Programming Semantic Web Applications: A Synthesis of Knowledge Representation and Semi-Structured Data

Doctoral Dissertation

Ora Lassila

Nokia Research Center
3 Cambridge Center
Cambridge, MA 02142, USA

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Department of Computer Science and Engineering for public examination and debate in Auditorium TU2 at Helsinki University of Technology (Espoo, Finland) on the 6th of November, 2007, at 12 noon.

Helsinki University of Technology
Department of Computer Science and Engineering
Laboratory of Software Technology

Distribution:
Helsinki University of Technology
Department of Computer Science and Engineering
Laboratory of Software Technology
P.O. Box 5400
FI - 02015 TKK
FINLAND

ISBN 978-951-22-8985-1 (PDF)
ISSN 1239-6885 (PDF)
TKO-A44/07

ABSTRACT OF DOCTORAL DISSERTATION	HELSINKI UNIVERSITY OF TECHNOLOGY P. O. BOX 1000, FI-02015 TKK http://www.tkk.fi

Author Ora Lassila

Name of the dissertation

Programming Semantic Web Applications: A Synthesis of Knowledge Representation and Semi-Structured Data

Manuscript submitted 2007-05-18	Manuscript revised 2007-09-24
Date of the defence 2007-11-06	
☒ Monograph	☐ Article dissertation (summary + original articles)
Department	Department of Computer Science and Engineering
Laboratory	Laboratory of Software Techology
Field of research	Knowledge Engineering
Opponent(s)	Professor Lynn Andrea Stein
Supervisor	Professor Markku Syrjänen
Instructor	

Abstract

Software application development is largely centered around various representations of data and representations of the world in which the software operates. Often, while a software system itself is specified in terms of procedures and *procedural semantics*, the data the system uses and manipulates has *declarative semantics*; connecting the two is often an *ad hoc* endeavor.

The issues of complex data representations are amplified within artificial intelligence applications that employ sophisticated *knowledge representation*. More recently, applications involving *Semantic Web* technologies are faced with the same situation. The Semantic Web is an attempt to enable sophisticated data representation for and within the context of World Wide Web content, aiming to enable more automated and autonomous applications to be built that take advantage of data on the Web. As such, the Semantic Web represents a vision for the next generation of Web applications and Web usage.

This dissertation focuses on the representation of (Semantic Web) data as *directed, labeled graphs*. A method is introduced for querying these representations by expressing *path patterns*, enabling software programs to be "glued" to complex representations. The query mechanism is then extended to implement *reasoning* (i.e., logical inference) for this data, and to hide the reasoning process from application programs. A reasoner is presented for data based on an extended version of the RDF(S) data model. The outcome is a synthesis of two views of (Semantic Web) data, namely the view of the data as a logic formalism, and a view of the data as *semi-structured graphs*.

An evaluation of the query mechanism is presented, contrasted against other approaches to querying RDF(S) data. Examples of various software applications making use of the Semantic Web, the path query mechanism, and the reasoner are also presented.

Keywords Semantic Web, Knowledge Representation, Programming	
ISBN (printed) 978-951-22-8984-4	ISSN (printed) 1239-6885
ISBN (pdf) 978-951-22-8985-1	ISSN (pdf) 1239-6885
Language English	Number of pages 145
Publisher Department of Computer Science and Engineering, Helsinki University of Technology	
Print distribution	
☐ The dissertation can be read at http://lib.tkk.fi/Diss/	

VÄITÖSKIRJAN TIIVISTELMÄ	TEKNILLINEN KORKEAKOULU PL 1000, 02015 TKK http://www.tkk.fi

Tekijä Ora Lassila

Väitöskirjan nimi

Semanttinen Web ja sovellusohjelmointi tietämyksen esittämisen ja puolirakenteisen datan synteesinä

Käsikirjoituksen päivämäärä 2007-05-18	Korjatun käsikirjoituksen päivämäärä 2007-09-24
Väitöstilaisuuden ajankohta 2007-11-06	
☒ Monografia	☐ Yhdistelmäväitöskirja (yhteenveto + erillisartikkelit)

Osasto	Tietotekniikan osasto
Laboratorio	Ohjelmistotekniikan laboratorio
Tutkimusala	Tietämystekniikka
Vastaväittäjä(t)	Professori Lynn Andrea Stein
Työn valvoja	Professori Markku Syrjänen
Työn ohjaaja	

Tiivistelmä

Ohjelmistosovellusten kehittämisessä keskeisessä asemassa ovat erilaiset tiedon esitysmuodot sekä esitykset siitä maailmasta, jossa ohjelmistojärjestemä toimii. Usein, vaikka itse ohjelmisto on määritelty kokoelmana *proseduureja* ja sillä on *proseduraalinen semantiikka*, sen käsittelemällä tiedolla on *deklaratiivinen semantiikka*; näiden kahden yhdistäminessä käytetään monasti varsin satunnaisia menetelmiä.

Monimutkaisen tiedon esittämisen ongelmat ovat erityisen vaikeita edistyneitä *tietämyksen esittämisen* menetelmiä käyttävissä tekoälysovelluksissa. Viime aikoina *Semanttisen Webin* tekniikoita käyttävät sovellukset ovat samojen haasteiden edessä. Semanttinen web on yritys liittää kehittynyttä tiedon esitystä nettisisältöön tai sen yhteyteen, päämääränä enenevässä määrin automaattisten tai itsenäisesti toimivien, nettisisältöä hyödyntävien sovellusten kehittäminen. Tästä näkökulmasta Semanttinen Web edustaa visiota netin seuraavan sukupolven sovelluksista sekä käytöstä.

Tässä väitöskirjassa keskitytään (Semanttisessa webissä olevan) tiedon esittämiseen *suunnattuna verkkona* jonka *kaaret on nimetty*. Väitöskirja esittelee menetelmän, jolla näitä esitysmuotoja voidaan tarkastella ilmaisemalla tietoon kohdistuvat kyselyt *polkuina*; tätä tekniikkaa voidaan sitten käyttää proseduraalisten ohjelmien “liimaamiseen” monimutkaisiin tiedon esitysmuotoihin. Kyselymekanismia laajennetaan toteuttamalla looginen päättelymekanismi em. tiedolle; kyseinen päättely voidaan sitten piilottaa sovellusohjelmalta. Väitöskirjassa esitettävää päättelymekanismia käytetään laajennetun RDF(S)-tietomallin kanssa. Tuloksena on synteesi kahdesta eri tavasta tarkastella Semanttisen webin dataa (joko loogisena formalismina tai *puolirakenteisena* tietona).

Kyselymenetelmää arvoidaan suhteessa muihin tapoihin kysellä RDF(S)-dataa. Lopuksi esitellään esimerkkejä sovellusohjelmista, jotka käyttävät Semanttista webiä, polkukyselyjä sekä päättelyä.

Asiasanat Semanttinen web, tietämyksen esittäminen, ohjelmointi

ISBN (painettu) 978-951-22-8984-4	ISSN (painettu) 1239-6885
ISBN (pdf) 978-951-22-8985-1	ISSN (pdf) 1239-6885
Kieli Englanti	Sivumäärä 145

Julkaisija Tietotekniikan osasto, Teknillinen korkeakoulu

Painetun väitöskirjan jakelu

☐ Luettavissa verkossa osoitteessa http://lib.tkk.fi/Diss/

Preface

For the past 20 years I have been interested in combining procedural programs with declarative knowledge representation. In 1996 I started thinking of knowledge representation on the World Wide Web; this work led to the emergence of the "Semantic Web" and its basic building block, the RDF representation language. Naturally I wanted to apply my earlier work now in the context of RDF: in this thesis I discuss possible solutions to this specific integration problem, seen as part of a larger set of obstacles in developing "Semantic Web applications", mainstream software systems that exploit the potential of the Semantic Web.

Markku Syrjänen (at HUT) and Stephen F. Smith (at CMU) were instrumental in supporting my early experiments in integration. Tim Berners-Lee (at MIT) urged me to pursue KR in the context of the World Wide Web (by asking me in 1996: "What do *you* think is wrong with the Web?"). Several other people also helped me to realize the dream of the Semantic Web: Mark Adler, Sadhna Ahuja, Art Barstow, Jan Bosch, Franklin Davis, Wayne DeMello, Li Ding, Sapna Dixit, Tim Finin, Barbara Heikkinen, Jim Hendler, Jamey Hicks, Ian Horrocks, Jyri Huopaniemi, Eero Hyvönen, Bob Iannucci, Deepali Khushraj, Juhani Kuusi, Pertti Lounamaa, Eve Maler, David Martin, Deborah McGuinness, Sheila McIllraith, Eric Miller, Jim Miller, Heli Nyholm, Terry Payne, Franklin Reynolds, Heikki Saikkonen, Marko Suoknuuti, Ralph Swick, Katia Sycara, Louis Theran, Danny Weitzner, and others. I would also like to thank the anonymous reviewers of my conference articles on which this dissertation is largely based. Furthermore, I am indebted to the friendly staff of Starbucks coffee shops in Nashua, NH and Leominster, MA where most of this thesis was written.

I am grateful to my wife Marcia and my daughters Lauren and Grace for their love, support, and patience. Finally, I would like to dedicate this work to my parents, Heljä and Ola Lassila: without you, none of this would ever have happened.

– Ora Lassila (Hollis, NH, September 2007)

Contents

Preface **5**

Contents **7**

List of Figures **11**

List of Tables **13**

1 Introduction **15**

1.1 Structure of the Dissertation 17

1.2 Relation to Author's Earlier Work 18

2 On Data Representations **20**

2.1 Knowledge Representation and Reasoning 20

2.2 Integrating Applications with Data Representations 25

3 Semantic Web **26**

3.1 Automation and Agents 27

3.1.1 Meaning of "Meaning" 30

3.2 Semantic Web Formalisms 34

3.2.1 Resource Description Framework 34

3.2.2 Other Formalisms 39

3.3 Querying Semantic Web Data 40

3.4 Service-Oriented Computing and Web Services 44

3.4.1 Semantic Web Services 44

3.4.2 Agent-Based Systems 46

3.5 Semantic Web Use Case: Ubiquitous Computing 47

3.6 A Philosophical Note 48

4 Challenges in Building Semantic Web Applications **49**

4.1 Using Complex Declarative Representations 51

4.1.1 "Identity Crisis" in RDF 51

4.2 Using Reasoning 52

4.3 Implementing Serendipity 52

5 Exposing Representation to Application Logic **54**

5.1 Traditional Approaches to Data Interfaces 54

5.2 Procedural Attachment in Frame-based Systems 55

5.3 A Path Query Language 60

5.3.1 Implementing the Path Query Language 62

6 Hiding Reasoning from Application Logic **67**

6.1 Exposing the Deductive Closure 70

6.2 Entailment and "RDF(S)-Closures" 71

6.3 Reasoning in RDF(S) via Forward-Chaining Rules 72

6.4 Reasoning in RDF(S) as Theorem-Proving 73

6.5 Reasoning in RDF(S) as Query Rewriting 73

6.5.1 Type and Subclass Rules 75

6.5.2 Subproperty Rules 77

6.5.3 Domain/Range Rules 78

6.5.4 Inverting Paths with Default Values 82

6.5.5 Implementation Summary and Evaluation 83

6.6 Generalization of Rewriting Approach to Reasoning 84

7 Practicality of RDF(S) in Applications **87**

7.1 Semantic Theory for RDF^{++} 88

8 Concrete Semantic Web Platform **91**

8.1 Toolkit Concepts 92
8.2 Hidden Reasoner 93
8.3 Input and Output of Data 94
8.4 Practical Evaluation of Expressive Power of WilburQL 94

9 Exploiting Reasoning and Serendipity in Applications 98
9.1 Trivial Example: An RSS Formatter 98
9.2 Browsing Semantic Data 99
9.2.1 Browsing RDF Data 100
9.2.2 Architecture and Implementation of OINK 102
9.2.3 Lessons Learned from OINK 105
9.3 Social Networks and Organizational Structures 106
9.4 Service Composition and Substitution 109
9.4.1 "DAML-S Lite" – a Subset of OWL-S/DAML-S 110
9.4.2 Service Substitution 111
9.4.3 Composing Workflows 112
9.4.4 Practical Implementation Using RDF(S) 113
9.4.5 Lessons Learned from Service Substitution 115

10 Conclusions 116

References 119

List of Figures

3.1 Semantic Web "layers" 35
3.2 Statement $\langle A, P, B \rangle$ 36
3.3 Typical PICS label 37
3.4 Simple graph example 42

4.1 Requirement of expressive power in applications discussed in this dissertation 50

5.1 Genealogy of author's frame-based representation systems 56

6.1 DFA corresponding to the complex path from rewrite pattern (6.6) 76
6.2 Reified Statement S 79
6.3 Computed shortcut $seq(inv(\texttt{rdf:subject}), \texttt{rdf:predicate})$ 80
6.4 Inferring the type of node A using the domain rule 81
6.5 DFA corresponding to the complex path from rewrite pattern (6.21) 82

9.1 RSS-to-XHTML formatter 99
9.2 A typical page from OINK 101
9.3 The overall architecture of OINK 103
9.4 Visualization of navigation history and automatically generated queries . . . 106
9.5 Exporting a FoaF profile from the phone 107
9.6 Substitution of a Weather Information Service 114

List of Tables

2.1 Terminological correspondences between OOP, frame systems, and DL 23

6.1 Implementation Summary 83

8.1 Comparison of RDF Query Languages 97

1 Introduction

The best way to become acquainted with a subject is to write a book about it.
– Benjamin Disraeli

Any non-trivial software system will embody *representations* of the world (or some aspect or detail thereof) in *multiple* forms. Most notably, such a system is a combination of *program code*, with *procedural* semantics, and *data*, with (typically) *declarative* semantics. By and large, the procedural and declarative semantics are "connected" mostly via informal methods and/or completely "out-of-band" (for example, the details of the connection are only known to the software developer and, in the worst case, "live" inside his head). The aspects of the connection also include actual methods of access to the data. The connection is most "natural" when the programming language's native data representation is used; usually, however, data sources are external, and data has to be marshalled/unmarshalled when accessed by a program. Examples of typical external data representations are databases, documents (e.g. XML documents) and knowledge bases. This dissertation addresses some of the issues of "connecting" programs with data, particularly when it comes to *knowledge representation* and knowledge bases.

