate the SQL views and the Web Service stubs. The figure shows an example where client and server are both written in Java. Note that WSDF remains language-independent markup. Tools for other languages can easily be developed.

The actual invocation of the service works as follows. First, the application needs to determine which service to call. Using the SQL statement select * from callable, a list of Web Services where the required parameters are available is returned from the SQL view `callable`. Heuristics or a backtracking algorithm to select the next service can use the WSDF type information. The input parameters are obtained from the database and the call is made. At the same time, a new `WebServiceCall` instance and the respective parameters used are inserted into the database. This is an important step for correctly interpreting the result. Once it arrives, a tuple is inserted which associates the result with the call. This will then cause the service semantic description rule's body to be true. At this point the rule head's values can explicitly be asserted and the Web Service call's information be archived or deleted from the active deductive state.

8.5 Integrating Legacy Data

Besides integrating external services, leveraging existing data is a major point for making agent technology feasible. Assume a university's information system contains a table with student data such as student ID, first name, last name, and so on. It is easy to expose this data as RDF triples and load them into a reasoner. However, since the existing enrollment and transcript applications will stay operational, usually the data is kept in both the database and the Semantic Web worlds. This is a very undesirable situation since it creates redundancy. Inspired by the KAON suite of tools [114], we developed a very simple mechanism for this, which again bases on SQL views [43]. Rather than defining a fact table as suggested in section 8.2, we create a view which converts the existing relational data into OntoSQL facts:

```
create view hasStudentIDFact as
select "http://www.mit.edu/~"+loginName+"/", studentID from student
```

Obviously, this mapping must be done manually, since only a domain expert can identify the corresponding fields in both the database schema and the ontology. Furthermore, adjustments must often be made to webize the data. In our example, the URIs for students are given by the tilde-login name addressing for personal web pages, which needs to be computed in via a select expression. Note that in the example, the `studentID` attribute of the view is updatable which means that an update on `hasStudentIDFact` affects the original `student` table. Consequently, the strength of this simple approach is its combination with the OntoSQL framework. Together, both the Semantic Web agents and the legacy applications operate on the same data source.

8.6 OntoLang

Besides obtaining information by querying existing information sources like web services and calendar applications, the user can provide information to

the system using a very simple natural language interface. This interface bases on ideas from the MIT Start system [89] and the Template Definition Language [94]. Classes, RDF instances, and relations are tagged with corresponding lexical terms. A simple sentence such as "I am in Karlsruhe" can then be analyzed by identifying "I" as the user, "Karlsruhe" as a city, and "am in" as the `hasLocation` predicate. Since the signature of the predicate matches the types of the objects found in the sentence, the statement $hasLocation(User, Karlsruhe)$ can be asserted.

A similar language to concept mapping approach was for instance used in a document management application at Xerox corporation [48]. Within the context of the Semantic Web, this approach of enhancing ontologies with linguistic information was pioneered by an ontology focused crawling application, written on top of the KAON system [102].

8.6.1 Modeling Simple Statements in RDF

OntoLang is a small linguistic tool that can parse simple subject, predicate, object sentences. This is done based on an ontology's predicate definitions. The example shown in figure 8.18 illustrates how the predicates are attached with possible ways of their appearance in a sentence. The `enrolledIn` predicate, for instance, can be expressed by saying that someone is enrolled in a course or that someone takes a course. With the exception of `enrolledIn`, the inverse predicates' word forms are also given. Note that, this information can easily be encoded in RDF triples such as:

```
Subject:   http://www.mit.edu/onto#enrolledIn
Predicate: http://www.i-u.de/schools/eberhart/ontolang/wordForm
Object:    is enrolled in
```

Consequently, an instance of one of the classes shown in figure 8.18 also has its possible linguistic representation attached. The resource corresponding to IU's transaction processing course could have the word forms "transaction processing", "IT401", or "IT 401". Both Java the programming language and Java the coffee bean have "Java" as their word form.

Matching Given the ontology and the word forms, OntoLang is able to process a simple sentence into an RDF triple by matching the predicates and nouns onto the sentence and checking whether the types are correct. The following example shows how the associated type information helps to disambiguate similar word forms for different concepts. Consider the input sentence "the tomcat web server is written in Java". The word tomcat appears in the word forms list of the resource corresponding to the tomcat web server. In turn Java can be interpreted in the two ways mentioned above. However, only the interpretation as a programming language makes sense, since the predicate `isWrittenIn` establishes a relationship between software and programming languages.

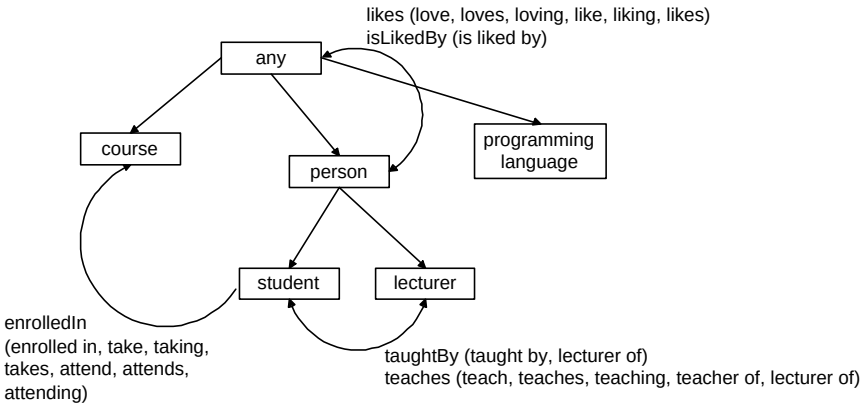

Figure 8.18: OntoLang uses a linguistic tagging of ontological relations in order to identify statements in a sentence.

Stemming We perform stemming on the noun word forms. This means that both chair and chairs are reduced to their linguistic root. Therefore, OntoLang considers them to be equal. This approach is definitely only suitable for our very rudimentary approach, since we loose potentially valuable information during this process. The predicates are not stemmed, since there typically aren't too many predicates in an ontology and since stemming can be problematic at times. Consider "teaches" and "taught". Both have the same stem, however, they denote inverse relationships. It makes a big difference if A teaches B or if A is taught by B.

Extending OntoLang OntoLang is based on a set of interfaces for predicates, nouns, etc. In the example we provide a simple sample implementation that reads the flat text files or RDF input. Customized parsers that get fed from various ontology representation formats can easily be introduced into the system via a parser factory mechanism.

8.6.2 Using OntoLang for Ontology Engineering

As section 9.2 will show, OntoLang is mainly intended as a simple front-end for driving SmartGuide, an intelligent retrieval application. However, we are thinking about possible extensions to support the complicated tasks of knowledge acquisition, both on a factual and a conceptual level. We believe that the area of text mining will be crucial for generating a substantial number of facts for a knowledge base. Consider the following example. Currently, the abundance of scientific publications reporting on various experiments, their parameters, and their results need to be read, understood, and converted to a structured format by humans. Obviously, ontologies play a crucial role for those kinds of

applications, in order to make the machines partly understand the text.

In a more interactive setting, OntoLang might be a useful tool to allow a user to permanently enter facts into the knowledge base in a casual manner. Stemming and thesauri-based features are important to give the user a certain freedom in expressing a fact naturally.

Last but not least, a natural language interface might be helpful in detecting missing ontological concepts. The language interfaces can simply collect utterances that were not parsed successfully. If a certain word appears very often, it can be presented to the ontology engineer who might then update the thesaurus or add a corresponding concept to the ontology. Along the same lines, a correction feature could be built. Assume that, before processing a user utterance, the system double checks with the user to make sure what it understood is what the user meant. This can be done by printing a representation of the formal RDF graph for instance. The system would then also collect sentences that were understood incorrectly. A frequent pattern of errors might lead to an ontological error.

Chapter 9

Building a Smart Librarian

This chapter will now describe the applications that were built on top of the base technologies presented in the previous chapter. We will first describe the data and the underlying ontology used in section 9.1. After this, the SmartGuide system [39] is introduced. SmartGuide performs a variety of inferences in order to decide which hyperlinks to recommend to the user. This section focuses especially on the integration of external information sources. SmartDialog is an extension of SmartGuide in that it searches the current state of the knowledgebase in order to identify useful questions to be answered by the user. SmartGuide applies a steepest descent search over a generality measure for documents. The SmartAPI system has a different focus. Rather than the traditional document retrieval scenario, it provides a solution for intelligently supporting programmers in coping with large documentation on various programming libraries. Nevertheless, it largely bases on the same technologies as the other two applications.

9.1 Ontologies Used

As we pointed out in the data and ontology survey in section 6.1, only very little data and few ontologies are freely available on the Web today. It would be quite awkward to try to come up with an application area that would allow reusing some of the available sources. Therefore, we developed most of the ontology and the accompanying data ourselves. Occasionally, we were able to integrate certain components of external sources.

We used Protégé to model the RDF Schema ontology, which was then uploaded into a SQL Server database. We used both Protégé and Microsoft Access 2000 as a data entry front-end. Protégé's RDF output was loaded into the database. In turn, Access was directly connected to the central data store via ODBC. The rules were written in Prolog syntax with a simple text editor. The Prolog2RuleML and OntoSQL systems were then used to upload the rule base into the database server. The focus of this work was the domain of distributed

systems. Two related domains on various Java APIs and on universities were also developed for an extension of SmartGuide and the SmartAPI application. The tables 9.1 and 9.2 show the number of classes, predicates, and rules in the ontologies as well as the size of the RDF graph data which was developed. The following sections describe the individual domain ontologies in more detail.

9.1.1 Distributed Systems Ontology

The distributed systems ontology contains information about software, protocols, programming languages, potential errors, etc. We found it to be quite easy to quickly whip up a small example containing some dozen terms. However, the task of coming up with useful and general definitions turned out to be quite hard.

In a first attempt, we defined a very fine-grained model, which for example included information about processes, memory segments, and communication paths between them. The idea was to use such data in order to predict potential causes for errors and malfunctions. We abandoned this approach since too much data and too many details would have been required for such deep analysis. Instead, we decided to follow a more heuristic approach which based on comparably shallow data and reasoning chains. This model turned out to be well suited for the SmartGuide application since a human user can easily compensate potential errors or misleading results produced by the system. The situation would be quite different, if the system was geared to function in an unattended mode, for instance to perform self-diagnosis and automatically carry out actions to solve a potential error.

In general, we found it useful to start the modeling process by writing down knowledge to be represented in textual form first. When dealing with web proxies, for instance, we would like our system to know that a web proxy mediates communication between a browser and a web server, that a browser can be configured to bypass a proxy, that certain internet connections are only available via a proxy, and so on. From these statements, we extracted the classes and properties in a first step. In a second step, the statements were transformed into implication rules. The typical implication rules state fairly obvious things about the domain. For instance, one rule can conclude that a user working with Servlets will need to use an external Java library for this. In turn, this conclusion will trigger a further rule, stating that external libraries need to be correctly included in the classpath environment. Even though these rules are quite simple, their combination allows the system to derive a series of facts. We found this to be much more useful for picking the appropriate document compared to the traditional matching of search terms and keywords. After all, these conclusions are trivial for humans but definitely not obvious for the machine.

Shallow vs. Deep Reasoning The classpath example given above also nicely illustrates our distinction between shallow and deep reasoning. It would be possible to store information on the paths of Java libraries on a computer system. With this and the knowledge that the classpath environment variable is

a semicolon-separated list of paths, one could automatically conclude whether a required library is correctly installed or not. For our application, we believe this deep reasoning approach would be too elaborate. It should be sufficient to present the user with a tip describing the problem, given that in the current situation she or he is likely to have encountered this problem.