Connecting programs with data, especially *external* data, is particularly important for the *Semantic Web* [21]. The Semantic Web is an idea of the future direction of the World Wide Web, aimed at making Web content (i.e., *data*) more amenable to not only processing but also *intepretation* by machines. In many ways, the Semantic Web is not a new idea; instead, it is an amalgamation of pre-existing technologies from multiple domains and communities, and it is this amalgamation, if anything, that is "new." The technological components of the Semantic Web include *networking technologies* (particularly those associated with the Internet), *knowledge representation & reasoning*, *information retrieval*, *agent-based systems*, and others. In this dissertation, I will primarily focus on the knowledge representation aspects

of the Semantic Web, particularly as they are related to concrete application development. On the one hand, one could argue that the Semantic Web knowledge representation has its legacy in *frame-based* representation systems, and my work on the Semantic Web and *Semantic Web programming* has been strongly influenced by work undertaken in developing frame-based representation systems. On the other, techniques developed for processing *semistructured data* [1] can be applied to Semantic Web representations, particularly representations expressed in W3C's *Resource Description Framework* (RDF) language.

Despite the original somewhat high-flying vision for the technology [21], the concrete realization of the Semantic Web has proven to be quite challenging. There are a number of reasons for this, some of which are not technical in nature, but have more to do with the social, "cultural" and business environments where the Semantic Web technology has to be deployed [175]. From the technical perspective – and this is the viewpoint taken in this dissertation – the challenges are often quite pragmatic. I will claim that one of the main challenges of implementing "Semantic Web applications" (i.e., software systems making use of Semantic Web technologies and information sources) is the difficulty in providing a harmonious co-existence for traditional (procedural) programming techniques with largely declarative knowledge representation and reasoning technologies which form the core of the Semantic Web. By and large, the Semantic Web is *all about data*, hence issues of accessing, acquiring, transforming, manipulating and communicating information are paramount (this is not to say that there would not exist similar issues in other software domains).

Rather than forcing software developers to deal with the *logic-based* view of the Semantic Web and associated representational models (and one could argue that this is – if not incomprehensible – at the very least "alien" to most developers), presenting the models as familiar *data structures* (and programming methods) would make the Semantic Web more palatable to the larger developer community. In this light, what follows is not an attempt to invent, say, better reasoning methods but rather an approach to make things more readily

acceptable to, and thus deployable by, the largest number of developers.

If one follows the line of thinking presented in the original Semantic Web vision [21] one will observe that one of the fundamental differences between "traditional" software systems and those systems that exploit the Semantic Web is that the latter are required to be able to deal with "unanticipated" situations and data representations (and thus *agent-based systems* are acknowledged as a key inspiration for the development of the Semantic Web idea). This aspect further emphasizes the need for technologies that make it easy and straightforward for programs to deal with complex representations of information.

The *mainstream adoption* of Semantic Web technologies is predicated on mainstream application developers building applications that take advantage of Semantic Web technologies. These technologies have aspects that are often completely alien to typical software developers (e.g., reasoning). The more we can do to ease the software development process, possibly by hiding some of the key mechanisms or "packaging" them in a palatable way, the better the chances for adoption. This is ultimately the goal of this dissertation.

1.1 Structure of the Dissertation

Chapter 2 will review the prerequisite technologies for understanding and implementing Semantic Web applications, with emphasis on pragmatic software issues. Chapter 3 will present a vision for the Semantic Web, as well as an overview of the current state of Semantic Web technologies. Chapter 4 discusses some of the pragmatic challenges in building Semantic Web software, and will be the basis of the subsequent discussion on program/representation integration.

Chapter 5 demonstrates how to expose declarative representations to procedural programs in a "natural" and convenient way; emphasis will be on a *path-based query language* that allows programs and representation to be "glued" together. The integration of data and procedural

programs is discussed in the context of World Wide Web Consortium's (W3C) Resource Description Framework (RDF) -standard in particular. In a sense, this work represents the amalgamation of "Semantic Web applications", applications that make use of a frame-based representation system, and the use of semi-structured data.

Chapter 6 will discuss how to build programs that access the *deductive closure* of the representation at hand, and how to hide the associated reasoning processes and mechanisms (that effectively create the deductive closure); this will be done by proposing a reasoning algorithm based on the query language presented in the preceding chapter. Chapter 7 will discuss the practical applicability of RDF in software applications and will introduce and discuss extensions of the representation language that improve the applicability. It will also demonstrate how these "new" languages can be implemented using the approach proposed in the previous chapters, and why the proposed extensions are critical to Semantic Web applications.

Chapter 8 describes a concrete implementation of a "Semantic Web platform" based on the ideas presented in Chapters 5–7. Chapter 9 will then present some examples of how this platform can be used. The overall conclusions will be discussed in Chapter 10.

1.2 Relation to Author's Earlier Work

> *Keep true to the dreams of thy youth.*
>
> – Friedrich von Schiller

This dissertation is based on my earlier published and unpublished work from the past ten years (and even more). The intent, generally speaking, is to take the original ideas of the Semantic Web, as I view them, and use them as insight into how such software applications should be developed and structured, such that they can take advantage of the Semantic Web technologies. As such, the dissertation draws heavily from my previously published work,

especially the following:

- *Semantic Web* in general [21] and *RDF* in particular [122, 124, 141].
- *Integration of frame-based representation and object-oriented programming* in general [95, 118, 120, 144, 121, 117, 140] as well as *integration of RDF and programming* in particular [125, 127, 133].
- *Applications of Semantic Web technologies* in general [135, 131, 138, 132, 139], and *Semantic Web Services* in particular [126, 137, 174, 110].

2 On Data Representations

> *To be sure, mathematics can be extended to any branch of knowledge, including economics, provided the concepts are so clearly defined as to permit accurate symbolic representation. That is only another way of saying that in some branches of discourse it is desirable to know what you are talking about.*
>
> – James R. Newman

A discussion on *representation* will serve as background information for this thesis. It can be recognized that issues of *access and manipulation of data* are some of the most fundamental in computer science, and therefore cut across all areas of the discipline. Focus in this thesis, however, will be on aspects of *knowledge representation* because this provides the most appropriate background for a subsequent discussion on the Semantic Web and related technologies.

2.1 Knowledge Representation and Reasoning

Knowledge Representation (KR) is an important subfield of artificial intelligence, aiming to facilitate the representation of information (about the "real world") in such a way that automated systems can better interpret that information. To quote Brachman and Levesque [28]:

> The notion of *representation of knowledge* is at heart an easy one to understand. It simply has to do with writing down, in some language or communicative medium, descriptions or pictures that correspond in some salient way to the world or the state of the world. In Artificial Intelligence (AI), we are concerned with writing down descriptions of the world in such a way that an intelligent machine can come to new conclusions about its environment by

formally manipulating these descriptions.

This characterization is relevant, since this dissertation looks at knowledge representation from the viewpoint of *interfacing* classical, procedural programs with systems that facilitate, not only the management of representations, but also the process of reasoning from them. This should be seen against what Fikes and Kehler [65] define as the basic criteria for a knowledge representation language, namely *expressive power*, *understandability* and *accessibility*:

1. *Expressive power* measures the possibility and ease of expressing different pieces of knowledge with the system. Since this dissertation largely focuses on W3C's RDF as the underlying knowledge representation formalism (see section 3.2.1) with relatively low expressive power. Those things that might be used to extend this formalism, without necessarily complicating the system in ways that would undermine the two other goals, are of particular interest.

2. *Understandability* measures whether knowledge in a system can be understood by humans. This dissertation will focus largely on *frame-based* representation, an approach that is relatively easy to introduce to – and to be understood by – software practitioners who do not necessarily have any formal background in knowledge representation.

3. *Accessibility* measures how effectively the system can access and use the knowledge contained within. This characteristic is central to this thesis, specifically as it relates to the ease of *interfacing* procedural programs with a knowledge representation subsystem.

One of the key realizations about knowledge representation is that not all formalisms or structures qualify as *representation*; instead, a representation formalism needs to be associated with a *semantic theory* to provide the basis for inference [87].

The early work on knowledge representation focused on human associative memory – in a metaphoric sense – and introduced *associative* formalisms and structures for capturing information about the world. These structures, known as *semantic networks* [178, 206, 27], represent information as labeled graphs where vertices denote *concepts* and arcs denote *relationships* between concepts.

Semantic networks are related to another *structured object* -based approach to knowledge representation, namely *frames* [158, 65, 105, 39]. The simplistic view of frame-based representation is that a *frame* represents an object or a concept. Attached to the frame is a collection of attributes – or *slots* – and these may initially be filled with default values. When a frame is being used, the values of slots can be altered to make the frame correspond to the particular situation at hand. According to an interpretation by Minsky [158], the slots of a frame might represent questions most likely to arise in a hypothetical situation represented by the frame.

Frames, soon after the inception of the idea in the 1970s, were criticized as not introducing anything new to the field of KR; for example, Pat Hayes has said, *"most of 'frames' is just a new syntax for first-order logic"* [88]. Although this statement is certainly easy to accept, it does not diminish the value of frame systems as easy-to-understand tools for simple KR (starting from what might be called "structural modeling"), nor does it exclude a more "formal" approach. An example of a frame-based system that argues both these points is Ontolingua [62]. It provides a frame-based syntax, but then translates all information into KIF,[1] which is just a first-order logic encoding of the information.

The advent of frame systems and semantic networks also led to the early work on *description logics* [11] – as we know them today – with the introduction of KL-ONE [26, 29]. This work began with an emphasis on making term definitions in semantic networks more precise. Description logics provide representation and reasoning languages with precise semantics.

[1] `http://logic.stanford.edu/kif/kif.html`

They also limit language expressiveness so that reasoners can be built that can provide complete (and sound) inference in a tractable manner.

There also exists a connection between frame systems and object-oriented programming (OOP) [95, 117, 140], particularly if we think of the "structural modeling" aspect mentioned above. The basic vocabulary is different, but what the terms denote are approximately the same, as shown in Table 2.1.[2]

OOP Systems	**Frame Systems**	**DLs**
instance	frame, instance, individual	instance, individual
instance variable, slot, attribute	slot	role, attribute
value	filler, value	filler
class, type	frame, schema	class, concept

Table 2.1: Terminological correspondences between OOP, frame systems, and DL

From the adoption viewpoint, it can be observed that many people understand OOP even if they have never heard of frame systems.[3] We can think of frame systems very pragmatically through a "heuristic" interpretation (they are vehicles for storing knowledge and performing inferences) and depart from Minsky's "metaphysical" interpretation.

In comparison to OOP systems, frame systems – as indicated above – typically embody some notion of *reasoning*. Frame system reasoning may sometimes be *incomplete* (i.e., there is no guarantee that everything that could be deduced from a given set of information may be deduced) and frame systems do not typically make guarantees about the computational *tractability* of their inference. Description Logic -based systems typically provide information

[2]Note that in Description Logics the term "attribute" has sometimes been used to distinguish single valued roles from multi-valued roles. Attributes in these systems have a maximum cardinality of 1.

[3]There are some alarming indications, though, that perhaps even some aspects of OOP – such as *inheritance* – may be poorly understood by programmers [22, as an example].

– many times proofs – concerning the tractability of their inference, and if they do not provide complete inference, they typically provide a detailed discussion of what kind of reasoning can be computed (e.g., [24]). They also provide precise semantics – typically denotational semantics – for the meanings of term expressions.

Ontologies have been around for many years. The Merriam Webster dictionary, for example, dates *ontology* circa 1721 and provides two definitions [156]: 1) *A branch of metaphysics concerned with the nature and relations of being*, and 2) *a particular theory about the nature of being or the kinds of existents.* These definitions provide an abstract philosophical notion of ontology. Ontologies have slowly moved into a more mathematical and precise domain, and the notion of a *formal ontology* has existed for over a century: Smith [189] points out that by the year 1900 the philosopher Husserl distinguished them from formal logic [94].

People, as well as artificial agents, typically have a notion or conceptualization of the meaning of terms. Just as the specification of inputs and outputs of a software program could be used as a specification of the program itself, ontologies can be used to provide a concrete specification of term names and meanings. Considering ontologies as specifications of the conceptualizations of terms, there is much room for variation, and the spectrum of Web ontologies typically range from simple controlled vocabularies, through informal concept hierarchies, to something where arbitrarily complex logical relationships can be specified between defined concepts. In practical terms, one would expect the following properties to hold in order to consider something an ontology [140]:

1. Finite controlled (extensible) vocabulary

2. Unambiguous interpretation of classes and term relationships

3. Strict hierarchical subclass relationships between classes

The following properties for ontologies are typical but not mandatory:

4. Property specification on a per-class basis

5. Inclusion of individuals (i.e., instances) in the ontology

6. Value restriction specification on a per-class basis

A widely cited succinct definition of the term *ontology*, representing a departure from the abstract philosophical notion, is Gruber's *"a specification of a conceptualization"* [78]. The *ontological approach* characteristic of the Semantic Web is predicated on the existence and use of ontologies[4] for the purposes of employing mechanisms of reasoning.

2.2 Integrating Applications with Data Representations

The issue of integrating potentially complex, typically external, data representations with application programs is pervasive in software development. In contemporary software development projects, there are frequent needs to query, interpret and manipulate two popular data representations, namely *relational databases* and *XML documents*. Solutions to "binding" data and programs range from high-level architectural approaches – such as OMG's *Model-Driven Architecture* [192] – to source code -level solutions – such as Sun Microsystems' *Java Architecture for XML Binding* (JAXB).[5] Many of these solutions typically take some form of software *generation* from declarative specifications, whether these be UML models or XML schemata. In the former, relatively complete software components can be generated; in the latter, class or function *interfaces* are generated that correspond to particular data declarations.

[4]In practice ontologies are documents or files that formally define the relationships between terms for any particular domain of discourse

[5]`http://java.sun.com/webservices/jaxb/`

3 Semantic Web

> *The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation.*
>
> – T. Berners-Lee, J. Hendler & O. Lassila in "*The Semantic Web*" [21]

The World Wide Web was designed to be *universal*, able to accommodate a wide spectrum of information that varies along many dimensions. One of the shortcomings of the "classical" World Wide Web, however, is that it was built as a means of distributing information that was produced primarily for human consumption, as opposed to this information being consumed and interpreted solely, or at least primarily, by machines (i.e., artificial agents). The *Semantic Web* [21] is an attempt to rectify this shortcoming.

In order for the Semantic Web to function, computers must have access to structured collections of information and sets of *inference rules* that they can use to conduct automated reasoning. This is, obviously, an area where the application of knowledge representation technologies is appropriate. Due to the nature of the World Wide Web, as an open, decentralized, and often *chaotic*[6] environment, traditional knowledge representation approaches may not be directly suitable. These systems have typically been centralized, requiring everyone to share the same definitions of common concepts in their vocabulary of discourse. Central control, however, can be stifling, and increasing the size and scope of such systems quickly becomes quite unmanageable.

Ultimately, systems that are built to take advantage of the Semantic Web must accept that missing or contradictory information, paradoxes and unanswerable questions are a price that must be paid to achieve versatility. We make the language for the rules as expressive

[6] In the colloquial sense of the word.

as needed to allow wide deployment (and acceptance) of the reasoning mechanisms. This philosophy is similar to that of the conventional World Wide Web. Early in the Web's development, critics pointed out that it could never be a well-organized library; without a central database and tree structure, one would never be sure of finding everything. They were of course right, but the expressive power of the system made vast amounts of information available, and search engines (which would have seemed quite impractical earlier) can now produce remarkably complete indices of a lot of the material out there. The challenge of the Semantic Web, therefore, is to provide a language – or as it is turning out, a set of compatible languages – that expresses both *data* and rules for *reasoning about the data*, and that allows rules from any existing knowledge-representation system to be exported onto the Web.

Adding logic to the Web – the means to use rules to make inferences, choose courses of action and answer questions – is the central technical task faced by the Semantic Web community. A mixture of mathematical and engineering decisions complicate this task. The logic must be powerful enough to describe complex properties of objects but not so powerful that agents can be tricked by being asked to consider a paradox. Fortunately, a large majority of the information we want to express is along the lines of "a Yorkshire Terrier is a type of dog," which is readily written in existing languages with a little extra vocabulary.

3.1 Automation and Agents

Making Web content more amenable to automated processing is an important driver of the Semantic Web. This goal is largely related to the observation that much of information technology we use today comes in the form of "tools" that still require human users to do a lot of work. The more of the menial details we can automate, the better humans can focus on matters of importance. The ultimate manifestation of this thinking is the deployment of *autonomous agents*, software systems that operate with a great degree of autonomy and perform tasks for their "owners" – that is, their human users [101, for example].

In the context of this dissertation the term "agent" is defined as a software system that acts autonomously (typically on users' behalf) and maintains some type of discourse with users. Much of agent research has concentrated on the *Beliefs-Desires-Intentions paradigm* (BDI) which structures agents as having a *representational model* of their world (beliefs), *goals* of what results to achieve (desires) and a *plan* of how to get there (intentions).

The word "agent" has often given rise to an incorrect association to omnipotent "secret agents" (like James Bond – this observation is true, for example, in Finnish where the corresponding word "agentti" does not have the generic connotation – *"a person or thing that takes an active role or produces a specified effect"* – as the word does in English). In her invited speech at AAAI'97 [196], Katia Sycara pointed out that agents should rather be thought of like *show-business agents* – they don't necessarily know how to do something, but they know someone who knows. James Hendler believes that the analogy of *travel agents* is a suitable one [91] as it implies a dialogue between the user and his agent(s) throughout the process of solving some particular problem.

Many automated Web-based services already exist without semantics,[7] but other programs, such as software-based agents, have no way to locate services that would perform specific functions. This process, called *service discovery*, can happen only when there is a common language to describe a service in a way that lets other agents "understand" both the function offered and how to take advantage of it. Services and agents can advertise their function by, for example, depositing such descriptions in directories analogous to the Yellow Pages.

Some low-level service-discovery schemes are currently available [179], such as Universal Plug

[7] Actually, there hardly is any such thing as a Web-based service – or any software system for that matter – *without semantics*. Every system has some kind of semantics (at least if the term is used loosely), but with the advent of the Semantic Web the term "semantics" is much overused and abused, often without any clear idea of what it means. In this dissertation, the term is used to indicate a situation where the semantics of a system (or a formalism) are *declarative* and *accessible* to other systems in a *machine-interpretable* form – this may still imply that agreements exist between said systems and/or their designers (such as agreeing on the semantics of RDF).

and Play or UPnP,[8] which focuses on connecting different types of devices, and Jini,[9] which aims to connect services. These initiatives, however, attack the problem at a structural or syntactic level and rely heavily on standardization of a predetermined set of functionality descriptions. Standardization can only go so far, because we can't anticipate all possible future needs, nor should we restrict the set of future interactions to those we can anticipate today.

The Semantic Web, in contrast, is more flexible. The consumer and producer agents can reach a shared understanding by exchanging ontologies, which provide the vocabulary needed for discussion. Agents can even "bootstrap" new reasoning capabilities when they discover new ontologies. Semantics also makes it easier to take advantage of a service that only *partially* matches a request. A typical process will involve the creation of a "value chain" in which subassemblies of information are passed from one agent to another, each one "adding value," to construct the final product requested by the end user. The creation of complicated value chains automatically, on demand, will require some agents to exploit artificial intelligence technologies in combination with the Semantic Web. But the Semantic Web will provide the foundations and the framework to make such technologies more feasible.

From the practical standpoint, in order for the Semantic Web vision to be realised, we need various mechanisms for establishing *trust* between communicating parties (such as agents). *Digital signatures*, encrypted blocks of data that computers and agents can use to verify that the attached information has been provided by a specific trusted source, are one of the prerequisites. You want to be quite sure that a statement sent to your accounting program that you owe money to an online retailer is not a forgery generated by the computer-savvy teenager next door. Agents (as well as humans) should be skeptical of assertions that they read on the Semantic Web until they have checked the sources of information. Generally, this implies a requirement of (potentially sophisticated) *trust models* as well as mechanisms for

[8] `http://www.upnp.org/`
[9] `http://www.jini.org/`

introducing and enforcing various *policies* for access control, privacy, etc. [204, 104, 103, 130].

3.1.1 Meaning of "Meaning"

> *Once we know the number one, we believe that we know the number two, because one plus one equals two. We forget that first we must know the meaning of plus.*
> – from Jean-Luc Godard's movie "Alphaville" (1965)

The Semantic Web is a framework for describing the meaning of concepts and phenomena. Here "meaning" does not necessarily refer to the metaphysical notion – after all, the Semantic Web technologies are really about Computer Science, not about Philosophy. For the Semantic Web, "meaning" is a means for facilitating *automation* by allowing objects of discourse to be *identified*, *described* (in terms of their characteristics) and *related* to other objects.

First, for the Semantic Web to work – as far as the automation aspect is concerned – it is critical to be able to distinguish terms that differ in *meaning*. Human languages thrive when using the same term to mean somewhat different things, but automation does not. Imagine that a business hires a clown messenger service to deliver balloons to its customers on their birthdays. Unfortunately, the service transfers the addresses from a billing database to its database, not knowing that the "addresses" in it are where invoices are sent and that many of them are post office boxes. The hired clowns end up entertaining a number of postal workers – not necessarily a bad thing but certainly not the intended effect. Using a different, unique and non-conflicting *identifier* for each specific concept solves that problem. An address that is a mailing address can be distinguished from one that is a street address, and both can be distinguished from an "address" that is a speech. Another typical, illustrative example would be the term "bridge": are we talking about civil engineering, telecommunications, dentistry, a card game, or perhaps metaphorically (as in "bridging the gap")?

A basic requirement, therefore, for a formalism useful in describing things, is that it can

identify terms and concepts in a unique manner. The initial solution adopted for the Semantic Web is to *name* concepts in a unique manner, using Universal Resource Identifiers or URIs [20].[10] Later – in Chapter 7 – it will be shown that this approach may not be sufficient, as some objects may simply not have been named this way but can still be identified using other means.

The characteristics (attributes) of an object can be described as elementary sentences consisting of a *subject*, a *verb* and an *object*. These sentences, when written in a document, allow this document to make assertions that particular things (people, Web pages or whatever) have properties (such as "is a sister of," "is the author of") with certain values (another person, another Web page). This structure turns out to be a natural way to describe the vast majority of the data processed by machines. The subject and the object of an assertion are each identified by URIs. The verbs are also identified by URIs, thus enabling anyone to define a new concept, a new verb, just by defining a URI for it somewhere on the Web. Naturally the assertions, in addition to stating information about objects, allow objects to be related with one another.

Of course, this is not the end of the story, because two databases may use different identifiers for what is in fact the same concept, such as a zip code. A program that wants to compare or combine information across the two databases has to know that these two terms are being used to mean the same thing. Ideally, the program must have a way to discover such common meanings for whatever databases it encounters. A solution to this problem is provided by the use of ontologies or concept taxonomies.

A taxonomy can define classes of objects and relations among them. For example, an address may be defined as a type of location, and zip codes may be defined to apply only to locations, and so on. Classes, subclasses and relations among entities are a very powerful tool for Web

[10]These are really like the links on a Web page, except that URIs do not necessarily "point" to a dereferencable object. URLs, Uniform Resource Locators, are the most common type of URI.

use. We can express a large number of relations among entities by assigning properties to classes and allowing subclasses to inherit such properties. If zip codes apply to locations such as towns and towns generally have Web sites, we can discuss the Web site associated with a zip code even if no database links a zip code directly to a Web site.

Inference rules in ontologies supply further power. An ontology may express the rule "If a city code is associated with a state code, and an address uses that city code, then that address has the associated state code." A program could then readily deduce, for instance, that a Cornell University address, being in Ithaca, must be in New York State, which is in the U.S., and therefore should be formatted to U.S. standards. The computer doesn't truly "understand" any of this information, but it can now manipulate the terms much more effectively in ways that are useful and meaningful to the human user.

With ontology pages on the Web, solutions to terminology (and other) problems begin to emerge. The meaning of terms, or XML tags, used on a Web page can be defined by pointers from the page to an ontology. Of course, the same problems as before now arise if we point to an ontology that defines addresses as containing a "zip code" and you point to one that uses a "postal code". This kind of confusion can be resolved if ontologies (or other Web services) provide equivalence relations: one or both of our ontologies may contain the information that my zip code is equivalent to your postal code. This information could even be discovered from a third ontology (perhaps provided by a third party).

The scheme for sending in the clowns to entertain customers is partially solved when the two databases point to different definitions of address. The program, using distinct URIs for different concepts of address, will not confuse them and in fact will need to discover that the concepts are related at all. The program could then use a service that takes a list of postal addresses (defined in the first ontology) and convert it into a list of physical addresses (the second ontology) by recognizing and removing post office boxes and other unsuitable addresses. The structure and semantics provided by ontologies make it easier for

an entrepreneur to provide such a service and can make its use completely transparent.

Ontologies can enhance the functioning of the Web in many ways. They can be used in a simple fashion to improve the accuracy of Web searches – the search program could look for only those pages that refer to a precise concept instead of all the ones using ambiguous keywords. More advanced applications could use ontologies to relate the information on a page to the associated knowledge structures and inference rules. For example, a person's home page could provide information that relate the "owner" of the page to his friends, colleagues, employer, projects, publication, etc. in such as way that an automated agent would understand the nature of these relations. Looking at anyone's home page in a browser will often make these relations clear to a human observer; a computer program, however, would have to be very complex to guess that some of this information might be in, say, the person's biography and to understand the English (or other natural) language used there. In fact, work on describing *social networks* using Semantic Web means is a clear step in this direction: schemata (based on Semantic Web formalisms) such as FoaF – "Friend of a Friend" [32, 58] – and CoaC – "Colleague of a Colleague" [138] – enable automated systems to reason about interpersonal as well as organizational relationships.

This type of markup language makes it much easier to develop programs that can tackle complicated questions whose answers do not reside on a single Web page. Suppose you wish to find a certain Ms. Cook you met at a trade conference last year. You don't remember her first name, but you remember that she worked for one of your clients and that her son was a student at your alma mater. An intelligent search program can sift through all the pages of people whose name is "Cook" (sidestepping all the pages relating to cooks, cooking, the Cook Islands, and so forth), find the ones that mention working for a company that's on your list of clients and follow links to Web pages of their children to track down if any attend your old school.[11]

[11]Of course, again, the earlier remark about policies for privacy, access control, etc. applies.