Document Metadata In order to distinguish our system from the traditional keyword-based search, we first decided to use simple rules such as "if the user encountered a `ClassNotFound` exception, recommend this page" for selecting the appropriate documents. This approach makes sense, because rather than simply tagging the page with the `ClassNotFound` keyword, it describes the circumstance of encountering it to be the key trigger for recommending it. Nevertheless, given the fact that thousands of such rules would be needed in SmartGuide, using the rule mechanism seemed impractical. We therefore decided to implement a special mechanism for matching the documents. Rather than a list of keywords, the metadata contains a list of conditions such as "current user encountered `ClassNotFound`", which are then evaluated by the application, not the core inference engine. Different semantics can be implemented by the application. In a rule sense, the outcome would be the conjunction of these conditions. An alternative to a Boolean result is a weighted measure, which depends on the fraction of conditions, which were evaluated to true. The SmartDialog application described in section 9.3 further extends this list by a document specificity measure.

Meta Concepts A common problem, which is encountered in real-world conceptual models, is the question whether an element should be modeled as an instance or as a class. Schreiber provides the following example which arose during the development of an image retrieval system[1]. Statements had to be made about species such as apes and particular apes being depicted on certain photographs. Consequently, a class species was defined which is the set of all species. Properties of species would be their habitat or general diet information. Ape is an instance of this class but ape is also the set of all apes that have specific names, ages, weights, and so on. As mentioned in section 2.1.2, it would clearly be wrong to make ape a subclass of species, since a specific ape would then also be an instance of species. In this context, concepts such as ape, which are both class and instance, are referred to as meta-concepts. Even if the top-level class, like species, is omitted, the question on whether a large taxonomy should be modeled as classes or instances remains. The UNSPSC service taxonomy is a prominent example here. One project within the DAML effort mapped all UNSPSC classes into corresponding DAML classes[2]. This is a straightforward approach, however, it might turn out to be impractical for certain software tools. Alternatively, one can define a generic class `ServiceType` with a `parentServiceType` property.

[1] http://www.cs.man.ac.uk/~horrocks/OntoWeb/SIG/challenge-problems.pdf
[2] http://www.daml.org/ontologies/106

In our case the issue of meta-concepts surfaced with software. There are different `SoftwareTypes` such as compilers or editors having a specific purpose. The apache web server would be an instance of `SoftwareType`. In turn, all apache installations on specific computers and with specific configurations are instances of the concept apache. Since the goal of our effort is to encode general knowledge about software types, which is applicable in various scenarios, we decided to shift the focus to the meta level and talk about software types rather than installations of software. The latter would be a valid alternative for an IT infrastructure management system for example.

9.1.2 University Rules and Ontology

We extended the distributed systems ontology with general information about a university. This has two reasons. First and foremost, it provides SmartGuide with the vocabulary and knowledge necessary to tap into the university information system. Secondly, we envision a user portal that comprises not only information for learning, but is also useful for getting around on campus and finding out about events and procedures.

The major subdivision can be made into locations such as buildings, offices, lecture halls, parking structures, etc., events such as courses being taught or a campus tour being held, and persons such as lecturers, students, visitors, and couriers. These classes are linked via relations such as lecturers teaching courses, or students taking courses. We used a small sample university ontology[3] as a basis. Objects within the university such as buildings and faculty are represented by RDF resource URIs. Rules and constraints were written in RuleML using predicates defined by the ontology. Consider the following example stating that every lecture of a class that a student is enrolled in is treated as an appointment:

$$hasAppointment(S, L) \quad \leftarrow \quad enrolledIn(S, C) \wedge hasLecture(C, L)$$

The link to the distributed systems ontology is established by providing information on topics for classes and software to be used for homework assignments. Furthermore, it might be possible to establish a skill profile for students in order to better customize search results for them.

9.1.3 Java API Ontology

The Java API ontology is a natural extension of the distributed systems ontology. Rather than defining course grained concepts and their relationships, it extends these general concepts and maps them to classes, methods, and constants found in the Java language and the Java API. The generic concept of a file can be linked to the class `java.io.File` for instance. It actually turned out that examining a well-designed API can provide good input for the ontology

[3] http://kaon.semanticweb.org/ontos/test.kaon

Ontology	Classes	Predicates	Rules
Distributed Systems	27	35	38
University	19	23	11
Java API	10	12	4

Table 9.1: Sizes of the ontologies

Ontology	Nodes	Edges
Distributed Systems	417	574
University	94	123
Java API	66093	82587

Table 9.2: Sizes of the data sets

engineering process, since relationships between concepts often manifest themselves in the API. The ontology itself is not very big. It mainly talks about Java classes and their methods, methods having parameters, return values, and associated exceptions. However, the amount of data stored is quite large since most of the information can be collected and processed automatically. We used the Java reflection API for this purpose, which allows to introspect classes and objects at runtime. A small loader program examines the classes belonging to the API to be processed and uploads the relevant information into the database. The data extracted with this automatic procedure was manually augmented with links to concepts from the distributed systems ontology in order to provide a rich source for the SmartAPI application described in section 9.4.

9.2 SmartGuide

Much work has been done in the area of recommendation systems. Billsus and Pazzani [13] describe how machine learning techniques can be applied to ratings given by other users. The Deep Map project [105] covers several aspects such as natural language interfaces and the use of ontologies for smart applications. Several other papers could be cited here. Our approach differs in that it focuses heavily on software engineering issues such as sharing and reuse of domain ontologies, the integration of existing information sources, and the ease of deployment into an existing IT landscape.

9.2.1 Formal Description of SmartGuide

The previous section, which described some of the design issues of our distributed systems ontology, already gave some hints about the implementation of the SmartGuide system. We will now explain the ideas and concepts behind SmartGuide in detail. SmartGuide is a link recommendation system. As

we mentioned before, the goal is to retrieve a document containing the desired information rather than actually computing a suitable answer. This makes SmartGuide comparable to a traditional search engine which takes a query Q and usually displays the ranked result $(d_1, d_2, ..., d_n)$. We can therefore formally view a search engine as a function SE:

$$SE(Q) = (d_1, d_2, ..., d_n)$$

SmartGuide also computes a list of documents, however, the input is a set of facts $f_1, f_2, ..., f_n$ rather than a simple query. The input facts are collected from the user as well as internal and external information sources. Additional information is derived from these facts via inferencing over a ruleset R. This is formalized as a function I_R which produces a new set of facts given the original one. We can now describe SmartGuide as a function SG:

$$SG(I_R(f_1, f_2, ..., f_n)) = (d_1, d_2, ..., d_n) \tag{9.1}$$

The advantage is clearly the fact that more input data as well as domain knowledge encoded in the ruleset are used for the retrieval process.

9.2.2 User Sessions and Fact Lifetime

In order to keep the system as simple as possible, we distinguish between two kinds of facts. The first kind are base facts which are asserted by an operator or knowledge engineer. The system treats these facts as permanent and has no mechanism of deleting them. Deletions must be made from an external application if they are necessary.

The second kind of facts are temporary in nature. All statements issued by a user, collected from external agents or Web Services fall in this category. SmartGuide assumes those facts to be true for the duration of a user session. After the session is closed, the temporary facts are deleted and only the base facts remain in the knowledge base.

9.2.3 Deployment

We deployed SmartGuide in two scenarios, the first one designed to support students with their work and studies, the second one geared towards providing smart services on mobile devices, for instance when a new student arrives on campus.

Help System Since the help system scenario can rely on powerful hardware and needs to cope with large data volumes, we opted for an implementation based on OntoSQL. Figure 9.1 shows the HTML user interface. The user can state simple sentences, which are then analyzed by OntoLang. SmartGuide presents what it understood, giving the user the chance to undo and rephrase the statement. The links are simply displayed at the bottom of the page. Every

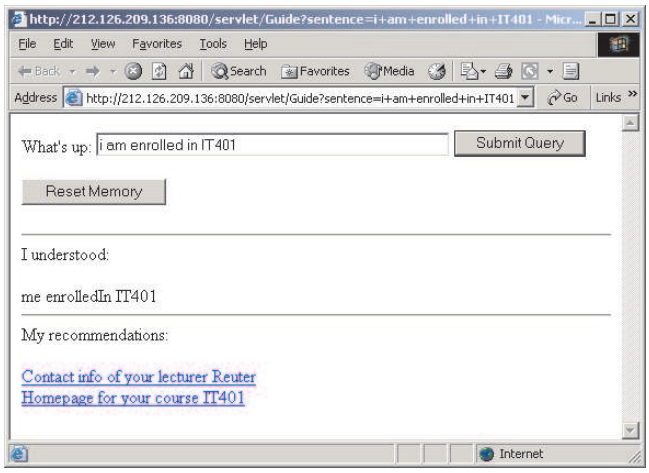

Figure 9.1: The SmartGuide interface. The user can make simple natural language statements. Based on this as well as background and domain knowledge, the system recommends links.

time another statement is made, SmartGuide reevaluates the document conditions, potentially causing some documents dropping below the required matching threshold and others to appear in the recommendation list. SmartGuide uses the quotient of true conditions over the total number of conditions as a matching score.

In the sample statement "I am enrolled in IT401" from figure 9.1, the user refers to her- or himself. This very common pattern is treated in a special way. Before using the system, users need to authenticate themselves. This way, a reference like "I" or "me" can be associated to the correct user resource in the knowledge base. The "Reset Memory" button also hints at another feature. Statements made by the user during a search session are treated as temporary facts, which are undone after the user logs off. This makes sense, since a statement about having encountered a certain exception while trying to solve an exercise is most likely not true a week from now. The implementation simply rolls back the database transaction, which remains open for the entire user session. We chose the read committed transaction isolation level in order to avoid the long running transactions locking many resources. Dirty reads are actually wanted, however, it is unlikely that they will occur, unless a user makes statements about another user or about a commonly used resource.

Mobile Guidance and Assistance Besides using SmartGuide as a help system, we envision another application area when deploying the system on a mobile device. Every time the user enters an unfamiliar environment such as an airport, SmartGuide can obtain information from external sources and combine

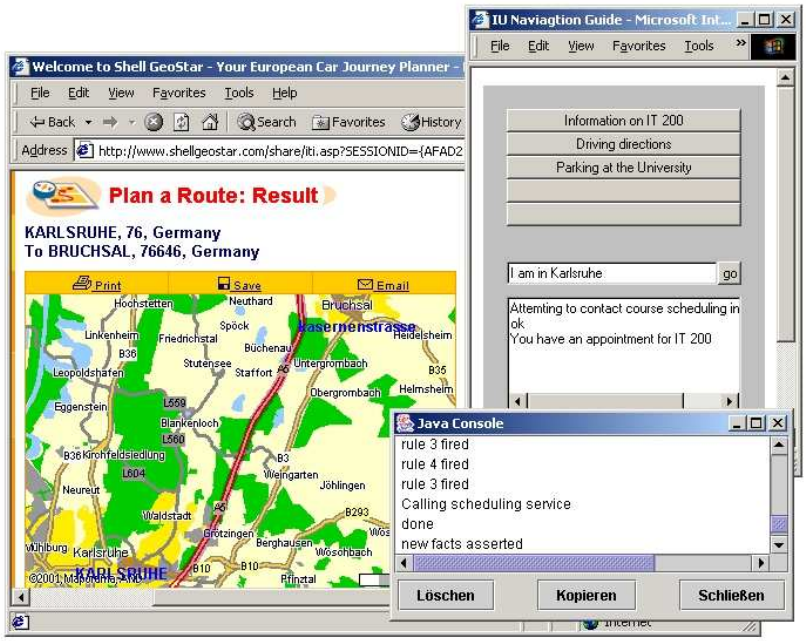

Figure 9.2: An OntoJava enabled mobile application. Rules trigger calls to the university information system and suggest web pages to the user based on collected information.

it with the local knowledge, for instance from the user's calendar. Figure 9.2 shows an example of a student getting help from SmartGuide. The system knows that the student needs to go to the International University, since a class he is enrolled in is starting soon. Given the statement that he is still in Karlsruhe, the system displays driving directions to the campus. The example also outlines a big advantage of OntoJava. The application as well as the generated engine is deployed as an applet guiding the user through the universities website and additional services.