3.2 Semantic Web Formalisms

There are several technologies, mostly various representation formalisms, that make up the basic technological framework for the Semantic Web. The formalisms are layered on one another, the aggregate often being referred to as the "Semantic Web layer-cake"; a more appropriate metaphor would be that of a staircase, since the layers represent increasingly complex "steps" towards the full-blown vision of the Semantic Web. Some of the layers are illustrated in Figure 3.1.

The technology that is often mentioned in connection with the Semantic Web, but in many ways is rather inconsequential, is the *Extensible Markup Language* or XML [31, 30], intended as a flexible means of encoding complex structures. In reality, XML has a tree-like data model [48] and is, as such, not *ideally* suited to encoding other kinds of structures. XML allows the introduction of new *markup tags* and is therefore often confused as being able to introduce "meaning" – in short, XML allows users to add arbitrary structure to their documents but says nothing about what the structures mean.

3.2.1 Resource Description Framework

The *Resource Description Framework* or RDF [122, 124, 141, 13, 113, 33] is the key building block of the Semantic Web. It is layered on XML (effectively, using XML to encode its *syntax*) and introduces a graph-like data model that uses *Universal Resource Identifiers* or URIs [20] to name nodes and arcs. This combination makes RDF not only reminiscent of semantic networks of yore, but also well suited to *shared* representation of information. RDF graphs can span multiple internet hosts, making it possible for one *ontology* or concept taxonomy to easily refer to another one, whether the author of the former has any control over the latter or not. RDF Schema [33] is an associated ontological vocabulary that allows simple ontologies or concept taxonomies to be built using RDF (subsequently, this dissertation will

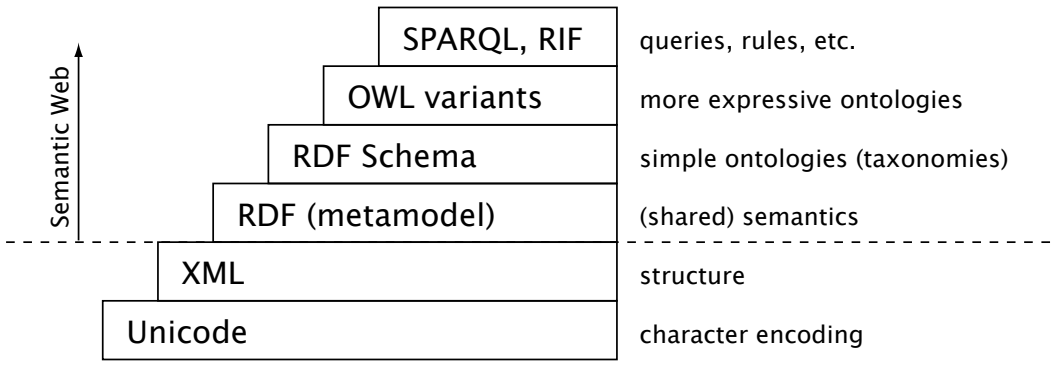

Figure 3.1: Semantic Web "layers"

use the name "RDF(S)" to refer to the representation formalism that uses the RDF data model with the RDF Schema vocabulary).

RDF graphs, for the purposes of defining their semantics or implementing RDF processing, can be thought of as consisting of *subject/predicate/object* -tuples (or "triples"; *subject* and *object* are the endpoints of an arc, and *predicate* names the arc). Triples represent RDF *statements*, asserted facts about (Web) resources. Figure 3.2 shows a fragment of an RDF graph, with the statement $\langle A, P, B \rangle$; note that the node (i.e., "resource") P *names* the arc linking A to B – in a way you could think of P (or the things we know about P) as the embodiment of the meaning of the types of arcs named by P.

The genesis of the W3C Resource Description Framework (RDF) dates back to 1997. W3C's Metadata Activity was an effort to produce a single framework for all the applications which needed to use some type of metadata. Key influences for the design of RDF came from the Web development community itself, in the form of HTML metadata and the Platform for Internet Content Selection (PICS) [157, 115]. Other influences came from the library community, the structured document community (in the form of SGML and, more importantly, XML), and the knowledge representation community. Framework design contributions also came from object-oriented programming and modeling languages, and databases.

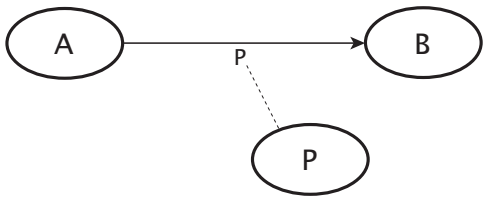

Figure 3.2: Statement $\langle A, P, B \rangle$

Content rating was keenly debated in the standardization community around the time RDF work began. Attempts to balance free speech and protection of minors had resulted in PICS, the W3C's content rating -architecture. This is a metadata mechanism suited to simple content description, but because attribute values could only be chosen from numeric ranges, it had very limited use as a general metadata architecture. PICS did, however, introduce the notion of machine-interpretable schemata for metadata. Figure 3.3 shows a typical PICS label providing information about the content of a Web page.

Attempts to turn PICS into a general metadata mechanism led the W3C to work on "PICS-NG," RDF's predecessor [123]. The original PICS application of content rating ultimately contributed to the charter of W3C's RDF work and the requirements specification of RDF.

Others had also worked on proposals for various frameworks for metadata (and more generally machine-interpretable data) for the Web. These proposals included Netscape's Meta Content Framework [80] and Microsoft's XML-Data [145]. In the summer of 1997, the authors of the various metadata specifications met at MIT and started a joint Web metadata project. Eventually this project got "blessed" by the W3C membership and was chartered as the "RDF Model and Syntax Working Group".

From the beginning, it was obvious that the creators of RDF had to walk a very fine line between simplicity (and thus the ability to deploy) and the expressive power of the formalism. In some sense, RDF had to be at the same time simple enough for the larger Web community to accept and deploy, and "not too offensive" to the knowledge representation

```
(PICS-1.1 "http://www.gcf.org/v2.5"
 by "John Doe"
 labels on "1994.11.05T08:15-0500"
        until "1995.12.31T23:59-0000"
        for "http://w3.org/PICS/Overview.html"
        ratings (suds 0.5 density 0 color/hue 1)
        for "http://w3.org/PICS/Underview.html"
        by "Jane Doe"
        ratings (subject 2 density 1 color/hue 1))
```

Figure 3.3: Typical PICS label

(KR) community to tolerate so that more expressive formalisms could be based on it. The relationship with the KR community was perhaps the more difficult goal, yet it has now been realized with the introduction of the DAML+OIL and OWL ontology languages (see Section 3.2.2).

What are RDF's major benefits? After all, XML offers structured data that could be used to encode and transport attribute/value pairs. In fact, RDF and XML are complementary: RDF defines an object model for metadata, and it only superficially addresses many encoding issues that transportation and file storage require, such as internationalization and character sets. For these issues, RDF relies on XML. But RDF introduces functionality that XML does not have.

One design goal for RDF was to enable metadata authors to specify semantics for data based on XML in a standardized, interoperable manner. RDF also offers features like collection containers and higher-order statements. RDF's main advantage, however, is that it requires metadata authors to designate at least one underlying schema, and that the schemata are sharable and extensible. RDF is based on an object-oriented mindset, and schemata cor-

respond to classes in an object-oriented programming system. Organized in a hierarchy, schemata offer extensibility through subclass refinement. To create a schema slightly different from an existing one only requires that you provide incremental modifications to the base schema. Through schemata sharability, RDF supports the reusability of definitions resulting from the metadata work by individuals and specialized communities.

Due to RDF's incremental extensibility, agents processing metadata will be able to trace the origins of schemata they are unfamiliar with to known schemata. They will be able to perform meaningful actions on metadata they weren't originally designed to process. For example, suppose you were to design an extension to the Dublin Core schema [57] to leverage work done by the library community and also to allow organization-specific document metadata. To do so, you could simply use standard tools designed for plain Dublin Core. Because of the self-describing nature of RDF schemata, a well-designed tool would be able to do meaningful processing for the extended properties as well.

RDF's sharability and extensibility also allow a "mix-and-match" use of metadata and metadata schemata. Metadata authors will be able to use multiple inheritance to provide multiple views to their data, leveraging work done by others. Moreover, it's possible to create RDF instance data based on multiple schemata from multiple sources - that is, interleaving different types of metadata. This will lead to exciting possibilities when agents process metadata. For example, a processing agent may know how to process several types of RDF instances individually, but it will later also be able to reason about the combination. Effectively, the combination is more powerful than the sum of its parts.

From an implementation standpoint, RDF offers a clean, simple object model independent of the transport syntax of metadata. It is also important to remember that although the RDF specification defines a serialization syntax for RDF based on XML, RDF itself is not dependent on XML: it could also use other syntaxes, such as S-expressions (as originally proposed by the author) or the N3 notation [17] now popular with RDF experimenters.

The benefit of adopting XML as the basis for RDF's syntax, although initially driven more by political aims rather than sound technical design goals, is slowly being justified by the realization that "legacy" XML formats can now be transformed using XSLT [43] into RDF (one can think of an XSLT script as embodying the semantics of the source format); this will allow some "Web 2.0" technologies – say, *microformats* [109] – to interface with Semantic Web technologies. A more rigorous effort to provide a framework for format transformations is W3C's GRDDL [89] that allows appropriate XSLT documents to be linked from "legacy" XML data and thus enables Semantic Web agents to automatically transform such data into RDF or OWL.

3.2.2 Other Formalisms

Soon after its introduction, RDF was succeeded by a more expressive knowledge representation language called DAML+OIL [201], developed within the DARPA Agent Markup Language (DAML) research program. It is an ontology language that is based on *description logic* [11]. DAML+OIL served as input to W3C's *Web Ontology Language* OWL [152, 52, 173].

A great deal of work was done to establish formal foundations for the Semantic Web (it should be noted that the original RDF specification [141] did not specify *formal* semantics). KIF [70, 71] was used to specify axiomatic semantics for RDF and DAML+OIL [66]; subsequent work on KIF has lead to the ISO Common Logic effort.[12] Furthermore, a number of other formalisms and languages have emerged, either as part of various research activities or through standards definition processes. These include F-Logic [112], FLORA [207], HiLog [40], ISO Topic Maps [102], SWRL [92], and RIF.[13]

For reasons that are explained later, this dissertation will focus on RDF(S) as the representation formalism of choice, and will discuss how it could be extended to make it ideal for a

[12] `http://cl.tamu.edu/`
[13] `http://www.w3.org/2005/rules/`

broad class of Semantic Web applications.

3.3 Querying Semantic Web Data

Several different query languages have been proposed for Semantic Web representations [83]. Most of the query languages proposed for RDF use a query model based on *relational algebra* [45] and essentially support "RDF-friendly" formulations of relational queries where graphs are viewed as collections of tuples consisting of arc endpoints and the arc label (i.e., these tuples are RDF *statements*). Queries can thus be expressed over a single relation, which, we find, has both benefits and drawbacks. On the benefit side, it can be noted that these queries (and the underlying storage of tuples) can take advantage of ubiquitous, commercial relational database systems; as a drawback, relational queries cannot be formulated that would adequately reflect the *graph nature* of the underlying data – most importantly, the relational algebra cannot express a transitive closure of a relation [5].

Examples of relational query languages for RDF include RDQL [185] and SPARQL [176]. The latter is of particular interest since it was defined by W3C's *Data Access Working Group*, hence there is an expectation that it will become the "officially blessed" query language for RDF. Queries in SPARQL are expressed using a format resembling the SQL `SELECT` statement:

`SELECT` *variables* `WHERE` *condition*

where *condition* has references to (and binds) variables that appear in *variables*. The *condition*, also referred to as the *query pattern*, consists of the following:

- **Triple Patterns**: Patterns of the subject/predicate/object relation that RDF graphs consist of can be specified, where any element of the tuple can either be a constant (a reference to a URI or a literal) or a variable.

- **Constraints**: These are boolean-valued expressions that limit the values of the variables matched by the triple patterns.
- **Groups**: By default, a group of patterns represents a conjunctive condition.
- **Disjunction Operator**: Patterns can combined using a disjunction operator, to represent alternatives.
- **Optional Operator**: The language allows *optional* results to be returned (the failure to satisfy an optional pattern will not make a query fail; successfully satisfying an optional pattern will bind result variables).

Additionally, the language allows queries to be limited to specific (named) graphs.

Not surprisingly, given that RDF representations are *graphs*, the choice of the *relational model* as the basis for SPARQL has inspired lots of critical discussion [10, for example].

SPARQL is ostensibly agnostic with respect to RDF(S) entailments; the suggestion is that SPARQL could be used to query an RDF "triple store" where entailments have already been computed (i.e., treating the *deductive closure* as the graph against which queries are executed). Although this is a reasonable approach to *implementing* a query engine, special care has to be taken to avoid the conflation of the theoretical underpinnings of RDF (namely the *model*) with a practical implementation (as a *data structure*) when the formal semantics of such a query language are defined. It is not altogether clear whether this has been broadly understood in the "SPARQL community" (as an illustrative example, the reader is referred to the mailing list discussion between the author and W3C's Dan Connolly regarding SPARQL "last call" comments[14]).

Some RDF query languages, such as [107, 106] and [150], attempt to combine relational queries with some special knowledge of (class) hierarchies to make the language better suited

[14] `http://lists.w3.org/Archives/Public/public-rdf-dawg-comments/2005Sep/0025.html`

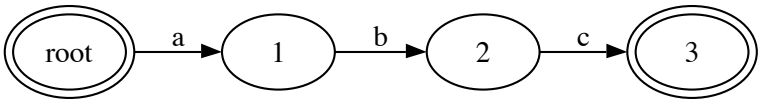

Figure 3.4: Simple graph example

for RDF; related to this is the view that the underlying graph storage system should take care of any reasoning related to class hierarchies so that a query language would not have to, but obviously this approach does not address the issue of how to query *other* hierarchical or repetitive structures.