9.2.4 Leveraging External Datasources

This section describes how external datasources are being fed into our rule systems.

From the University Information System In order to minimize the software engineering efforts, we opted against a WSDF based integration. This would have required such services to be implemented to begin with. Instead, we chose the much more direct way of integration via the view mechanism described in section 8.5. We imported information on students, courses, and enrollments

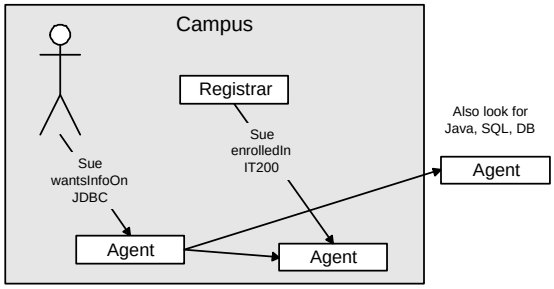

Figure 9.3: Agents using different rule bases collaborating on a document retrieval task.

this way. Since data on the topics, homework assignments, and projects of the courses are not available in structured format, we developed a mockup database for this and inserted some sample data.

The biggest advantage of this simplification is that it solves the problem of deciding when to invoke a service, how long to keep cached results, and when to drop the query results from the local fact base. If a WSDF based solution would be necessary, triggering invocations could be done similar to OntoAgent's approach of using action rules. Consider the example from figure 9.2. Given the student ID, the following action rule specifies the universities enrollment system to be queried:

$$getEnrollments(S) \leftarrow studentID(S, ID)$$

Note that the observations from section 8.3.7 about result caching and the lifetime of asserted facts also apply here. We currently do not support result caching. Query results are treated like user statement, which means they are deleted at the end of the session.

From Other Agents This section explains the rationale and the design of how various SmartGuide agents can collaborate amongst themselves. An important aspect of how humans find information can be characterized by the term search context. If I know that Jim is the database guru in my company, he could probably point me to a good tutorial on JDBC. What is important here is that Jim also knows me. Thus he knows, for example, which level of difficulty would be appropriate. This context is lost, if an external hotline is called for help. However, an external service might have a larger knowledge base to work with.

This observation can be mapped onto an agent system. Every user would have his or her personal document retrieval agent. In addition, there are larger scale agents, similar to today's search engines. The agents share a basic ontology on users, user interests, documents, keywords, and synonyms.

The agents expose intelligence in the following ways: given a search query by the user, they decide which other agents to involve in the search. Agents also know where to obtain further information on the user that might be helpful to determine the search context. Finally, agents can reason about which documents might be helpful.

Figure 9.3 illustrates a sample session. Sue works on the homework for the course IT200. In order to obtain information for solving a problem that came up, Sue provides her agent with a search string via the $wantsInfoOn$ predicate. The corresponding triple is inserted into the personal agent's database, triggering action rules to send out information messages to other agents containing this data. A friend's agent is involved in the hope that the friend encountered the same problem and found a good reference. Positive feedback from Sue's friend might have caused her agent to adapt the rule base in order to recommend the reference next time around. An external agent at JavaGuru.com is also invoked. The friend's agent queries the university's registrar to obtain information on which courses Sue is enrolled in, hoping this will help in the decision making on which documents to recommend. The JavaGuru.com agent does not query the contextual knowledge, but it has a large body of domain knowledge built-in, allowing it to compute terms related to the original query about JDBC. Further action rules cause any recommendation to be sent back to Sue's agent, which displays the result.

This sample illustrates how queries and information messages triggered by action rules bring this agent ecosystem to life. The agents work in different environments, have different rules, and work with varying degrees of domain knowledge.

9.3 SmartDialog

Just like the performance of a conventional search engine greatly depends on the right formulation of the query, SmartGuide greatly relies on a solid base of facts available about the user and his or her current context. SmartDialog addresses this problem by asking the user questions, which are likely to be helpful in finding a suitable document.

The principle behind this mechanism is quite simple. Equation 9.1 provides a formal description of the retrieval process. What is needed first is a quality measure Q for the recommendations. Our hypothesis here is that the specificity of the documents in the result set is a good heuristic. If a document's metadata is very specific and detailed, the chances that it is included in the result set are quite slim. If it is returned anyway, it is likely to be useful to the user. We currently assign a specificity value S from 1 to 10 manually when the metadata for the document is entered. In the future this can be replaced by a user feedback mechanism or by simply tracking how often the system recommends a document. The quality measure of a set of recommendations is then defined as:

$$Q(d_1, d_2, ..., d_n) = \sum_{i=1}^{n} S_{d_i} \tag{9.2}$$

After a quality metric is available, we can try optimizing the input, i.e. the facts. This is done in the following way. Assume we had a set of potential questions $Q_1, Q_2, ..., Q_n$ to ask. A sample question might be "which operating system did the problem occur on?". Each of the questions has an associated set of answers $A_{i1}, A_{i2}, ..., A_{im}$ and since every answer to the question translates into a corresponding fact, i.e. $occuredOn(problem, A_{ij})$, each question also results in a set of potential facts. Therefore, we can temporarily insert each of these facts and compute how this affects the answer quality by evaluating equation 9.1. The different values for a question are then weighed according to the answers' likelihoods, since the answer "Windows" is much more likely than "Free BSD". Finally, the weighted average quality change for this question is recorded. This process is repeated for all possible questions. The question with the highest quality, i.e. likelihood of improving the recommendation result quality, is then chosen and presented to the user. The following pseudocode summarizes shows the question selection process:

```
generate questions
for each question Q
    for each answer to Q
        F = fact corresponding to Q into fact base
        insert F
        measure quality
        delete F
    compute average weighted quality AWQ[Q]
ask question with highest AWQ
```

9.3.1 Question Generation

The remaining issue to be explained is how the questions and the corresponding answer sets are being computed. We suggest three approaches here.

The Usual Suspects For several domains it is possible to define an a priori set of questions to be asked. In a way this is similar to the questions asked by a simple expert system as the one shown in figure 4.1. SmartDialog simply goes though a hard-coded list of questions and checks whether an appropriate fact is already in the fact base. The differencs to a classical expert system is that SmartDialog is able to decide which question to ask. The order in which questions are asked is determined dynamically rather than following a fixed path. For instance, if the rule base does not contain a fact on which operating system the user works with then the associated question is being considered.

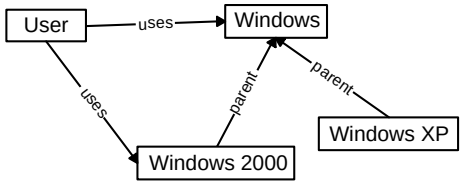

Figure 9.4: The existing fact that Windows is used is refined by the information that the user works with Windows 2000.

Taxonomic Refining of Facts Quite often, facts can be refined. Assume the user stated that she or he is working with Windows. In this case, it might be helpful to know which version of Windows is being used. We call this pattern taxonomic refining of facts. We search for facts, which consist of subjects or objects that are part of a taxonomic hierarchy. These are selected as candidates for asking questions about whether a more specific kind of resource is involved. Consider the fact: "current user uses Windows". Windows might be modeled as an instance of the software type and consequently, Windows 2000 and Windows XP have a parent child relationship to Windows. Figure 9.4 shows the corresponding fact graph. If no fact about a more specific version of Windows being used can be found in the knowledge base, then the question "What kind of Windows is used?" is asked. The actual text of the question is derived from a template associated with the predicate: "What kind of <subject> is used?". Once the answer is asserted, the more unspecific fact is obsolete. However, it is unproblematic to leave it in the fact base as an explicitly asserted fact, since the statement "user uses Windows" is true in any case, since it can be concluded from "user uses Windows 2000".

Figure 9.5 shows a sample SmartDialog session. At first some very general links turn up. Only after the question "What kind of Excpetion" is answered with "ClassNotFoundException" very specific suggestions can be recommended by the system. Note that the fixed set of answers is encoded in a drop-down list. This method is somewhat restrictive but we found it to work well in that it minimizes the chance of misunderstandings in the clarification dialog.

Extending the Fact Graph Internally, facts are stored as subject, predicate, objects triples. The rationale behind the last method is that if these triples are viewed as a graph, additional facts that can be connected to the existing facts are more likely to be helpful or applicable in the first place. Starting from the resource representing the current user, we establish the fact graph. The subject and object type information leads us to predicates having the respective type as the domain or range. Assume no information about software usage can be found in the knowledge base. The mere presence of the "uses" predicate will trigger the question "What kind of software are you using?".

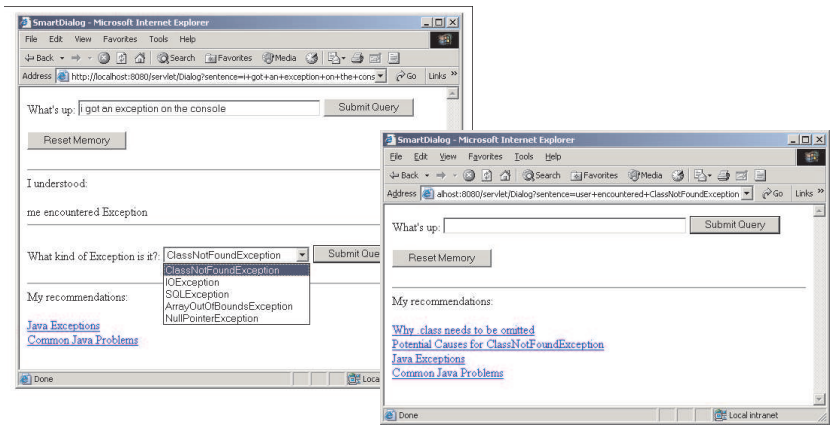

Figure 9.5: SmartDialog determines that more detail on the user's exception is most likely to be helpful.

9.4 SmartAPI

The SmartAPI application tries to address the problem of software developers having to cope with the ever-increasing number of application programming interfaces for databases, XML, file IO, etc. We observed that quite often, a certain sequence of statements needs to be coded in order to perform a single logical operation like opening a file line by line or sending a select query and reading the result. The problem is, that unless the programmer is experienced and has used the required class libraries several times before, such a seemingly trivial task requires quite some reading in the respective documentation. Consider the first sample task of reading a file line by line. Given a string that contains the path to the file, this Java code needs to be written:

```
import java.io.*;
...
    try
    {
        File f = new File(path);
        FileInputStream fis = new FileInputStream(f);
        InputStreamReader isr = new InputStreamReader(fis);
        BufferedReader br = new BufferedReader(isr);
        String line;
        while ((line = br.readLine()) != null)
        {
            ...
        }
    }
```

```
catch(IOException e)
{
    e.printStackTrace();
}
```

In order to finally be able to use the line variable for the desired purpose, this non-trivial skeleton has to be programmed. Note that there actually is a convenience class called `FileReader` that allows replacing the first three constructors. However, many programmers do not know about this since it is buried among sixty classes and interfaces in the `java.io` package.

The idea of SmartAPI is that with information on the currently available data, such as path, and the goal to be achieved, such as reading a file line by line, SmartAPI can write this skeleton automatically. This includes the exceptions to be caught, the required import statements, and the actual code.

SmartAPI consists of three parts. The search algorithm, which determines the invocation sequence, is implemented in Java using a breadth-first graph strategy. The nodes are sets of objects being available to use in the current program context as well as the list of methods called so far. The search goal is to find a state in which the desired method is called. In case of the task being "reading a file line by line", this method is `BufferedReader.readLine`, in case of reading the results of a database query, it is `ResultSet.getInt`, `getString`, etc. The second part is the loader application, which provides the raw information on methods, exceptions, datatypes, etc. that can be read directly from the Java class metadata. The third component allows to semantically tag this raw data, much like WSDF describes a Web Service as shown in section 8.4. The file constructor, for instance, takes a string parameter, which semantically represents a file system path. Similarly, `java.io.File` is linked to the ontological concept of a file. This allows SmartAPI to determine that a file has several rows, i.e. a loop is likely to be necessary.

Compared to WSDF, we omit describing methods and constructors in further detail with rules since the Java API is not purely functional in nature. Furthermore, the goal of the application is to generate a skeleton. The programmer will always have to adjust and customize the SmartAPI suggestion. However, much of the initial reading of documentation can be avoided.

Figure 9.6 visualizes the search process. The starting state is given by the user in the form of currently available objects. In this case, the only information given is a path. Note that the concept path comes from the ontology. The Java API represents paths as strings. The goal is specified as "reading a file line by line". Internally, this task is associated to calling the `readLine` method on a `BufferedReader` object. Note that unlike with WSDF, we do not have rules stating that it must be the same `BufferedReader` that is associated with the path that was originally given. Consequently, there is no guarantee that the resulting suggestion will be correct. Once the search starts, new states are reached by calling methods on the objects available, or by constructing new objects from existing ones. Many paths will lead to dead ends. SmartAPI will then choose the shortest path leading to the desired goal state.

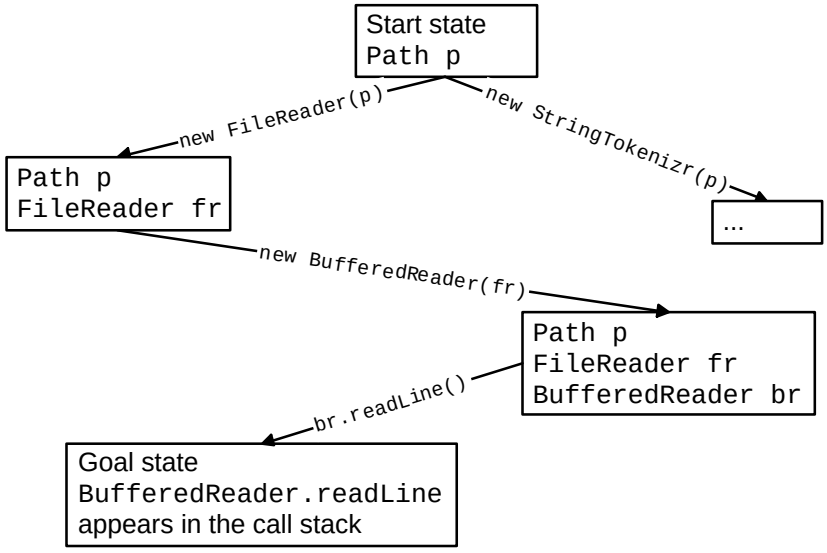

Figure 9.6: Internal search process of SmartAPI.

As with WSDF, we currently do not handle side effects. Consider writing a database application. Calling the method, which advances a result set cursor does not change anything in our representation of states.

9.5 Security

So far, only the helpful aspects of intelligent systems have been presented. However, experience shows that there will always be users trying to take advantage of technological innovations. The type of systems we implemented are also very prone to being abused, since most features only work well if sufficient context information is available. It is therefore crucial that the user is in total control over which information is disclosed to certain other agents, and which information stored in a central knowledge base is accessible to others. In the framework of this thesis, we did not work out a detailed solution for this problem and the following two paragraphs only discuss some of the basic concepts, which apply to this problem.

9.5.1 Disclosing Information to Other Agents

As we outlined in section 5.7, Tim Berners-Lee's layered Semantic Web architecture already includes the notion of a web of trust. It bases on formal proofs, which are assembled by the requesting agent and validated by the queried agent. Reconsider the example of an agent A acting on behalf of O who is authorized to

access resource R. The corresponding proof submitted by A in order to access R might look as follows:

$$hasAccess(A, R) \leftarrow actsOfBehalfOf(A, O) \wedge hasAccess(O, R)$$

This approach sounds quite logical, however, it bases on several assumptions. The agents need to identify themselves with digital signatures, use secure communication channels, and must be able to assemble and validate the proofs. More importantly, the queried agent still needs to accept the rule given above as such, and it needs to verify that the facts given, i.e. $actsOfBehalfOf(A, O)$ and $hasAccess(O, R)$, are true. The second fact will probably be found in the local knowledge base. The first fact stating that A acts on behalf of O might be stated and digitally signed by O, which raises the question whether O is trustworthy enough given that fact that O has access to R.

We are very skeptical that a formal proof mechanism will be sufficient or vastly more beneficial than simple access control lists for resources. For instance, it is impossible to encode that you should not loan out borrowed things, which essentially is the dilemma in the example above. We think that, at least for less sensitive information, a more heuristic approach will be more suitable. This belief is inspired by todays email filtering tools such as SpamAssasin[4] and websites like ShareReactor[5] that have sprung up around peer to peer file swapping services. In both application scenarios, the goal is similar, namely to identify which email is spam and which downloadable file has good quality. SpamAssasin uses a very simple point system. A limited set of conditions is checked for each email. If a condition, for example the email containing a hyperlink, is true, then the email gets a point. If after all checks, the point total exceeds a certain threshold then the email is classified as spam.

Other tools leverage the entire Internet community. Users can submit a digital footprint of spam email messages or downloaded files with the wrong or bad content to a central site. Other users can then refer to this site in order to perform the classification. Online auction sites, such as EBay, use a similar system to rate the trustworthiness of buyers and sellers. However, even this community-based approach has already been undermined. Users accumulated good ratings with several small transactions only to run away with the money of their last big sale without delivering the goods.

Looking at the various existing solutions that are based on communities and heuristics, it seems much more likely for solutions along those lines to materialize for security and trust among agents on the Semantic Web.

9.5.2 Security in Multi User Knowledge Bases

Apart from inter-agent trust and security issues, we also need to consider these points for shared knowledge bases. As our OntoSQL prototype shows, security issues within knowledge or deductive databases are somewhat similar to the

[4] http://spamassassin.org/
[5] http://www.sharereactor.com/

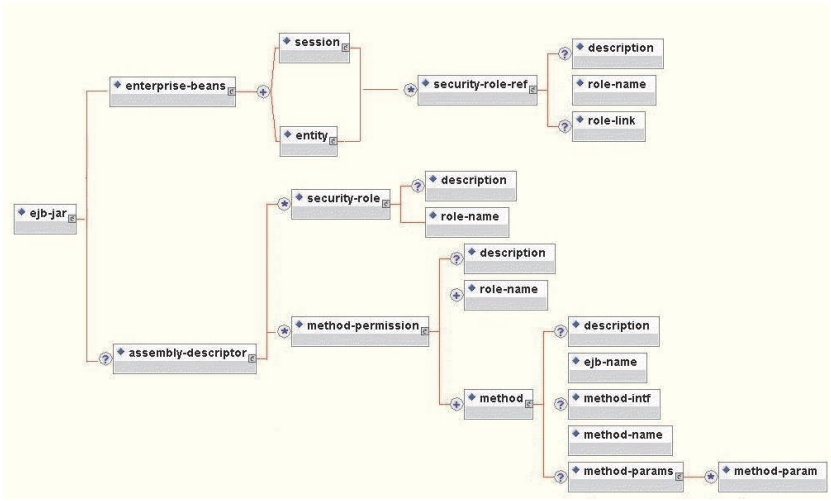

Figure 9.7: Security elements of a Enterprise Java Beans (EJB) application server.

security issues of today's database servers. Consequently, we will first look at security mechanisms of database and application servers before we outline the differences to systems like OntoSQL.

Rights and user management in a database server can be compared to an operating system's file management. Users and user groups are granted certain types of access rights such as read and write to tables and views. Views allow a more fine-grained control over a table rather than just declaring the entire table as readable or protected. A view can be defined over a table, selecting and projecting only the information which should be accessible to a user U. The actual base table can then be protected from U, only allowing partial access to selected information via the view.

Figure 9.7 shows that application servers also have the notion of users, roles, and groups, however, the resources they protect are methods or business logic. Before a method can be invoked locally or remotely, the application server checks whether the current security context matches the method metadata.

Speaking in the knowledge base terminology, database systems usually protect an entire concept such as a table storing information about employee performance. Usually, access to such security relevant information is given to a generic user, which an application uses to read the data. This application will then make sure, that an employee can only access other employees' information if they work under her or him for example[6]. These more fine-grained access rights cannot be conveniently modeled with the built-in mechanisms of databases and

[6]Also, consider a web email application such as Hotmail. It is very unlikely, that the millions of users exist in the email database. Instead, authentication and security is handled by the web application

application servers.

The main difference between traditional systems and knowledge-based systems is that more application knowledge and logic is specified declaratively using rules and ontologies rather than being coded in imperative languages like C++ or Java. Consequently, it makes sense to also specify access rights using these mechanisms. Rules appear to be the most straightforward approach here. Consider the employee performance example from above. The following rule expresses who is allowed to read employee data.

$$empInfoReadable(E, I) \leftarrow empInfo(E, I) \wedge worksFor(E, currentUser)$$

We believe this approach has two key benefits. Firstly, it is only natural to express access rights in the same manner as other domain rules. Secondly, only minimal extra effort is required for the implementation. In this case, a special function $currentUser$ needs to be introduced. Furthermore, normal users would only be allowed to access the $empInfoReadable$ predicate.

9.6 Evaluation

We believe that our approach of providing ontology-based infrastructure for various types of intelligent applications is promising. With only minor additions, the same knowledge base can be reused in the different components. OntoLang, for instance, only requires additional linguistic tagging for existing concepts. Besides the faster development of the component itself, the various solutions can be integrated easily since they base on the same semantics. Another big plus is the fact that standard markup languages allow relying on third-party tools during development.

Unlike we originally planned, large parts of the storage, inference, text processing, and communication components had be coded by the author. Individual solutions and tools for the various tasks we had to accomplish definitely exist. However, their integration seemed to be an almost impossible task. Furthermore, the search for data and ontologies we could build on was disappointing. This caused us not having enough time to develop a larger prototype. Consequently, our experiments only provide a proof of concept. No conclusion about how technologies like SmartDialog will scale up in a large deployment can and should be made.

Chapter 10

Further Work

This section will outline possible extensions to our work as well as future research directions building on the various technologies provided in this thesis.

10.1 OntoJava and OntoSQL

With respect to our inference engines OntoJava and OntoSQL, we illustrated certain features such as multiple inheritance and constraints, which have not yet been implemented. An interesting issue would be to exploit object relational features such as SQL 99 types and inheritance and replacing the simple fact tables. This would allow more efficient data access and checking of RDF Schema types within the database engine. The next step from there would be a closer integration of OntoJava and OntoSQL. We are also investigating a feature of the JESS rule engine that allows us to reason with generic Java objects. Applying this mechanism to Enterprise Java Beans would open new opportunities of embedding rule engines into custom application servers. Security and portability aspects of Java reaction rules are other important topics. Finally, an important line of research would be to incorporate more powerful rule variants as well as description logic languages such as OWL. Together with Benjamin Grosof, we started working on a mapping of an OWL subset to rules as well as on integrating general UML and ER models into such a framework.

10.2 OntoAgent

Important aspects for future research on OntoAgent are security and trust amongst the agents. Currently, a malicious agent can query the entire deductive databases of all other agents. To solve this issue, we are thinking about dropping the remote query mechanism and requiring an agent to send a regular message instead. This would allow the other agent to decide what to reply. We are also thinking about certain fact metadata fields such as who asserted a fact, where it came from, when it was asserted, and when it will expire.