Several storage solutions for RDF combine *a priori* reasoning – i.e., generation of all possible entailments – with subsequent querying of the generated deductive closure using various query mechanisms that do not have to be aware of the semantic theory for RDF(S). Examples of systems adopting this approach include Sesame [34] and Oracle 10g database server [195].

Query languages for *semi-structured* and *graph-based* data [2, 49, 155, 47] are better suited to querying RDF *as graphs*, given that RDF data often contains repetitive and recursive patterns of relations. These languages, including those defined (and standardized) for XML, such as XPath [44, 76], typically formulate queries as *paths* through a graph. Path query languages have also been developed for RDF; examples include Versa [168] and – to some extent – the aforementioned RQL. Since paths of fixed length can also be expressed using relational queries, *true* path languages can be distinguished by their ability to express repetition (transitive closure); obviously there is also a "convenience factor", since if a query language's syntax does not naturally support paths, they can be very cumbersome to express and result in complex conditional expressions. For example, given the simple graph in Figure 3.4, a path from `root` to `3`, consisting of a sequence of three arcs `a`, `b`, and `c`, would result in the following SQL query:

```
SELECT t3.o FROM t t1, t t2, t t3
WHERE t1.s = root AND t1.p = a
  AND t1.o = t2.s AND t2.p = b
  AND t2.o = t3.s AND t3.p = c
```

where the graph is assumed to be stored in a table called `t` with columns `s` (for *subject*), `p` (for *predicate*), and `o` (for *object*).

Some query languages for RDF have also been proposed that are essentially rule-based, or based on the execution of a logic program [81, 18, 188]; these languages offer expressive power beyond the relational algebra. Furthermore, query languages have been conceived for the more expressive Semantic Web languages, such as the DAML Query Language [64] for DAML+OIL and OWL-QL [63] for OWL; these languages, effectively, allow deductions in a knowledge base that uses one of the aforementioned representation formalisms.

Formal foundations for querying Semantic Web formalisms (and RDF specifically) are studied in [82] where the authors observe that challenges are introduced by blank nodes, reification and the treatment of those parts of RDF vocabulary that have a predefined semantic theory (they also point out that existing work on RDF query languages does not, typically, have sound formal basis but mostly consists of *ad hoc* query formalisms). Treatment of "anonymous" resources such as blank nodes (and potentially reified statements) is further elaborated in [208, 209].

It should further be noted that even outside the domain of the Semantic Web and RDF, there are many real-world applications requiring data representations that are inherently recursive, but particularly, applications involving artificial intelligence and KR have this quality. The relational algebra has, on several occasions, been extended to accommodate more complex representations and queries – hierarchies, transitive closure, etc. [3, 100].

3.4 Service-Oriented Computing and Web Services

Service-oriented computing is an emerging paradigm in information systems, and has received *a lot* of attention during the last several years [172]. In order to realize and implement service-oriented systems, the services themselves have to be "grounded" in practical, concrete technologies. At the core of these technologies are "Web services" – functionality that can be invoked remotely over the Web; they represent a strong recent trend in the development of distributed information systems. Several industry standards have emerged in this area, including SOAP [25], an invocation protocol, WSDL [42], a formalism for describing the interfaces of Web Services, and a multitude of other specifications. All these specifications are being offered with the promise of greater opportunity for *automation* of tasks and improved *interoperability* of information systems.

Albeit one could argue that Web services represent progress in the right direction, the Web services architecture falls short of the goals of improved automation and interoperability, because it is based on heavy *a priori* standardization and ultimately retains humans in the loop. Each Web service interface – whether it be expressed in WSDL or in some other way – represents its own small vocabulary. The maintenance and management of all the emerging vocabularies will result in a phenomenon that can only be compared to the biblical story of the "Tower of Babble" [73]. Our true goal should be something like "serendipitous interoperability", the ability of software systems to discover and utilize services they have not seen before, and that were not considered when these systems were designed [126].

3.4.1 Semantic Web Services

To mitigate some of the perceived shortcomings of the current Web service technologies, qualitatively stronger means of representing the service semantics are required, enabling fully automated discovery and invocation, and complete removal of unnecessary interaction

with human users. The Semantic Web offers means of advancing beyond the current proposed architectures for Web services by allowing information systems to *reason* about data sources and the functionality of other systems, and consequently allowing them to better take advantage of these.

In the context of Web Services, the application of the Semantic Web to representing information and its semantics will enable the following:

- Description of semantics of services to allow their automatic discovery, even if the services offered only partially match the needs of the requester.

- Automatic composition of multiple services – possibly partially matching ones – into a "super-service" satisfying the needs of the requester (whether the requester be a human or an artificial agent).

Some of the Semantic Web's original motivations were to increase automation in processing Web-based information and to improve the interoperability of Web-based information systems. The development of representational issues and logical frameworks (such as OWL) will take us only so far; to fully realize this vision, also behavioral issues must be tackled (for example, interactions between "SemanticWeb agents"). Serendipitous interoperability – that is, the unarchitected, unanticipated encounters of agents on the Web is an important component of this realization.

Semantic Web techniques, which consist of applying knowledge representation techniques in a distributed environment (potentially on a Web-wide scale), have proven useful in providing richer descriptions of Web resources. Semantic Web Services [174], as a new research paradigm, is generally defined as the augmentation of Web Service descriptions through Semantic Web annotations, to facilitate the higher automation of service discovery, composition, invocation, and monitoring in an open, unregulated, and often chaotic[15] environment

[15] Again, in the colloquial sense of the term.

(that is, the Web). Several research and "pre-standardization" activities in Semantic Web Services have emerged, the best known perhaps being the DAML-S/OWL-S work [7, 8, 149], developed within the DAML research program, and WSMO [181, 51], developed within the European Semantic Systems Initiative (ESSI) research program. Semantic Web Services represent an important step toward the full-blown vision of the Semantic Web, in terms of utilizing, managing, and creating semantic markup.

The relationship between the Semantic Web and the current Web Service architecture depends on one's viewpoint. In the near term, the deployment of Web Services is critical, and Semantic Web techniques can enhance the current service architecture. In the longer term, the Semantic Web vision itself will dominate, with Web Services offering a (hopefully) ubiquitous infrastructure on which to build the next generation of deployed multiagent systems.

3.4.2 Agent-Based Systems

As discussed in Section 3.1, central to the Semantic Web vision – in its "full-blown" manifestation – is the emergence of *autonomous agents* that can perform tasks *on behalf of* their human users. In a way, agent-based systems represent an "extreme" manifestation of the service-oriented architecture.

From the Semantic Web viewpoint agents are interesting because the Semantic Web, when seen as a facilitator of interoperability and automation, is an enabling technology for agent deployment. In this regard, the representational model an agent uses is of particular interest, as well as the agent-to-agent communication that it enables.

3.5 Semantic Web Use Case: Ubiquitous Computing

One can imagine that the Semantic Web will be breaking out of the "virtual" realm of the World Wide Web and extending into our physical world. URIs can point to anything, including physical entities, which means the Semantic Web languages can be used to describe devices such as cell phones or home appliances. Such devices can advertise their functionality – what they can do and how they are controlled – much like software agents. Being much more flexible than low-level schemes such as UPnP, such a semantic approach opens up a world of exciting possibilities.

For instance, what today is called *home automation* requires careful configuration for appliances to work together. Semantic descriptions of device capabilities and functionality will let us achieve such automation with minimal human intervention. A trivial example occurs when a user answers his phone and the stereo sound is turned down. Instead of having to program each specific appliance, he could program such a function once and for all to cover every local device that advertises having a volume control – the TV, the DVD player and even the media players on the laptop computer that he may have brought home from work this one evening.

Standards have now emerged for describing functional capabilities of devices (such as screen sizes) and user preferences. Built on RDF, these standards are the W3C Composite Capability/Preference Profile or CC/PP [114] and its derivative, the OMA User Agent Profile or UAProf [169]. Initially, they allow cell phones and other "non-standard" Web clients to describe their characteristics so that Web content can be tailored for them on the fly. With the addition of mechanisms for describing *functionality* – OWL-S [7, 8, 149], for example (see section 3.4.1) – and with the full versatility of languages for handling ontologies and logic, devices could automatically seek out and employ services and other devices for added information or functionality. It is not hard to imagine a Semantic Web -enabled microwave oven consulting a frozen-food manufacturer's Web site for optimal cooking parameters.

3.6 A Philosophical Note

Finally – to reflect on the overall vision presented in [21] – it should be noted that there is an aspect of the Semantic Web that reaches beyond the use of the associated technologies as a means of developing software applications. With the risk of sounding melodramatic it could be argued that the Semantic Web can support the *evolution of human knowledge* as a whole.

Human endeavor is frequently caught in a tension between the effectiveness of small groups acting independently and the need to integrate their efforts and achievements with the wider community. A small group can innovate rapidly and efficiently, but often in doing so produces a subculture whose concepts may not be understood by others. Conversely, coordinating actions across a large group can be painfully slow and take an enormous amount of communication. The world works across the spectrum between these extremes, with a tendency to start small – from a personal idea – and move toward a wider understanding over time.

An essential process is the joining together of subcultures when a wider common language is needed. Often two groups independently develop very similar concepts, and describing the relation between them brings great benefits. Like a Finnish-English dictionary, or a units conversion table, the relations allow communication and collaboration even when the commonality of concepts has not yet led to a commonality of terms. The Semantic Web, predominantly via naming every concept by a URI, lets anyone express new concepts that they invent with minimal effort.[16] Its unifying logical language will enable these concepts to be progressively linked with one another. This structure will open up the knowledge and workings of humankind to meaningful analysis by software agents, providing a new class of tools by which we can live, work and learn together.

[16]Later, however, it will be demonstrated that this naming mechanism may be somewhat idealistic and may require to be supported by other means of establishing "identity" of concepts.

4 Challenges in Building Semantic Web Applications

Semantic Web is – in some ways – a problematic set of technologies: Any *specific* problem that is presented as one lending itself well to be solved using Semantic Web technologies, will typically also have a *specific* solution using some alternate ("conventional") technologies. This aspect of the Semantic Web makes its marketing difficult; consequently, building applications that truly benefit from Semantic Web technologies may not be straightforward. The fact that Semantic Web technologies are not a solution for specific, *known* problems suggests that they are a solution, instead, for those problems that are yet to be (fully) defined. Semantic Web, therefore, exhibits a strong flavor of *serendipity*, and applications taking advantage of Semantic Web technologies should also be able to behave in serendipitous ways. In the desire to move towards systems that do more *on behalf of* their users (rather than merely *facilitating* the users to do certain things), the ability for automated systems to behave with some *autonomy* and to exhibit "sensible" behavior in unanticipated situations is paramount.

It could be argued [126] that the Semantic Web is a means for achieving interoperability (of services, of information sources, etc.), a means qualitatively stronger than the traditional *a priori* standardization -based approach. With Semantic Web technologies, we predominantly do not standardize *what* will be said (in a "dialogue" between two information systems, for example); instead, we standardize *how* to say things. "Meaning" in an inter-system dialogue will then arise from the use of representations that enable semantics to be expressed and shared.

Motivating this dissertation is the desire to address some of the concrete software issues arising from the fairly abstract description above. Specifically, the following issues will be investigated:

- How to handle complex, declarative representations; how to "bind" procedural code with declarative data.

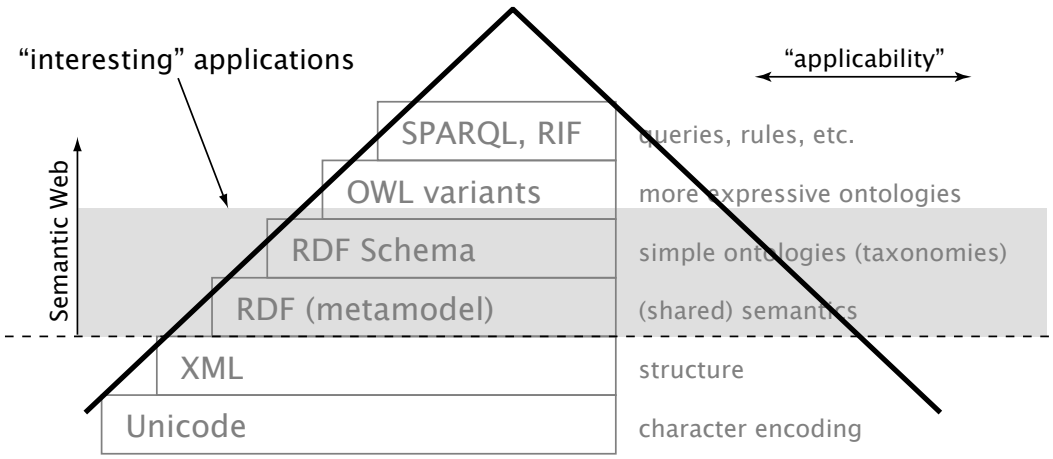

Figure 4.1: Requirement of expressive power in applications discussed in this dissertation

- How to build complex, declarative representations, particularly vis-à-vis the management of identity of objects from multiple sources.
- How to best handle reasoning as part of a software system.
- How to build systems that exhibit a degree of serendipity in their behavior.