Last but not least we need to gain more experience on performance issues as well as the handling of different approaches, for example regarding integrity constraints. It will make sense to revisit the Java vs. SQL design choices for implementing reaction rules, once the database vendors support more functionality.

We believe that with respect to organizing a collection of software agents, potentially via rules, much can be learned from the areas of self-organizing networks and agent behavior patterns [28].

10.3 WSDF and SmartAPI

We described our initial efforts on the Web Service Description Framework. We currently only cover the simplest case of calling methods that do not have side effects. The obvious next step is to try to extend WSDF to more complicated services. We need to explore if for those cases it is sufficient to describe the services using rules.

We think a promising line of research would be to try to extend the framework from logic-based clients to clients written in imperative languages. Assuming that core data structures are semantically tagged, it should be possible to automatically generate code for a bridge that mediates between the different representations before and after invoking the service.

Furthermore, the relationship of our approach to workflow and process description languages like XLANG or WSFL and standardization efforts such as RosettaNet or ebXML is of interest. Finally, the syntax needs to be revisited. Since the languages used for WSDF, namely RuleML, WSDL, and RDFS, have very different notation styles, the current syntax can definitely be improved with respect to its clarity.

The SmartAPI system is closely related to WSDF, even though our application areas for these technologies are quite different. Consequently, SmartAPI will also benefit greatly from modeling method calls with side effects. The system needs to be evaluated further. A good benchmark would be to see the performance compared to a simple collection of frequently used code fragments. On the user interface side, we are thinking about developing a plug-in for the popular Eclipse editor. It is also possible to minimize the user input by scanning the variables available from the chosen context within the source code.

10.4 SmartGuide and SmartDialog

The most important next step with respect to our intelligent sample applications was already mentioned in section 9.6. All systems need to be tested with larger ontologies and more data. This is also the base for a quantitative performance assessment, which compares the usefulness of our ideas compared to expert systems, question answering systems and search engines.

An interesting point from the engineering side would be to provide the user

with explanations on why a certain document is recommended. It might be useful for the user to have such additional information instead of purely relying on the SmartGuide ranking and the document title.

10.5 Probabilistic Reasoning

A very fundamental concern, which needs to be addressed in the future, is the feasibility of cleanly including probabilities into the reasoning process. Our SmartGuide and SmartDialog systems implement some heuristics basing on probabilities and somewhat fuzzy metrics of specificity. These algorithms were placed on top of a traditional Boolean inference mechanism. We looked into approaches such as Fuzzy Logic and Bayesian Networks. However, we believe probabilistic relational models to be the most promising candidate [63, 64, 92].

Chapter 11

Summary

This thesis described how ontologies and the Semantic Web in particular can provide an infrastructure for developing and integrating intelligent applications. We will now summarize our individual results and contributions and provide an outlook on the future of the Semantic Web.

11.1 Results and Contributions

Seven key questions were raised in the introduction. The following text picks those up again and summarizes if and how they have been answered.

1. How much and what kind of semantically tagged data is available on the Web today?

   We crawled the Web for RDF data. Four different search strategies, namely scanning homepages listed in DMOZ, crawling in the proximity of Semantic Web portal sites, retrieving URLs containing the string RDF, and searching the URIs from previously found RDF facts, were used. The RDF facts were analyzed according to their predicate namespaces and their location. A promising development is the Adobe RDF metadata format, which is produced by Adobe tools. Researchers are currently working on embedding RDF support into the popular Plone content management system. We believe that such easy to use tools that expose existing metadata in RDF format will be the crystallization point for the Semantic Web.

2. Can rules and ontology be integrated into a mainstream IT infrastructure of databases and object oriented programming languages?

   We presented our OntoJava and OntoSQL inference engines which leverage proven technologies such as Java and relational databases as core components to build on. Despite some of the inherent limitations, we strongly believe that the advantages of this approach by far outweigh the

limitations. There is an abundance of Java tools like IDEs and debuggers available. The javadoc tool, for instance, can automatically create a browsable documentation of the ontology. Java's reflection interface also enables us to easily inspect the ontology from an application or browse the state of the object database. OntoJava allows for easy extension and integration of the ontology into the existing IT landscape. Finally, it is possible to customize the inference mechanism if, for example, probabilistic reasoning is to be introduced. OntoSQL addresses the forward chaining limitation of the Java approach. It is also not limited to main memory storage and provides persistence, transactional safety, security and backup management, as well as many middleware and data entry solutions such as ODBC and Access. The fact that virtually every enterprise builds its IT infrastructure on relational databases is probably the most convincing argument for trying to introduce Semantic Web technology this way.

3. Can ontologies and event condition action rules be used to define the behavioral aspects of agents?

   OntoAgent builds on Boley et. al's [14] idea of specifying an agent entirely using the Semantic Web mark-up languages RDF, RDF Schema, and RuleML. We described our implementation of OntoAgent, which is based on the OntoSQL tool that allows us to use a relational database as an inference engine. With a set of add-ons, written in Java, we were able to augment the system with the necessary components, i.e. reaction rules, a command library, and a messaging subsystem based on HTTP. Several design choices and trade-offs were discussed. In several cases, we opted against the pure SQL variant with assertions and triggers due to varying database implementations, unimplemented SQL99 features, and engineering simplicity. We believe that this is an extremely promising approach since it relieves programmers from many of the burdens that are usually inherent with the implementation of agent systems. It also makes it easier to integrate agents written by different teams. We identified the key points of agreeing on a common RDF Schema and of proper action and reaction rules for enabling collaboration.

4. Is it possible to invoke a semantically tagged service on the fly?

   We presented WSDF, a semantic annotation of Web Services based on the existing languages WSDL, RDFS, and RuleML. Using a WSDL compiler and our OntoSQL RuleML to SQL converter, a client application is able to process the generic descriptions into running code. Web Services can therefore be invoked simply by agreeing on an RDFS ontology a priori. Even though we restrict ourselves to the simplest case of methods without side effects, this is a major step forward, since traditionally programmers must either be aware of a certain UDDI tModel or simply read a textual description and program corresponding client code. Compared to other approaches like WSMF or DAML-S, we provide a complete solution that

not only specifies service mark-up, but for the first time also specifies how the client has to interpret the results.

5. Is it possible to base linguistic features on the same mechanisms that are used for obtaining data from information sources other than the user?

   OntoLang uses linguistic annotations to parse and understand simple natural language statements, which are fed into the RDF fact base. This way, OntoLang is seamlessly integrated with the inference engine and other applications that also access the inference component. Since annotations required by OntoLang are also given is RDF format, no extra parser is required. Only the algorithm matching a sentence onto the ontology was implemented in Java. SmartDialog leverages the knowledge base to predict how answers to potential questions affect the pool of possible documents to be suggested. The situation is similar to OntoLang since the fact storage could be reused but the logic had to be written by hand rather than using rules.

6. Does an underlying ontology provide the necessary foundation for various system components to interoperate seamlessly?

   Assuming that the ontology is represented in both the facts' data model and the query and inference layer, all applications can be interfaced via this foundation. The mediator is the shared knowledge and data model. Note that just as in the database world, updates to the knowledge base made by one component are immediately visible to all other components. This way it was very easy to integrate OntoLang with WSDF, simply because both base on ontologies marked up in Semantic Web languages.

7. Are ontologies a useful help for developing a complex application?

   By developing an entire suite of sample applications, we provided a proof of concept that an ontology-based infrastructure is useful for developing intelligent applications. SmartGuide and SmartDialog demonstrate how a complicated document retrieval application, which takes the user's current context into account and asks clarification questions, can be built with relatively little engineering effort. SmartAPI shows that a vastly different application can also base on the same core components.

Our thesis is that data integration and software engineering can benefit from establishing ontologies as the basis for a variety of intelligent applications. We believe that the thesis is correct and that our approach is a viable solution for integrating existing data, knowledge, and tools into combined and therefore much more powerful systems. We provide tools for using ontologies and rules as the foundation of intelligent software. The sample implementation of an online help system proved that the engineering effort could be reduced significantly. The prototype is also able to leverage existing legacy databases, other agents, as well as Web Services are information sources demonstrating the data integration aspect. The weakness of our work is that the concepts need to be validated in larger deployments.

11.2 Outlook on the Semantic Web Initiative

The Semantic Web initiative offers exciting possibilities. The standardization of various mark-up languages for semantically well-defined statements, ontologies, and rules allows users all over the world to easily publish and share machine-readable data and knowledge. The range of potential applications is only limited by one's imagination. Integrated travel portals, intelligent support for online learning, or smart enterprise application integration solutions are some examples.

As it can already be seen in the XML world, the standardization of mark-up languages has another key advantage besides sharing and reuse of data. Being able to build on a large pool of parsers, editors, converters, and other tools supporting these standardized languages is another huge advantage.

Despite the advantages we mentioned, we observe a chicken and egg situation, which currently prevents the fast adoption of these new technologies. Data and knowledge is available, however, they are not being published in Semantic Web formats since there currently are no Semantic Web applications consuming the data. In turn, writing applications only makes sense if data is available. We have two recommendations for overcoming this problem. First and foremost, we believe that the development of tools and systems should favor issues of scalability, user friendliness, and interfaces to existing data repositories, over advanced features such as the Web of trust or the integration of highly expressive logic variants. We believe that our work, especially with OntoSQL and WSDF, goes in the right direction here. Secondly, more research projects should aim at developing Semantic Web applications. Such implementations will get a larger community interested and will provide valuable experience and feedback for adjustments to current and for the specification of future standards.

Despite the relatively slow rate of adoption so far, there are many promising developments, such as Adobe's embracing of RDF. Also, more and more enterprise application integration projects are starting to use ontologies. Obviously, EAI projects have a more immediate and, in terms of monetary savings, measurable impact compared to projects from domains such as knowledge management that traditionally already employ ontologies. Successful ontology-based EAI projects are likely to promote the Semantic Web.

The areas of Web Services and Semantic Web share the dilemma of lacking a clear business model. Today's browser-based Web is mostly financed by banner advertisements. However, this model does not work once machines access Internet resources, regardless of whether it is a Web Service client or a Semantic Web agent. Microsoft is currently experiencing a heavy setback with its subscription-based myServices strategy. In our opinion, these problems will be solved as soon as a solid payment infrastructure is established. Once people come up with a killer application, it will be easy to establish a good and successful business model for it.

Acknowledgements

First and foremost, I want to thank Prof. Dr. Andreas Reuter for his valuable feedback and comments as well as for giving me the time required to do this research and the opportunity to meet colleagues during my visits to Berkeley, Stanford, and many other workshops and conferences. I also thank Prof. Dr. Wolfgang Wahlster for his suggestions and for accepting me as a PhD candidate at the University of Saarbruecken. Dr. Harold Boley, Said Tabet, Prof. Dr. Gerd Wagner, and Prof. Dr. Benjamin Grosof, the initiators of the RuleML initiative, always encouraged me in my line of research. Coming from a small institution, my participation in the RuleML effort was particularly important for me since it gave me the chance to collaborate with many other researcher that have the same interests. I also thank my current and former colleagues at the International University and the European Media Lab, in particular Prof. Dr. Stefan Fischer, Prof. Dr. Ian Cloete, Dr. Isabel Rojas, Dr. Wolfgang Becker, Horst Hellbrueck, and Ulrich Walther for many interesting and fruitful discussions and for making our work environment so pleasant. I also got many ideas from great discussions with Bill Andersen from OntologyWorks Inc. Last but not least, I thank my family, especially my wife Karin and my mother Hildegard, for encouraging and helping me during my studies and of course for proof reading this document.