Recalling Figure 3.1 illustrating the various representational layers of the Semantic Web, a simple hypothesis is adopted: Lower layers are applicable (and useful to) a larger number of applications than the upper layers; in other words, there will be more applications that can be implemented by using, say, RDF(S) than applications that will require, say, the more expressive variants of OWL. This is a "common sense" hypothesis, and no attempt is made to validate it. It should also be observed that the layers, as presented in Figure 3.1, do not strictly correspond to a continuum of increasing expressive power, and should therefore be considered "metaphorical". The hypothesis, and the class of applications considered "interesting" from the standpoint of this dissertation, is illustrated in Figure 4.1. In a way, this thinking is an acknowledgment of the principle "a little semantics goes a long way."[17]

[17]This principle is often called the "Hendler Hypothesis"; it was once a slogan of the SHOE project [90].

4.1 Using Complex Declarative Representations

The usage of complex, declarative data representations in software systems is by and large an issue of *access* to this data. How does one query underlying representation(s) and "connect" the queries with the (procedural) code that makes use of the results of these queries? The mechanisms for this connection have to be simple and natural to the programming language used. The design of said software systems also benefits from various aspects of "data housekeeping" being taken care of by the underlying knowledge representation middleware. For example, *reasoning*, hidden from the application program, can be used to perform duties that otherwise would result in complex procedural code. Naturally the properties of the representation language affect the ability to introduce (simple) reasoning and whether this reasoning can be hidden.

4.1.1 "Identity Crisis" in RDF

It has already been established that RDF(S) can be seen as a simple "Web-compatible" frame-based representation system. Essentially, representations are formed from frames referring to other frames; integration of representations from multiple sources is not possible without establishing the *identity* of the frames; RDF relies on URIs for object identity. W3C's "Architecture of the World Wide Web" [99, Section 2.] says:

> A resource should have an associated URI if another party might reasonably want to create a hypertext link to it, make or refute assertions about it, retrieve or cache a representation of it, include all or part of it by reference into another representation, annotate it, or perform other operations on it.

This principle is difficult to comply with when describing "real world" phenomena that typically do not have URIs; nor do conventions for predictably establishing URIs exist. Ways in which RDF(S) could be extended to mitigate this impending "identity crisis" are introduced

later in this dissertation.

4.2 Using Reasoning

It could be argued that in order for Semantic Web technologies to become part of the "mainstream" application development culture, they have to be presented in a way that is comprehensible by application developers. *Reasoning* is a mechanism that is largely not understood or even known by most software developers. Not only does a normal computer science or software engineering curriculum not expose students to these mechanisms, but the mere nature of reasoning (the fact that one does not really operate in terms of an "explicit" data structure but more with something derived from any concrete data the system may possess) may make this difficult. Naturally one could treat a reasoner the way most software frameworks treat database engines – an external entity with which there is a specific means of communication – but as can be observed from these frameworks, this may lead to further separation of multiple representations of the same information (e.g., program internal data structures vs. queries vs. query results vs. database schemata, etc.).

Depending on what type of semantic theory is associated with the representation used by an application, it may be possible to largely "decouple" reasoning from other aspects of the application logic. Later, an approach where reasoning is mostly hidden from the software developer will be pursued.

4.3 Implementing Serendipity

How to build software with emergent qualities and the capability of "doing the right thing" in unanticipated situations may in itself be worthy of multiple dissertations. To a large degree this is what the *multiagent paradigm* is about and what the corresponding research com-

munity has long grappled with (see [101, for example] and Section 3.1). Without disputing the progress that the agent research community has made, one must observe that by and large these technologies have failed to become part of the mainstream software development arsenal. It may therefore be warranted to ask whether there could be "simpler" ways in approaching serendipity.

5 Exposing Representation to Application Logic

Software developers are constantly faced with need to represent the same data in many different physical or conceptual formats or models. Typical examples include mapping between persistent storage (databases) and an application's internal (in-memory) representations, "marshalling" between the external wire formats of various communications protocols and the application's internal representations, and converting data received through user interfaces into structures more amenable to programmatic manipulation.

Knowledge-based applications typically need to make use of very sophisticated representations of information, and again, almost unavoidably there is a separation of the (declarative) representation from the application's (procedural) logic.

Having to deal with multiple representations of the same information may – and often does – lead to mappings that are created and maintained manually. These mappings are error-prone and by and large cause "clutter" in the concrete implementation. This clutter distracts the developer from the problem she is trying to solve.

5.1 Traditional Approaches to Data Interfaces

A way of interfacing complex representations with software systems is a mechanism where the applications poses *queries* to an external system that then responds with a set of *results* that, in turn, have to be "translated" by the querying application. This architecture is typical with systems making use of relational databases – often implemented with standardized interfaces such as ODBC and JDBC – as well as systems interfacing with knowledge bases – with, again, using standard interfaces such as OKBC and DIG.[18]

[18] `http://www-ksl.stanford.edu/software/OKBC/` and `http://dig.sourceforge.net/`

In existing libraries and toolkits for RDF(S) data are exposed to an application program via some *Application Programming Interface* (API). These APIs are not unlike the query interfaces discussed above. All RDF(S) libraries and toolkits offer an API – typically their own – through which the underlying data is accessible as *triples*. Good APIs hide details of graph storage and allow the isolation of application logic from the storage substrate – examples of such APIs include the Java-based *Jena* [151] and the C-based *Redland* [15]. An example of an API that fails in this regard is the programming interface for the Oracle RDF Data Model [162] which exposes details of the underlying relational storage of the RDF graphs.

5.2 Procedural Attachment in Frame-based Systems

In frame-based knowledge representation systems one approach to solving the "representation vs. program" -problem was to provide *procedural attachment* by either associating (procedural) behavior with various parts of the representation – typically as object-oriented "methods" – or with various transactional events that modified the representation; the latter is generally known as *access-oriented programming* [194]. Many attempts to do procedural attachment are quite clumsy, though, where the mechanisms of the programming language are largely separate from the mechanisms of the frame-based representation system.

Attempts to truly amalgamate frame-based knowlege representation and object-oriented programming, despite the known parallels of the two [105, 117, 140], are rare. The approach to the integration of knowledge representation and programming presented here is in part motivated by the body of work on *procedural attachment* in frame systems [105, Chapter 8], and has to be understood in the context of the succession of frame-based representation systems developed by the author, systems that have taken various different approaches to procedural attachment and more generally program/representation integration. The "genealogy" of

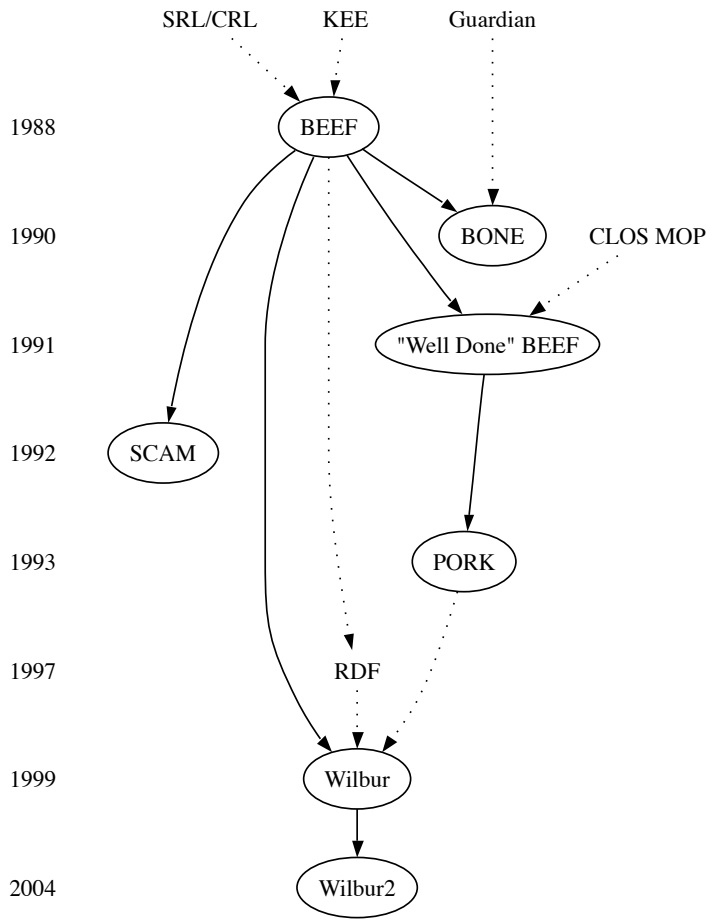

Figure 5.1: Genealogy of author's frame-based representation systems

these systems is illustrated in Figure 5.1:[19]

BEEF: The frame system BEEF [95, 118, 144] addressed the issue of integration of programming and representation by adding features found in object-oriented programming languages into a frame-based representation system. As a KR system, BEEF was strongly influenced by SRL/CRL [68, 69, 38] and KEE [65, 97], but went further in its implementation of procedural attachment. BEEF was implemented in Common

[19]In this diagram solid lines indicate the sharing of source code, and dotted lines indicate "influence".

Lisp [193] but predated the practical availability of the Common Lisp Object System (CLOS). It introduced its own method definition and invocation mechanism that was natural to use in Common Lisp programs yet seamlessly "glued" onto the frame-based representation. Since it implemented typical slot *daemons* as methods as well, one could argue that it provided a synthesis of object- and access-oriented procedural attachment approaches.

BONE: In an approach to building *distributed* knowledge-based applications, BONE [143, 144], as a version of BEEF, introduced a Guardian-like [147] framework for distributing representation frames yet allowing them to communicate.

SCAM: Regrettably undocumented, the SCAM frame system was a simplified version of BEEF originally built to facilitate the porting of software that depended on the CRL frame language [38]. SCAM served as the knowledge representation substrate for the automated planner component of the "Remote Agent" that flew with NASA's "Deep Space 1" probe past the Asteroid Belt in 1999 [163]. It is a demonstration that frame-based representation systems can be designed for extremely stringent efficiency and reliability requirements.

"Well-Done" BEEF and PORK: BEEF was built before the practical availability of CLOS, and ideas of reimplementing BEEF on top of CLOS emerged in the form of "Well-Done" BEEF [119], but were not realized until the inception of the PORK system [121]. PORK added "frame-like" representation features to an object-oriented programming language; in this, the approach it took was "inverted" from the one taken with BEEF. In practice, PORK was a frame system implemented as a metaobject extension [111] of CLOS. As such, it extended and modified the inner workings of the underlying object-oriented programming language to offer frame-based features.

Wilbur and Wilbur2: In order to test ideas during the design of RDF, source code from

BEEF was adapted to implement a toolkit for RDF programming. Most importantly, WILBUR extends the BEEF path query language to implement flexible low-level integration of RDF data with Common Lisp programs. WILBUR has since been re-engineered as WILBUR2, with an implementation of a reasoner for RDF. The framework for integration of declarative representation and procedural programs, presented in this dissertation, has been developed in the context of – and has its concrete manifestation in – WILBUR and WILBUR2. These systems are described in more detail in Chapter 8.

When BEEF and PORK were used to implement large knowledge-based software systems [95, 199, 142, 190, 136, 191] they served as the principal object-oriented programming languages for these systems.

The "near-seamless" integration of frame-based representation and object-oriented programming, as implemented in BEEF and PORK, was possible because the representation language semantics could be adjusted to correspond to the semantics of the object-oriented language. RDF(S) can be seen as an object-oriented type system,[20] but since its semantics is fixed – by [86] – the question about the possibility of adjusting the *programming language semantics* is raised. A common approach to programming Semantic Web applications (and many other applications utilizing KR) is to either treat RDF (say) as not having polymorphic object-oriented qualities at all (to merely look at it as a graph, for example) or to implement RDF's object system in a language that may also support its own "native" object system; in the latter case the two object systems remain separate, and it is not possible to use native language features (e.g., method dispatching and invocation) with the representation system's object system. In other words, RDF as the knowledge representation (KR) system is typically treated as an *application* of the underlying implementation language, forcing applications to mix application logic with the manipulation of the KR formalism, as opposed to the KR

[20]The term *object system* is used to denote an object-oriented, polymorphic type system – often part of an object-oriented programming language.

formalism being integrated as *part of* the implementation language itself.

Albeit the idea of treating RDF(S) as the object system of a programming language, as suggested above, would be an attractive approach to building RDF-based software applications, the inevitable differences in semantics between RDF(S) (given that its semantics is fixed) and the semantics of some object-oriented programming language present difficulties that are hard to overcome. Possible solutions include definition of a new "RDF-compatible" programming language – as suggested in [72] – or to provide a flexible means of querying and accessing the underlying representation – such as the extension to C++ presented in [148]. Some programming languages also allow altering their semantics to certain degree, via the use of a *metaobject protocol* [111], and languages that don't have sometimes been "retrofitted" with one [41, 180, for example]. The surreptitious augmentation of an object's type in RDF(S) – via the assignment of the object as the object (subject) of a statement whose predicate has a range (domain) definition – presents a problematic situation from the programming language viewpoint. In CLOS, as an example, altering the graph that constitutes a program's data representation would require recomputation of *class precedence lists*. This could happen in the middle of method invocation and alter the method combination of that particular invocation; thus it is infeasible to write non-trivial "real-life" programs under such conditions.[21]

Given that fully seamless integration is not possible, the next best alternative is to expose the representation to the program in a way that is as "effortless" as possible from the programming language viewpoint. This is the approach pursued here, with additional considerations with respect to reasoning mechanisms.

[21]The situation might be different had the RDFCore Working Group of W3C not decided to alter the treatment of domain and range from *restrictive* constraints to generative ones.

5.3 A Path Query Language

In the approach taken in this dissertation, it is proposed that RDF graphs can be exposed through a node-centric (i.e., "frame system") API. Central to this API is a *slot access* function $A_{lookup}(f, s, \mathcal{G})$: "given graph $\mathcal{G}$, give me the values of slot s of frame f". Since this is in the context of RDF, *resources* are frames and *properties* are slots. This type of access function is typical of many APIs for frame-based representation systems [38, 39, for example]. This basic API can be extended by supporting a query language which allows complex access paths – expressed as regular expressions of *slot names* (i.e., RDF properties) – to be used in place of atomic slot names. The query language proposed is an extension of the query mechanism of the BEEF frame system [95] which, in turn, is an efficient implementation of (a simplification of) the CRL/SRL path language [69]. It resembles query languages constructed for semi-structured and graph-based data [2, 49, 155, 47, for example].

Path expressions can take the following forms, expressed here in an abstract syntax:

- **Sequence** (*concatenation* in [155]): $seq(e_1, \ldots, e_n)$ matches a sequence of n steps in the graph, consisting of subexpressions $e_1, \ldots, e_n$; $seq^+(e_1, \ldots, e_n)$ matches any one of the expressions $seq(e_1, \ldots, e_i)$ where $i \in [1 \ldots n]$ (the expressions are matched in order, with shortest sequence first); in effect,

$$seq^+(e_1, \ldots, e_n) \equiv or(e_1, seq(e_1, e_2), \ldots, seq(e_1, \ldots, e_n)) \tag{5.1}$$

- **Disjunction** (*alternation*): $or(e_1, \ldots, e_n)$ matches any one of n subexpressions $e_1, \ldots, e_n$. The subexpressions are matched in the order they are specified.

- **Repetition** (*closure*): $rep(e)$ matches the transitive closure of subexpression e; furthermore

$$rep^+(e) \equiv seq(e, rep(e)) \tag{5.2}$$

- **Inverse**: Satisfaction of $inv(e)$ requires the path defined by the subexpression e to be matched in reverse direction – this is similar to the *inversion* operator of GraphLog [47].

- **Value**: $value(e)$ causes the value e to be generated in the matching process, ignoring any actual slot accesses. It is useful in specifying *default values*, typically using the idiom

$$or(path, value(default)) \tag{5.3}$$

- **Filters**: $filter(s)$ matches only those literals (and URIs of nodes) that contain the substring s; similarly, $lang(s)$ matches only those literals whose `xml:lang` attribute matches s. Furthermore, $restrict(v)$ matches only the value v. For example, the query expression

$$seq(e_1, restrict(e_2), e_3)$$

  matches arc labels (properties) e_1 and e_3 in sequence, but only if the *node* after the step e_1 is e_2.

- **Current node**: The query language supports the token $\perp$ (pronounced "self") that can be used as a valid query expression; it matches (and generates the value of) the current node in the matching process, ignoring any slot accesses. It is typically used in an idiom like

$$or(\perp, seq(\ldots)) \tag{5.4}$$

  as part of a sequence where *some* intermediary terminals are possible (but not every step in the sequence).

- **Wildcards**: The query language supports wildcards that match either *any* arc label or just RDF *container membership* properties. In the following discussion the symbols $*_{any}$ and $*_{member}$ will be used for these wildcards.

Given a *root node* (i.e., a search start point) and a *query expression*, a data/program integration API should provide functions for retrieving the first reachable node, for retrieving all reachable nodes, and for determining whether a path exists between two specified nodes. It should also allow for multiple queries to be combined using a set algebra by supporting the operators *union*, *intersection* and *difference*. Of these, the *intersection* operator is of particular interest, since it can be used to implement queries with conjunctive conditions.

5.3.1 Implementing the Path Query Language

One possible implementation strategy for the query language is to transform query expressions into optimized deterministic finite state automata [4, section 3.9] and use these to – effectively – "walk" the underlying RDF graphs (which are stored as RDF triples in main-memory databases with hashed indices). During traversal, graph nodes are marked with DFA states as in [155, section 5] but without restricting the system to simple paths – by marking the nodes with *all* applicable states WILBUR is able to find the correct answer to the problem presented in [155, example 8].[22]

Before the transformation into finite state automata, query expressions are normalized; essentially, operators that are considered "syntactic sugar" are expressed using a minimal set of operators (*seq*, *rep*, and *or*) by the application of the following simple transformation rules to exhaustion:

1. Unary uses of n-ary operators (*seq*, seq^+, and *or*) are reduced to their single operand:

$$op(e) \rightarrow e \tag{5.5}$$

2. Uses of n-ary operators (for $n > 2$) are reduced to binary versions of the same:

$$op(e_1, \ldots, e_n) \rightarrow op(e_1, op(e_2, \ldots, e_n)) \tag{5.6}$$

[22] In this example, an "obvious" result is not found because the algorithm marks a potentially terminal node with a non-terminal state of the DFA before visiting a terminal state.

3. Occurrences of rep^+ are removed:

$$rep^+(e) \rightarrow seq(e, rep(e)) \tag{5.7}$$

4. Occurrences of seq^+ are removed (note that this operator is equally easy to implement by merely marking additional states as terminals during the construction of the DFA corresponding to the query expression – in fact, in the concrete implementation described in Chapter 8 this is the approach taken):

$$seq^+(e_1, e_2) \rightarrow or(e_1, seq(e_1, e_2)) \tag{5.8}$$

5. Uses of inv are "pushed" to the leaves of the parse tree of the query expression (so that, effectively, individual arcs are inverted), using the following transformations:

$$inv(rep(e)) \rightarrow rep(inv(e)) \tag{5.9}$$

$$inv(or(e_1, e_2)) \rightarrow or(inv(e_2), inv(e_1)) \tag{5.10}$$

$$inv(seq(e_1, e_2)) \rightarrow seq(inv(e_2), inv(e_1)) \tag{5.11}$$

   This largely corresponds to the definition of inv as given in [37, section 4]. Note that expressions of the form $inv(a)$ where a is an atomic symbol are replaced with special tokens that can be interpreted directly by the query engine (using the notation from [37, section 4] these tokens would be written as a^-).

6. Similarly, uses of the various filter operators ($filter$, $restrict$) are replaced with special tokens interpreted in a special way by the query engine.

7. Subexpressions of type $inv(value(e))$ are simply removed, but later it will be demonstrated how – during the calculation of query results – the inverses of default values can be dealt with using *lazy evaluation*.

The query (and storage) engine can be implemented using the following functions:

Query Interface: $A_{lookup}(n, q, \mathcal{G})$ returns all nodes from the graph $\mathcal{G}$ reachable from node n using the path q:

```
1   A_lookup(n, q, G)
2       N ← {}, S ← {}
3       call walk(n, makedfa(q), 0, G)
4       return N
5   end A_lookup
```
(5.12)

where $walk(node, dfa, i, \mathcal{G})$ is defined as

```
1   walk(node, dfa, i, G)
2       if ⟨i, node⟩ ∉ S then
3           S ← S ∪ {⟨i, node⟩}
4           state ← dfa[i]
5           if is_terminal(state) then
6               N ← N ∪ {node}
7           end if
8           for ⟨input, j⟩ ∈ state do
9               for node' ∈ expand(node, input, G) do
10                  call walk(node', dfa, j, G)
11              end for
12          end for
13      end if
14  end walk
```
(5.13)

Note that in this definition i is the (index of a) state of the DFA dfa, $\mathcal{S}$ is the set of state/node pairs already encountered, and $makedfa(q)$ is a function constructing the DFA corresponding

to the query expression q [4, algorithm 3.5]. A DFA, in this case, is a vector of *states*, each of which is a set of pairs $\langle input, index \rangle$ where *input* is any edge label of the graph queried (more specifically, any edge parameter e of the function $expand(v, e, \mathcal{G})$, and *index* is a state index of the DFA. All states also record whether they are *terminal* or not.

Query Step Expansion: $expand(n, a, \mathcal{G})$ is used by the DFA walker to expand a query from node n via arc a (i.e., given n, and a as the next transition, this function returns the set of "next" nodes, if any).[23] The function *expand* is defined for the different types of "query atoms" (ordinary arcs a, inverse arcs $inv(a)$, value insertions $value(e)$, universal wildcards '$*_{any}$', container membership wildcards '$*_{member}$', and the "current node" token $\perp$) as follows:

$$expand(n, a, \mathcal{G}) = \{o \mid \langle s, p, o \rangle \in triple(n, a, *, \mathcal{G})\} \tag{5.14}$$

$$expand(n, inv(a), \mathcal{G}) = \{s \mid \langle s, p, o \rangle \in triple(*, a, n, \mathcal{G})\} \tag{5.15}$$

$$expand(n, value(e), \mathcal{G}) = \{e\} \tag{5.16}$$

$$expand(n, restrict(e), \mathcal{G}) = \begin{cases} \{n\}, \text{ if } n = e \\ \{\}, \text{ otherwise} \end{cases} \tag{5.17}$$

$$expand(n, filter(s), \mathcal{G}) = \begin{cases} \{n\}, \text{ if the URI of } n \text{ contains substring } s \\ \{\}, \text{ otherwise} \end{cases} \tag{5.18}$$

$$expand(n, \perp, \mathcal{G}) = \{n\} \tag{5.19}$$

$$expand(n, *_{any}, \mathcal{G}) = \{o \mid \langle s, p, o \rangle \in triple(n, *, *, \mathcal{G})\} \tag{5.20}$$

$$expand(n, *_{member}, \mathcal{G}) = \{o \mid \langle s, p, o \rangle \in triple(n, *, *, \mathcal{G}) \wedge p \in \mathcal{P}_{member}\} \tag{5.21}$$

where $\mathcal{P}_{member}$ is the (potentially infinite) set of container membership properties (`rdf:_1`,

[23] Not to be confused with *expand* in [37, section 4] where it denotes the rewriting of query expressions.

`rdf:_2`, `rdf:_3`, etc.). Furthermore, there are special cases for literals λ:

$$expand(\lambda, \texttt{rdf:type}, \mathcal{G}) = \begin{cases} \{\texttt{rdf:XMLLiteral}\}, \text{ if } \lambda \text{ is a well-typed XML literal} \\ \{\texttt{rdfs:Literal}\}, \text{ if } \lambda \text{ is any other kind of literal} \end{cases} \tag{5.22}$$

$$expand(\lambda, filter(s), \mathcal{G}) = \begin{cases} \{\lambda\}, \text{ if } \lambda \text{ contains the substring } s \\ \{\}, \text{ otherwise} \end{cases} \tag{5.23}$$

$$expand(\lambda, lang(s), \mathcal{G}) = \begin{cases} \{\lambda\}, \text{ if the } \texttt{xml:lang} \text{ attribute of } \lambda \text{ matches } s \\ \{\}, \text{ otherwise} \end{cases} \tag{5.24}$$

Graph Storage Access: $triple(s, p, o, \mathcal{G})$ returns all matching triples from the graph $\mathcal{G}$. The values of s, p, and o can either be constants (a node in the graph, or a literal value) or wildcards '$*$'. The function can be defined as follows:

$$triple(s, p, o, \mathcal{G}) = \{\langle \sigma, \pi, \omega \rangle \in \mathcal{G} \mid (s \stackrel{*}{=} \sigma) \wedge (p \stackrel{*}{=} \pi) \wedge (o \stackrel{*}{=} \omega)\} \tag{5.25}$$

$$\text{where } (x \stackrel{*}{=} y) \equiv \begin{cases} true & \text{if } x = *, \\ x = y & \text{otherwise} \end{cases} \tag{5.26}$$

The reader is referred to Chapter 8 for a discussion of a concrete implementation of this path query language and specifically to Section 8.4 for an evaluation of how it performs in comparison to other RDF query languages.

6 Hiding Reasoning from Application Logic

Reasoning – or *logical inference* – is typically not part of the "arsenal" of technologies employed in mainstream application development. Reasoning is *declarative* in nature and as such largely incompatible with the traditional procedural style of programming. With the advent and increasing acceptance of ontological technologies, the Semantic Web, and other sophisticated representational approaches, the importance of reasoning is elevated. It is therefore reasonable[24] to ask "can we identify appropriate and convenient ways of interfacing procedural programs with mechanisms of reasoning?" The question is not altogether different from the one that needed to be asked (and answered) when rule-based expert systems were gaining popularity (in the 1980's) and developers wanted to interface *rule processing* with procedural programs [9, 166, 98, 50, 148, for example].

Many approaches to RDF (and even OWL) have forced the application programmers to worry about reasoning separately, perhaps even *implement* a reasoner themselves – note that this observation is quite similar to what others have made about some other aspects and approaches to knowledge representation, e.g., Dean & McDermott's remarks about temporal reasoning [53, Section 1.].[25] In fact, most software toolkits for RDF processing merely concentrate on producing sets of triples from XML serializations of RDF graphs, and leave the inferential part to the application programmer. Since the implementation of the "inferential component" of what the model-based semantics of RDF [86] imply is actually a basic requirement for interoperability of RDF-based systems, support for this should be readily available to application programmers. Not only would this ease the task of writing RDF-savvy software, but it would improve the level of interoperability between these systems. Without this minimal support for inference, RDF is largely relegated to mere structured

[24] No pun intended.

[25] Dean & McDermott basically make the argument that the lack of "built-in" means of temporal reasoning in knowledge representation systems too often forces the development of (*ad hoc*) approaches to handling temporal aspects of the particular domain being represented.

data interchange, and its utility will be seriously jeopardized.

What are the *possible* ways of interfacing with a reasoner? They include at least the following:

1. Treat the reasoner similarily the way we tend to treat *database engines*, that is, as a separate component to which one can pose queries and get results back. The representation of the data, when conversing with the reasoner, is completely different from the representation of data that the procedural program uses. There is a conceptual "binding" between the representation, in the sense that a change to one representation forces changes to the other; sometimes (in the case of query engines) these changes have to be propagated to the source code manually.