Appendix A

List of Acromyns Used

ACL	Agent Communication Language
ADL	Advanced Distributed Learning
ARIADNE	Alliance of Remote Instructional Authoring and Distribution Networks for Europe
API	Application Programming Interface
ARPA	Advanced Research Projects Agency
ASCII	American Standard Code for Information Interchange
CBT	Computer Based Training
CIA	Central Intelligence Agency
CORBA	Common Object Request Broker Architecture
DAML	DARPA Agent Markup Language
DARPA	Defense Advanced Research Projects Agency
DBMS	Database Management System
DCOM	Distributed Component Object Model
DOM	Document Object Model
DTD	Document Type Definition
EAI	Enterprise Application Integration
ECA	Event Condition Action
EDI	Electronic Data Interchange
EDIFACT	United Nations Directories for Electronic Data Interchange for Administration, Commerce and Transport
EJB	Enterprise Java Beans
EOF	End Of File
FAQ	Frequently Asked Question
FCA	Formal Concept Analysis
FIPA	Foundation for Intelligent Physical Agents
GPS	Global Positioning System
HTML	Hypertext Mark-up Language
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IBM	International Business Machines Inc.

IDE	Integrated Development Environment
IMS	Instructional Management Systems
IST	Information Society Technologies
ISO	International Organization for Standardization
JDBC	Java Database Connectivity
JESS	Java Expert System Shell
JSR	Java Specification Request
KAON	Karlsruhe Ontology Tool Suite
KIF	Knowledge Interchange Format
KQML	Knowledge Query Mark-up Language
LMS	Learning Management System
LOM	Learning Object Metadata
LTSC	Learning Technology Standards Committee
MIT	Massachusetts Institute of Technology
NAICS	North American Industry Classification System
NEC	Nippon Electric Company
NYSE	New York Stock Exchange
OCL	Object Constraint Language
ODBC	Open Database Connectivity
OKBC	Open Knowledge Base Connectivity
OIL	Ontology Inference Layer / Ontology Interchange Language
OWL	Web Ontology Language / Ontology Works Language
PDF	Portable Document Format
PURL	Persistent Uniform Resource Locator
QEL	Query Exchange Language
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
RDQL	Resource Description Query Language
RPC	Remote Procedure Call
RPM	RPM Package Manager
RQL	RDF Query Language
RSS	Rich Site Summary
SAX	Simple API for XML
SCORM	Sharable Content Object Reference Model
SOAP	Simple Object Access Protocol
SOEP	Simple Ontology and Metadata Editor Plugin
SQL	Structured Query Language
START	SynTactic Analysis using Reversible Transformations
SWIFT	Society for Worldwide Inter bank Financial Telecommunications
TREC	Text Retrieval Conferences
UDDI	Universal Description, Discovery and Integration
UML	Unified Modeling Language
UNSPSC	Universal Standard Products and Services Codes
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name

WSDF	Web Service Description Framework
WSEL	Web Service Endpoint Language
WSFL	Web Service Flow Language
WSMF	Web Service Modeling Framework
WWW	World Wide Web
XHTML	Extensible Hypertext Mark-up Language
XLANG	XML Language
XML	Extensible Mark-up Language
XMP	Extensible Metadata Platform
XSL	Extensible Stylesheet Language
XSLT	XSL Transformations

Appendix B

Guide to the Software Download Pages

This appendix is to provide a quick overview of the software downloads that are available from the author's website at http://www.i-u.de/schools/eberhart/. Note that all information is also available on the accompanying CD. Nevertheless we invite you to visit the website for most up to date information and patches. The individual project pages can usually be found by appending the project's name to the author homepage's URL.

B.1 OntoJava

The OntoJava system is available from http://www.i-u.de/schools/eberhart/ontojava/. This site contains a quick introduction, a feature list, a list of limitations, and information on bugs and known issues. The download contains the sources as well as a set of Windows batch files for running the examples.

B.2 OntoSQL

The OntoSQL project page can be found at http://www.i-u.de/schools/eberhart/ontosql/. The download contains sources, binaries, and examples. The examples were tested on Microsoft SQL Server 2000, but should run on any other major database server. Currently, the OntoSQL system generates a text file containing the required table and view creation statements, which can be run against an ODBC data source using the upload tool provided.

B.3 OntoAgent

The OntoAgent system is available under http://www.i-u.de/schools/eberhart/ontoagent/. The download includes a small example, which uses two instances of a Java-enabled web server with two attached Microsoft Access databases to perform the agent communication and reasoning.

B.4 Prolog2RuleML

The prolog2ruleml converter tool is available at http://www.i-u.de/schools/eberhart/prolog2ruleml/. The site offers the application, the underlying JavaCC grammar as well as an online version of the system at http://212.126.209.136:8080/prolog2ruleml.html.

B.5 OntoLang

The OntoLang system's project page is located at http://www.i-u.de/schools/eberhart/ontolang/. The system and a small command-line based example are included in the download. A version of OntoLang is included in the online demo of SmartGuide. See the following section for this.

B.6 SmartGuide

The site http://www.i-u.de/schools/eberhart/smartguide/ offers a small introduction to the SmartGuide system. A demonstration application, which bases on the university domain, is available as well. Note that OntoLang is included in this application allowing the user to make simple statements such as "I am enrolled in IT401".

B.7 RDF Crawler

The page of our RDF Crawler http://www.i-u.de/schools/eberhart/rdf/ offers the source code, binaries, and software documentation. In addition, copies of all RDF files found during the search, as well as lists of all URL scanned are available in RDF and text and tab formats as well as a Microsoft Access database.

Bibliography

[1] A. Abecker, A. Bernardi, and M. Sintek. Proactive knowledge delivery for enterprise knowledge management. In G. Ruhe and F. Bomarius, editors, *SEKE Learning Software Organizations - Methodology and Applications*, pages 103–117. Springer-Verlag, 1999.

[2] E. Adar, D. Karger, and L. Stein. Haystack: Per-user information environments. In *Proceedings of the Eighth International Conference on Information Knowledge Management*, pages 413–422, Kansas City, MO, USA, 1999.

[3] Adobe Inc. A managers introduction to Adobe eXtensible Metadata Platform, the Adobe XML metadata framework. http://www.adobe.com/products/xmp/pdfs/whitepaper.pdf.

[4] A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara. DAML-S: Web Service description for the Semantic Web. In I. Horrocks and J. Hendler, editors, *Proceedings of the First International Semantic Web Conference (ISWC 2002)*, pages 348–363, Chia, Sardinia, Italy, June 2002. Springer.

[5] G. Antoniou. Nonmonotonic rule systems using ontologies. In M. Schroeder and G. Wagner, editors, *Proceedings of the international Workshop on Rule Markup Languages for Business Rules on the Semantic Web. In conjunction with the first International Semantic Web Conference (ISWC 2002)*, pages 128–139, Chia, Sardinia, Italy, July 2002.

[6] I. Balbin and K. Ramamohanarao. A generalization of the differential approach to recursive query evaluation. *Journal of Logic Programming*, 4(3):259–262, 1987.

[7] F. Bancilhon, D. Maier, Y. Sagiv, and J. Ullman. Magic sets and other strange ways to implement logic programs (extended abstract). In *Proceedings of the fifth ACM SIGACT-SIGMOD symposium on Principles of database systems*, pages 1–15, Cambridge, MA, USA, 1986. ACM Press.

[8] R. Bayer. Query evaluation and recursion in deductive database systems. Technical Report TUM-I8503, Technische Universitaet Muenchen, Munich, Germany, 1985.

[9] M. Bergman. The Deep Web: Surfacing hidden value. *In J. Turner and E. Trager, editors. Journal of Electronic Publishing. Ann Arbor, MI, USA*, 7(1), 2001.

[10] T. Berners-Lee and J. Hendler. Publishing on the Semantic Web. *Nature*, pages 1023 – 1024, April 2001.

[11] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, pages 28–37, May 2001.

[12] T. Berners-Lee, D. Karger, L. Stein, R. Swick, and D. Weitzner. Semantic Web development. http://www.w3.org/2000/01/sw/DevelopmentProposal, 2000.

[13] D. Billsus and M. Pazzani. Learning collaborative information filters. In *Proceedings 15th International Conf. on Machine Learning*, pages 46–54, Madison, WI, USA, 1998. Morgan Kaufmann.

[14] H. Boley, S. Tabet, and G. Wagner. Design rationale of RuleML: A markup language for Semantic Web rules. In *Semantic Web Working Symposium*, Stanford University, CA, USA, July 2001. http://www.semanticweb.org/SWWS/program/full/paper20.pdf.

[15] K. Bollacker, S. Lawrence, and C. Giles. CiteSeer: An autonomous Web agent for automatic retrieval and identification of interesting publications. In K. Sycara and M. Wooldridge, editors, *Proceedings of the Second International Conference on Autonomous Agents*, pages 116–123, New York, NY, USA, 1998. ACM Press.

[16] C. Bornhoevd. Semantic metadata for the integration of Web-based data for electronic commerce. In *Proceedings of the International Workshop on Advance Issues of E-Commerce and Web-based Information Systems*, page 137, Santa Clara, USA, April 1999.

[17] A. Bouguettaya, B. Benatallah, L. Hendra, M. Ouzzani, and J. Beard. Supporting dynamic interactions among Web-based information sources. *IEEE Transactions on Knowledge and Data Engineering*, 12(5), October 2000.

[18] P. Brezillon. Context in human-machine problem solving: A survey. Technical Report 96/29, Laforia, Universite Paris VI, Paris, France, 1996.

[19] R. Burke, K. Hammond, V. Kulyukin, S. Lytinen, N. Tomuro, and S. Schoenberg. Question answering from frequently asked question files: Experiences with the FAQ finder system. Technical Report TR-97-05, InfoLab, University of Chicago, 20, 1997.

[20] A. Campbell and S. Shapiro. Ontological mediation: An overview. In *IJCAI-95 Workshop on Basic Ontological Issues for Knowledge Sharing*, Montreal, QC, Canada, 1995.

[21] A. Canas, R. Hewett, M. Carvalho, and M. Carnot. Knowledge models for organizing and searching information. In *Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR 2001)*, August 2001. CD ROM Paper 129.

[22] L. Carr, W. Hall, S. Bechhofer, and C. Goble. Conceptual linking: ontology-based open hypermedia. In *Proceedings of the 10th International World Wide Web Conference*, pages 334–342, Hong Kong, China, 2001.

[23] M. Ciocoiu and D. Nau. Ontology-based semantics. In A. Cohn, F. Giunchiglia, and B. Selman, editors, *KR2000: Principles of Knowledge Representation and Reasoning*, pages 539–546, San Francisco, CA, USA, 2000. Morgan Kaufmann.

[24] P. Codognet and D. Diaz. WAMCC: Compiling prolog to C. In *International Conference on Logic Programming*, pages 317–331, Kanazawa, Japan, June 1995.

[25] The Gene Ontology Consortium. Creating the gene ontology resource: design and implementation. *Genome Research*, 11:1425–1433, 2001.

[26] T. Cooper and N. Wogrin. *Rule-based Programming with OPS5*. Morgan Kaufmann, San Francisco, CA, USA, 1988.

[27] S. Cranefield. UML and the Semantic Web. In *Proceedings of the International Semantic Web Working Symposium (SWWS)*, 2001. http://www.semanticweb.org/SWWS/program/full/paper1.pdf.

[28] K. Decker, A. Pannu, K. Sycara, and M. Williamson. Designing behaviors for information agents. In W. Johnson and B. Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 404–412, Marina del Rey, CA, USA, 5–8, 1997. ACM Press.

[29] S. Decker, D. Brickley, J. Saarela, and J. Angele. A query and inference service for rdf. Position paper for the W3C Query Languages meeting in Boston, December 1998. http://www.w3.org/TandS/QL/QL98/pp/queryservice.html.

[30] G. Denker, J. Hobbs, D. Martin, S. Narayanan, and R. Waldinger. Accessing information and services on the DAML-enabled Web. In S. Decker, D. Fensel, A. Sheth, and S. Staab, editors, *Proceedings of the Second International Workshop on the Semantic Web - SemWeb*, Hong Kong, China, May 2001.

[31] J. Denzinger. Distributed deduction and the Semantic Web. In *Proceedings of the AI-2002 Workshop on Business Agents and the Semantic Web*, pages 2–9, May 2002.

[32] C. Draxler. Prolog to SQL compiler. Technical report, CIS Centre for Information and Speech Processing, Ludwig-Maximilians-University, Munich, August 1993.

[33] J. Dvorak. Using clips in the domain of knowledge-based massively parallel programming. In G. Riley, editor, *Proceedings of the Third Conference on CLIPS*, pages 195–202, September 1994.

[34] A. Eberhart. Applications of the Semantic Web for document retrieval. In *Position paper at the Semantic Web Working Symposium (SWWS 2001)*, Stanford University, CA, USA, July 2001. http://www.semanticweb.org/SWWS/program/position/soi-eberhart.pdf.

[35] A. Eberhart. Building a domain-aware e-support system. In *Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR 2001)*, L'Aquila, Italy, August 2001. CD ROM Paper 44.

[36] A. Eberhart. An agent infrastructure based on Semantic Web standards. In *Proceedings of the Business Agents and the Semantic Web Workshop. In conjunction with the Fifteenth Canadian Conference on Artificial Intelligence*, pages 10–17, Calgary, AB, Canada, May 2002.

[37] A. Eberhart. Automatic generation of Java/SQL based inference engines from RDF Schema and RuleML. In I. Horrocks and J. Hendler, editors, *Proceedings of the First International Semantic Web Conference (ISWC 2002)*, pages 102–116, Chia, Sardinia, Italy, June 2002. Springer.

[38] A. Eberhart. OntoAgent: A platform for the declarative specification of agents. In M. Schroeder and G. Wagner, editors, *Proceedings of the international Workshop on Rule Markup Languages for Business Rules on the Semantic Web. In conjunction with the first International Semantic Web Conference (ISWC 2002)*, pages 58–71, Chia, Sardinia, Italy, July 2002.

[39] A. Eberhart. SmartGuide: An intelligent information system basing on Semantic Web standards. In *Proceedings of the International Conference on Artificial Intelligence)*, Las Vegas, NV, USA, June 2002. CD ROM.

[40] A. Eberhart. Survey of RDF data on the Web. In *Proceedings of the 6th World Multiconference on Systems, Cybernetics and Informatics (SCI-2002)*, Orlando, FL, USA, July 2002. CD ROM Volume XI.

[41] A. Eberhart. Survey of RDF data on the Web. Technical Report TR-01-2002, International University in Germany, Bruchsal, Germany, August 2002. Updated version with new results from 8/2002.

[42] A. Eberhart. Towards universal Web Service clients. In B. Hopgood, B. Matthews, and M. Wilson, editors, *Proceedings of the Euroweb 2002: The Web and the GRID: from e-science to e-business*, Oxford, UK, December 2002. http://www1.bcs.org.uk/DocsRepository/03700/3780/eberhart.htm.

[43] A. Eberhart. Semantic Web meets electronic commerce. *Electronic Commerce Research and Applications. Special issue on Headways in E-Commerce with the Semantic Web. Elsevier Science, London, UK*, 2003. accepted to appear.

[44] A. Eberhart and S. Fischer. *Java Tools: Using XML, EJB, Corba, Servlets and SOAP.* John Wiley and Sons, Chichester, GB, 2002.

[45] A. Eberhart, F. Schiele, and S. Fischer. The electronic course authoring and management system at the international university. In *Proceedings of the 4th World Congress on Integrated Design and Process Techology (IDPT99), published at IDPT2000*, Dallas, TX, USA, 2000.

[46] R. Elmasri and S. Navathe. *Fundamentals of Database Systems*, chapter 24, pages 729–760. Addison-Wesley, Redwood City, CA, USA, second edition, 1992.

[47] M. Erdmann and R. Studer. Ontologies as conceptual models for XML documents. In *Proceedings of the 12th Workshop for Knowledge Acquisition, Modeling and Management (KAW'99), Banff, Canada*, October 1999.

[48] J. Everett, D. Bobrow, R. Stolle, R. Crouch, V. de Paiva, C. Condoravdi, M. van den Berg, and L. Polanyi. Making ontologies work for resolving redundancies across documents. *Communications of the ACM*, 45(2), Feburary 2002.

[49] D. Fensel and C. Bussler. The web service modeling framework wsmf. Technical Report IR-493, Vfije Universiteit Amsterdam, Faculteit der Exacte Wetenschappen, Divisie W&I, February 2002.

[50] D. Fensel, J. Hendler, H. Lieberman, and W. Wahlster, editors. *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential.* MIT Press, Cambridge, MA, USA, 2002.

[51] D. Fensel and A. Perez. A survey on ontology tools. Technical Report OntoWeb Deliverable 1.3, OntoWeb consortium, May 2002. http://www.ontoweb.org/ download/deliverables/D13_v1-0.zip.

[52] D. Fensel, F. van Harmelen, I. Horrocks, D. McGuinness, and P. Patel-Schneider. OIL: An ontology infrastructure for the Semantic Web. *IEEE Intelligent Systems*, 16(2):38–44, 2001.

[53] R. Fikes and A. Farquhar. Distributed repositories of highly expressive reusable knowledge. *IEEE Intelligent Systems*, 14(2):73–79, March/April 1999.

[54] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an Agent Communication Language. In N. Adam, B. Bhargava, and Y. Yesha, editors, *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, pages 456–463, Gaithersburg, MD, USA, 1994. ACM Press.

[55] FIPA. FIPA KIF content language specification. Technical Report XC00010B, Foundation for Intelligent Physical Agents, 2000. http://www.fipa.org/specs/fipa00010/.

[56] C. Forgy. Rete: A fast algorithm for the many pattern/many object pattern matching problem. *Artificial Intelligence*, 19:17–37, 1982.

[57] G. Fox. Peer-to-peer networks. *IEEE Computing in Science and Engineering*, May/June:75–77, 2001.

[58] W. Frakes. *Information Retrieval: Data Structures and Algorithms*, chapter Stemming algorithms, pages 131–160. Prentice Hall, 1992.

[59] D. Franklin and K. Hammond. The intelligent classroom: providing competent assistance. In J. Mueller, E. Andre, S. Sen, and C. Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 161–168, Montreal, QC, Canada, 2001. ACM Press.

[60] S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Agent Theories, Architectures, and Languages*, pages 21–35, 1996.

[61] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley, Redwood City, CA, USA, 1995.

[62] M. Genesereth and R. Fikes. Knowledge interchange format, version 3.0 reference manual. Technical Report Logic-92-1, Computer Science Department, Stanford University, 1992.

[63] L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. *Relational Data Mining*, chapter Learning Probabilistic Relational Models. Springer-Verlag, Heidelberg, D, 2001.

[64] L. Getoor, D. Koller, and N. Friedman. From instances to classes in probabilistic relational models. In *Proceedings of the ICML-2000 Workshop on Attribute-Value and Relational Learning: Crossing the Boundaries*, Stanford, CA, USA, June 2000.

[65] S. Graham, S. Simeonov, T. Boubez, G. Daniels, D. Davis, Y. Nakamura, and R. Neyama. *Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI*. Sams, Indianapolis, IN, USA, December 2001.

[66] B. Grosof. Representing e-business rules for the Semantic Web: Situated courteous logic programs in RuleML. In *Proceedings of the Workshop on Information Technologies and Systems (WITS '01). In conjunction with the Internatinal Conference on Information Systems (ICIS-2001)*, New Orleans, LA, USA, December 2001.

[67] B. Grosof, M. Gandhe, and T. Finin. Sweetjess: Translating DAML-RuleML to JESS. In M. Schroeder and G. Wagner, editors, *Proceedings of the international Workshop on Rule Markup Languages for Business Rules on the Semantic Web. In conjunction with the first International Semantic Web Conference (ISWC 2002)*, pages 5–25, Chia, Sardinia, Italy, July 2002.

[68] B. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *Proceedings of the Twelfth International World Wide Web Conference (WWW 2003)*, Budapest, Hungary, May 2003. http://www2003.org/cdrom/papers/refereed/p117/p117-grosof.html.

[69] B. Grosof, Y. Labrou, and H. Chan. A declarative approach to business rules in contracts: courteous logic programs in XML. In M. Wellman, editor, *Proceedings of the first ACM Conference on Electronic Commerce (EC-99)*, pages 68–77, Denver, CO, USA, 1999.

[70] B. Grosof and T. Poon. Representing agent contracts with exceptions using XML rules, ontologies, and process descriptions. In M. Schroeder and G. Wagner, editors, *Proceedings of the international Workshop on Rule Markup Languages for Business Rules on the Semantic Web. In conjunction with the first International Semantic Web Conference (ISWC 2002)*, pages 72–93, Chia, Sardinia, Italy, July 2002.

[71] T. Gruber. Ontolingua: A mechanism to support portable ontologies. Technical Report KSL-91-66, Knowledge Systems Laboratory, Stanford University, Palo Alto, CA, USA, March 1992.

[72] T. Gruber. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, NL, 1993. Kluwer Academic Publishers.

[73] N. Guarino, C. Masolo, and G. Vetere. Ontoseek: Content-based access to the Web. *IEEE Intelligent Systems*, 14(3):70–80, May 1999.

[74] N. Guarino and C. Welty. A formal ontology of properties. In R. Dieng and O. Corby, editors, *14th European Conference on Artificial Intelligence,*

Workshop on Applications of Ontologies and Problem-Solving Methods, pages 97–112, August 2000.

[75] N. Guarino and C. Welty. Evaluating ontological decisions with ontoclean. *Communications of the ACM*, 45(2), Feburary 2002.

[76] R. Guha, O. Lassila, E. Miller, and D. Brickley. Enabling inferencing. Position paper for the W3C Query Languages meeting in Boston, December 1998. http://www.w3.org/TandS/QL/QL98/pp/enabling.html.

[77] K. Hammond, R. Burke, and S. Lytinen. *A Case-Based Approach to Knowledge Navigation. In D. Leake, editor. Case-Based Reasoning Experiences Lessons and Future Directions*, pages 125–136. MIT Press, 1996.

[78] S. Harabagiu, M. Pasca, and S. Maiorano. Experiments with open-domain textual question answering. In *Proceedings of the 18th International Conference on Computational Linguistics*, Saarbrueken Germany, August 2000.

[79] P. Hayes. Catching the dreams. http://www.aifb.uni-karlsruhe.de/~sst/is/WebOntologyLanguage/hayes.htm, 2002.

[80] J. Heflin, J. Hendler, and S. Luke. SHOE: A knowledge representation language for internet applications. Technical Report CS-TR-4078, Department of Computer Science, University of Maryland, 1999.

[81] J. Hendler. Agents and the Semantic Web. *IEEE Intelligent Systems*, 16(2), 2001.

[82] J. Hendler and D. McGuinness. The DARPA agent markup language. *IEEE Intelligent Systems*, 15(6):72–73, Nov./Dec. 2000.

[83] U. Hermjakob, E. Hovy, and C. Lin. Knowledge-based question answering. In *Proceedings of the 6th World Multiconference on Systems, Cybernetics and Informatics (SCI-2002)*, Orlando, FL, USA, July 2002.

[84] J. Hiller. MANDARAX - a Java opensource implementation of rule-based technologies. In *Proceedings of the International Conference on Object Technology*, 2001.

[85] W. Holsapple and K. Joshi. A collaborative approach to ontology design. *Communications of the ACM*, 45(2), Feburary 2002.

[86] D. Huynh, D. Karger, and D. Quan. Haystack: A platform for creating, organizing and visualizing information using rdf. In *Semantic Web Workshop. In conjunction with the Eleventh World Wide Web Conference*, 2002.

[87] J. Jacobs and A. Linden. Semantic Web technologies take middleware to next level. Technical Report T-17-5338, Gartner Group, August 2002.

[88] G. Karvounarakis, V. Christophides, D. Plexousakis, and S. Alexaki. Querying RDF descriptions for community web portals. In *17iemes Journees Bases de Donnees Avancees (BDA'01)*, pages 133–144, Agadir, Marocco, October 2001.

[89] B. Katz. From sentence processing to information access on the World Wide Web. In *Proceedings of the AAAI Spring Symposium on Natural Language Processing for the World Wide Web*, pages 77–86, Stanford, CA, USA, 1997.

[90] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42, 1995.

[91] H. Kim. Predicting how ontologies for the Semantic Web will evolve. *Communications of the ACM*, 45(2), Feburary 2002.

[92] D. Koller and A. Pfeffer. Probabilistic frame-based systems. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI)*, pages 580–587, Madison, WI, USA, 1998.

[93] W. Kraaij and R. Pohlmann. Viewing stemming as recall enhancement. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 40–48, Zuerich, Switzerland, 1996. ACM Press.

[94] H. Krieger and U. Schaefer. TDL – a type description language for HPSG, part 1: Overview. Technical Report RR-94-37, DFKI, November 1994.

[95] C. Kwok, O. Etzioni, and D. Weld. Scaling question answering to the Web. In *Proceedings of the 10th International World Wide Web Conference*, pages 150–161, Hong Kong, China, May 2001.

[96] Y. Labrou and T. Finin. Yahoo! as an ontology: Using yahoo! categories to describe documents. In *Proceedings of the Eight International Conference of Information Knowledge Management (CIKM 1999)*, pages 180–187, Kansas City, MO, USA, October 1999.

[97] O. Lassila. Web metadata: A matter of semantics. *IEEE Internet Computing*, 2(4):30–37, 1998.

[98] S. Lawrence and C. Giles. Searching the World Wide Web. *Science*, 280(5360):98–100, 1998.

[99] J. Lee. Icoma: An open infrastructure for agentbased intelligent electronic commerce on the internet. In *Proceedings of the International Conference on Parallel and Distributed Systems (CPADS)*, pages 648–655, Los Alamitos, CA, USA, 1997. IEEE Comp Soc.

[100] A. Levy and M. Rousset. Combining horn rules and description logics in CARIN. *Artificial Intelligence Journal*, 104, September 1998.

[101] R. Lienhart, W. Effelsberg, and R. Jain. Towards a visual grep: A systematic analysis of various methods to compare video sequences. In R. Jain I. Sethi, editor, *Storage and Retrieval for Image and Video Databases VI*, volume SPIE 3312, pages 271–282, 1998.

[102] A. Maedche, M. Ehrig, S. Handschuh, R. Volz, and L. Stojanovic. Ontology-focused crawling of documents and relational metadata. In *Proceedings of the Eleventh International World Wide Web Conference WWW-2002, (Poster)*, May 2002.

[103] A. Maganaraki, G. Karvounarakis, V. Christophides, D. Plexousakis, and T. Anh. Ontology storage and querying. Technical Report 308, Foundation for Research and Technology Hellas, Institute of Computer Science, Information Systems Laboratory, April 2002.

[104] A. Magkanaraki, S. Alexaki, V. Christophides, and D. Plexousakis. Benchmarking RDF schemas for the Semantic Web. In I. Horrocks and J. Hendler, editors, *Proceedings of the First International Semantic Web Conference (ISWC 2002)*, pages 132–146, Chia, Sardinia, Italy, June 2002. Springer.

[105] R. Malaka and A. Zipf. Deep Map - challenging IT research in the framework of a tourist information system. In D. Buhalis D. Fesenmaier, S. Klein, editor, *Proceedings of 7th. International Congress on Tourism and Communications, ENTER2000*, pages 15–27, Barcelona, Spain, 2000. Springer.

[106] T. Malone, K. Crowston, J. Lee, B. Pentland, C. Dellarocas, G. Wyner, J. Quimby, C. S. Osborn, A. Bernstein, G. Herman, M. Klein, and E. ODonnell. Tools for inventing organizations: Toward a handbook of organizational processes. *Management Science*, 45(3):425–443, 1999.

[107] P. Martin and P. Eklund. Large-scale cooperatively-built heterogeneous kbs. In *Proceedings of ICCS 2001, 9th International Conference on Conceptual Structures*, pages 231–244, Heidelberg, D, August 2001. Springer Verlag.

[108] M. Maybury and W. Wahlster, editors. *Readings in Intelligent User Interfaces*. Morgan Kaufmann, San Francisco, CA, USA, 1998.

[109] R. McGrath. Discovery and its discontents: Discovery protocols for ubiquitous computing. Technical Report UIUCDCS-R-99-2132, Department of Computer Science University of Illinois Urbana-Champaign, IL, USA, March 2000.

[110] D. McGuinness. *Ontologies Come of Age. In D. Fensel and J. Hendler and H. Lieberman and W. Wahlster, editors. Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*, pages 171–197. MIT Press, Cambridge, MA, USA, 2002.

[111] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, 1997.

[112] S. McIlraith, T. Son, and H. Zeng. Semantic Web services. *IEEE Intelligent Systems*, 16(2), 2001.

[113] G. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller. Introduction to wordnet: An on-line lexical database. *International Journal of Lexicography*, 3:245–264, 1990.

[114] B. Motik, A. Maedche, and R. Volz. A conceptual modeling approach for semantics-driven enterprise applications. In *Proceedings of the First International Conference on Ontologies, Databases and Application of Semantics (ODBASE-2002)*, Irvine, CA, USA, October 2002. Springer.

[115] I. Mumick and H. Pirahesh. Implementation of magic-sets in a relational database system. In *Proceedings of the ACM SIGMOD Int. Conf. on Management of Data*, pages 103–114, Minneapolis, MN, USA, May 1994.

[116] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmer, and T. Risch. EDUTELLA: A P2P networking infrastructure based on RDF. In *Proceedings of the eleventh international conference on World Wide Web*, pages 604–615, Hawaii, USA, 2002. ACM Press.

[117] N. Noy, M. Sintek, S. Decker, M. Crubezy, R. Fergerson, and M. Musen. Creating Semantic Web contents with Protege-2000. *IEEE Intelligent Systems*, 16(2):60–71, 2001.

[118] R. Orfali and D. Harkey. *Client/Server Programming with Java and CORBA*. John Wiley and Sons, 1997.

[119] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Importing the Semantic Web in UDDI. In C. Bussler, R. Hull, S. McIlraith, M. Orlowska, B. Pernici, and J. Yang, editors, *Proceedings of the Workshop on Web Services, E-Business and Semantic Web, CAiSE 2002*, pages 225–236, Toronto, ON, Canada, June 2002.

[120] M. Paolucci, T. Kawmura, T. Payne, and K. Sycara. Semantic matching of Web Services capabilities. In I. Horrocks and J. Hendler, editors, *Proceedings of the First International Semantic Web Conference (ISWC 2002)*, pages 333–347, Chia, Sardinia, Italy, June 2002. Springer.

[121] S. Pfeiffer, R. Lienhart, S. Fischer, and W. Effelsberg. Abstracting digital movies automatically. *Journal of Visual Communication and Image Representation*, 7:345–353, December 1996.

[122] J. Pollock. The Web Services scandal. *EAI Journal*, August 2002.

[123] R. Ramakrishnan and J. Ullman. A survey of research on deductive database systems. *Journal of Logic Programming*, 23(2):125–149, 1993.

[124] P. Van Roy. *Can Logic Programming Execute as Fast as Imperative Programming?* PhD thesis, University of California, Berkley, December 1990.

[125] K. Sagonas, T. Swift, and D. Warren. XSB as an efficient deductive database engine. In R. Snodgrass and M. Winslett, editors, *Proceedings of the ACM SIGMOD Int. Conf. on Management of Data*, pages 442–453, 1994.

[126] K. Sagonas, T. Swift, D.S. Warren, J. Freire, and P. Rao. The XSB programmer's manual, version 2.1 volume 2: Libraries and interfaces, April 1999.

[127] A. Schreiber, B. Dubbeldam, J. Wielemaker, and B. Wielinga. Ontology-based photo annotation. *IEEE Intelligent Systems*, May/June 2001.

[128] K. Scribner and M. Stiver. *Understanding SOAP: The Authoritative Solution*. Sams, Indianapolis, IN, USA, 2000.

[129] P. Seshadri, J. Hellerstein, H. Pirahesh, T. Leung, R. Ramakrishnan, D. Srivastava, P. Stuckey, and S. Sudarshan. Cost-based optimization for magic: algebra and implementation. In *Proceedings of the ACM SIGMOD Int. Conf. on Management of Data*, pages 435–446, Montreal, QC, Canada, June 1996.

[130] S. Singh. *The Code Book*, chapter 7. Anchor Books, Garden City, NY, August 2000.

[131] M. Sintek and S. Decker. TRIPLE - an RDF query, inference, and transformation language. In *Proceedings of the International Conference on Applications of Prolog*, pages 47–56, Tokyo, JP, October 2001.

[132] J. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks Cole Publishing Co., Pacific Grove, CA, USA, August 1999.

[133] S. Staab, M. Erdmann, A. Maedche, and S. Decker. An extensible approach for modeling ontologies in RDF(S). In *Proceedings of the ECDL Workshop on the Semantic Web*, pages 11–22, Lisbon, Portugal, September 2000.

[134] A. Steinacker, C. Seeberg, S. Fischer, and R. Steinmetz. Multibook: Metadata for the Web. In *2nd International Conference on New Learning Technologies*, pages 16–24, Bern, Switzerland, 1999.

[135] G. Stumme and A. Maedche. FCA-merge: A bottom-up approach for merging ontologies. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 225–234, Seattle, WA, USA, August 2001. Morgan Kaufmann.

[136] D. Suthers. Using learning object meta-data in a database of primary and secondary school resources. In *Proceedings of International Conference on Computers in Education*, Taipei, TW, November 2000.

[137] A. Swartz. MusicBrainz: A Semantic Web Service. *IEEE Intelligent Systems*, 17(1):76–77, 2002.

[138] A. Swartz and J. Hendler. The Semantic Web: A network of content for the digital city. In *Proceedings Second Annual Digital Cities Workshop*, Kyoto, Japan, October 2001.

[139] A. Uhl and H. Lichter. New wave searchables: Changing the paradigm of internet-scale search. In *Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR 2001)*, August 2001. CD ROM Paper 148.

[140] J. Ullman and J. Widom. *A First Course in Database Systems*. Prentice Hall, Englewood Cliffs, NJ, USA, 1997.

[141] M. Uschold and R. Jasper. A framework for understanding and classifying ontology applications. In *Proceedings of the IJCAI99 Workshop on Ontologies and Problem-Solving Methods(KRR5), Stockholm, Sweden*, volume 18, August 1999.

[142] L. van Elst and A. Abecker. Domain ontology agents for distributed organizational memories. *In R. Dieng-Kuntz and N. Matta, editors. Knowledge Management and Organizational Memories. Kluwer Academic Publishers, Netherlands*, July 2002.

[143] W. Wahlster. User and discourse models for multimodal communication. *In J. Sullivan and S. Tyler, editors. Int. User Interfaces. New York, NY, USA*, pages 45–67, 1991.

[144] Z. Zheng. Answerbus question answering system. In *Proceeding of HLT Human Language Technology Conference (HLT 2002)*, pages 24–27, San Diego, CA, USA, March 2002.

[145] C. Ziegler. Deus ex machina. *CT. Heise Verlag, Hannover, Germany*, 6, 2002.