2. Many ontological representations, including RDF, can be viewed as (at least closely related to) *frame-based* knowledge representation. As mentioned in the previous chapter, frame-based KR systems and object-oriented programming systems (OOPS) have strong parallels, and on the surface it would not be unreasonable to suggest that this "kinship" could be exploited. Certainly the similarity between frame-based KR and OOPS makes it easy to *explain and understand* the former. Even though reflecting declarative representations onto an object-oriented programming language seems doable [128], one unavoidably encounters problems because of the mismatch between representation language semantics and programming language semantics.

3. Given a "natural" representation of the declarative data that allows this data to be used and manipulated by procedural programs, one could envision that – in certain cases – reasoning could be completely hidden from the application developer. In the case of RDF and other representations whose semantics imply a unique *deductive closure*, one could take the approach where the procedural program (mostly) sees just the deductive closure instead of the explicit (perhaps externally acquired) representation. It then leads to the question of *how* this closure is generated (and *when*): it

could be done "up front", or it could be done "on demand". In most cases, issues of *truth maintenance* arise [56, 35].

Those RDF toolkits and libraries that offer reasoning tend to take the first approach. The second approach was typically used by frame-based representation systems via the provision of *procedural attachment* whose extreme manifestations rendered frame-based systems into object-oriented programming languages; traditional way was to implement some typically clumsy way of calling "methods", but some approaches, such as BEEF and PORK went further in attempts to integrate the KR system with the underlying programming language.

In this dissertation, the third approach is adopted, investigating whether *hiding* reasoning from the application developer, by way of exposing a dynamically generated *deductive closure* makes sense. A method extending the use of the path query language presented in the previous chapter is given, demonstrably a natural way of interfacing with graph-based representations, to implement reasoning. Issues of procedural attachment still remain, but, on the one hand, since it is not possible to design the representation language from scratch, trying to align its semantics with the underlying programming language's semantics is a futile task; on the other hand, clumsy "added-on" method invocation facilities may not serve any good purpose either. It therefore makes sense to pursue an approach where *access* to underlying representations is made easy. This, combined with the reasoning mechanisms hidden from the application developer, appears to be a promising direction.

"Semantic interoperability" of RDF-based systems has long been anticipated to materialize because of the polymorphism of shared types and relations as defined by the RDF Schema specification, but – as observed earlier – most software packages for RDF processing merely treat RDF graphs as data structures and leave the inferential part to the application programmer: The model theory for RDF formalizes this notion of inference in RDF, and it could be argued that the inferential mechanism is a *basic minimum requirement for interoperability* of RDF-based systems, and support for this should be readily available for application

programmers. Not providing this support may compromise the interoperability between RDF-based systems.

6.1 Exposing the Deductive Closure

This chapter[26] will investigate the computational aspects of deductive closures of RDF graphs, and pursue an implementation based on the path query language introduced in the previous chapter.

In the previous chapter a query language was examined that operated against a graph $\mathcal{G}$. In this chapter, the notation $\overline{\mathcal{G}}$ is used for the *deductive closure* of graph $\mathcal{G}$. As defined earlier, for *single arc* queries, the function A_{lookup} can be defined as

$$o \in A_{lookup}(s, p, \mathcal{G}) \iff \langle s, p, o \rangle \in \mathcal{G} \tag{6.1}$$

A new function $A_{closure}$ can be introduced, such that

$$A_{closure}(s, p, \mathcal{G}) = A_{lookup}(s, p, \overline{\mathcal{G}}) \tag{6.2}$$

In terms of the model-theoretical formulation of RDF(S) semantics [86], $A_{closure}$ can be defined as

$$o \in A_{closure}(s, p, \mathcal{G}) \iff \langle s, o \rangle \in \mathrm{IEXT}(\mathrm{I}(p)) \tag{6.3}$$

where $\mathrm{I}(x)$ is the RDF(S)-interpretation of a particular graph, and $\mathrm{IEXT}(y)$ is a binary relational extension of a property – i.e., the set of pairs which identify the arguments for which the property is true – as defined in [86]. In other words, $A_{closure}$ provides a view into $\mathcal{G}$ *as if* the deductive closure $\overline{\mathcal{G}}$ had been generated. One approach to implementing $A_{closure}$ will be demonstrated, given an implementation of A_{lookup} as well as other query and update facilities for $\mathcal{G}$.

[26] This chapter is based on my earlier paper [127]. Since its publication, I have found an elegant solution for handling the *domain* and *range* properties of RDF [127, Section 4.3]; this solution is explained in this chapter, and hence the material from the said section of the original paper has been largely omitted. The material, overall, has also been updated for a newer version of the RDF model theory [86]

6.2 Entailment and "RDF(S)-Closures"

The RDF Model Theory [86] defines entailment via the generation of a deductive closure from an RDF graph. The closure is a graph $\overline{\mathcal{G}}$, defined as follows:

$$\langle s, p, o \rangle \in \overline{\mathcal{G}} \iff \langle s, o \rangle \in \text{IEXT}(\text{I}(p)) \tag{6.4}$$

Computing this so-called "RDF(S)-closure" consists of two steps:

1. Addition of a set of axiomatic triples to the RDF graph in question. These triples effectively define classes and properties (and their domains and ranges) in the basic RDF ontological vocabulary.

2. Recursive application of forward-chaining rules to generate all legal triples entailed by the graph in question. These rules could be characterized as follows:

   - *Type Rules* assign default ("root") types for resources (rules **rdf1**, **rdfs4a** and **rdfs4b** in [85]).

   - *Subclass Rules* generate the transitive closures of *subclass* $\rightarrow$ *class* and *instance* $\rightarrow$ *class* links (rules **rdfs7**, **rdfs8** and **rdfs9**).

   - *Subproperty Rules* are used to generate the transitive closures resulting from *subproperty* $\rightarrow$ *property* links. They also propagate property values up the subproperty chain (rules **rdfs5** and **rdfs6**).

   - *Domain/Range Rules* infer resource types from *domain* and *range* assignments (rules **rdfs2** and **rdfs3**).

The rules are highly redundant, and their brute-force, exhaustive, iterative application may not always be a realistic way of computing the closure, although there is work that suggests that at least in some cases this approach is feasible [35]. The nature of brute-force rule

application can be demonstrated via this example: given a graph with only one triple, the rules in step 2 would generate 17 new triples (in addition to the 19 "static" triples added in step 1), but would also result in 493 attempts to add a redundant triple (i.e., one that was already in the database).[27]

6.3 Reasoning in RDF(S) via Forward-Chaining Rules

More optimized forward-chaining rule techniques – such as RETE [67] or TREAT [159, 160] – can be used to make the generation of the deductive closure more efficient. The RETE algorithm operates in terms of a "working memory", a set of tuples the changes of which are filtered through a network that minimizes the number of "checks" that have to be made to understand which rules *could* fire; various strategies can then be employed to decide which rule actually gets picked. If one thinks of the graph $\mathcal{G}$ as the working memory, and the addition and removal of triples as the transactional changes to the working memory, RETE can be conveniently adapted to work with RDF(S). For example, *Pychinko* [108] is a RETE engine adapted to work with RDF graphs, as a compatible replacement to CWM [18].[28]

The application of forward-chaining rules to generate the deductive closure may result in the addition of a large number of new triples in the database; most of these generated results may never even be needed. RETE also caches lots of information about the current working memory (i.e., the graph) in order to speed up the rule discrimination process. This approach may therefore not be feasible in memory-constrained situations. It is therefore interesting to investigate whether some balance could be found between computing the closure in advance vs. defining the access function $A_{closure}$ in such a manner that it can dynamically (i.e.,

[27] In fact, in the aforementioned article [35] Broekstra and Kampman contest this claim, originally presented in [127], but their reasoning may be based on the premise that most of their data sets had very simple class structure; the only "complex" ontology used led to 207% increase in the graph size. Generally, class-rich data (with more `rdfs:subClassOf` relations) would generate more inferred arcs in the deductive closure, as suggested by work on measuring query performance under different schema sizes, e.g. [198].

[28] At the time of writing, CWM used a naïve brute-force forward-chaining algorithm.

on-demand) generate correct results.

6.4 Reasoning in RDF(S) as Theorem-Proving

There are also several *backward-chaining* theorem-proving approaches to generating RDF(S) entailments. These include Euler[29] [164] as well as SiLRI and TRIPLE [54, 188]. Similarly to the approach presented below, these approaches can be used to compute only the entailments of interest, rather than generating *all* entailments as with the forward-chaining approaches.

6.5 Reasoning in RDF(S) as Query Rewriting

A scheme is proposed – a refinement of the earlier approach presented in [127] – to implement deductive closure generation by primarily using graph-walking techniques. The basic idea is that every *inferred* arc of the closure – that is, an arc that the original graph entails, as per [86] – is expressed as an *alternate path* through the original graph. The general goal of this approach is to delay the computation of entailments until they are needed, and to only compute those results that will actually get used. There are similarities to the theorem-proving approaches such as Euler; in the strictest sense, however, the query engine does not guarantee that the alternative paths discovered are Eulerian (nor Hamiltonian),[30] since a node can be visited as many times as it appears in *different states* of the finite state automaton directing the query.

The approach is based on the following basic assumptions:

1. Generally, the computation of the closure can be delayed (even at the expense of the

[29] `http://eulersharp.sourceforge.net/`

[30] An *Eulerian path* visits each edge in a graph exactly once, as first discussed by Euler in 1736 while solving the famous problem exemplified by the Seven Bridges of Königsberg [61]. Similarly, a *Hamiltonian path* visits each vertex in a graph exactly once [84].

time eventually spent in the computation) and memory consumption can be traded for time spent in computation.

2. The computational burden is split in two: some of the work is undertaken during every insertion into $\mathcal{G}$ (i.e., whenever new triples are asserted), and some during every access of $A_{closure}$.

3. Some features of RDF are more prevalent than others in "typical" data; the design of the system will be based on the following perceived distribution of prevalence:

   - subclassing is common,
   - subproperty definitions are used but sparsely,
   - subproperties of `rdf:subPropertyOf` are rare.

   The above distribution can be justified, say, by querying UMBC's Swoogle search engine.[31]

With regard to the dynamic computation of closures, the approach is based on the query language presented in the previous chapter and rewriting access path expressions when accessing the underlying graph. The definition of the slot access function $A_{closure}$ now takes the form

$$A_{closure}(n, path, \mathcal{G}) = A_{lookup}(n, path', \mathcal{G} \cup \mathcal{G}') \tag{6.5}$$

where $\mathcal{G}'$ is a set of triples that have to be added to the original graph $\mathcal{G}$, n is a node in the graph, and $path'$ is the path expression $path$ suitably *rewritten*. The reasoning algorithm is expressed as a set of rewrite patterns of the form $path \overset{c}{\mapsto} path'$ (where the operator $\overset{c}{\mapsto}$ reads as "is rewritten as") and a definition of the set of additional statements $\mathcal{G}'$.

[31] These queries (run on 2006-12-01) revealed 26,008 documents with `rdfs:subClassOf` relations, 5,922 documents with `rdfs:subPropertyOf` relations, and a mere 103 documents with subproperties of `rdfs:subPropertyOf`; this is not the full picture, though, since many of the documents with subclass relations had several hundreds of thousands "usages" of said relation, so the bias in the distribution is even more dramatic than these numbers will lead us to believe.

Computing RDF(S) entailments via graph-theoretical means (namely via graph homomorphisms) has been studied in [12]. The approach to computing RDF(S) entailments via regular expressions is considered in [6]; generally, the query rewriting approach presented is reminiscent of the idea of rewriting query expressions to answer queries based on a set of views rather than the original data contained in a database [171, 37, 77] – in this case the deductive closure is another "view" of the original RDF graph.

A partial solution is demonstrated first: it implements only the *type* and *subclass* rules discussed in section 6.2. This solution will then be extended by adding support for the *subproperty* rules; finally, the original approach presented in [127] is augmented by extending the query language to allow the implementation of the *domain/range* rules.

6.5.1 Type and Subclass Rules

For the two core relations `rdf:type` and `rdfs:subClassOf` the rewritten paths (referring to equation 6.5) are, correspondingly:

$$\texttt{rdf:type} \overset{c}{\mapsto} or(seq(\texttt{rdf:type}, rep(\texttt{rdfs:subClassOf})), value(\texttt{rdfs:Resource})) \tag{6.6}$$

$$\texttt{rdfs:subClassOf} \overset{c}{\mapsto} or(rep(\texttt{rdfs:subClassOf}), value(\texttt{rdfs:Resource})) \tag{6.7}$$

where rewrite pattern (6.6) says that in order to find all values of `rdf:type` of an instance, you first traverse the atomic `rdf:type` link once, and then the atomic `rdfs:subClassOf` link an arbitrary number of times (including zero). Accessing all values of this relation computes the transitive closure of `rdfs:subClassOf`, starting from the designated classes of the instance being queried. Similarily, rewrite pattern 6.7 accesses the transitive closure of `rdfs:subClassOf`. Note that the effects of pattern 6.7 are built into pattern 6.6 so that these rules do not need to be applied recursively. The disjunctions in both expressions ensure that if

the exhaustive search (i.e., transitive closure computation) fails, a default value is generated. Figure 6.1 shows the DFA corresponding to the rewrite rule (6.6).

Apart from `rdf:type` and `rdfs:subClassOf`, other arc labels (RDF properties) are unaffected by the rewrite process, since there is no semantic theory for them. Complex path expressions are rewritten by traversing them recursively, rewriting subexpressions, as follows:

$$op(e_1, \ldots, e_n) \overset{c}{\mapsto} op(e'_1, \ldots, e'_n) \text{ where } \forall i \in [1, n], e_i \overset{c}{\mapsto} e'_i \tag{6.8}$$

Since the WILBUR implementation of a "triple database" always loads a basic "RDF schema" into every newly created database, step 1 of the closure generation process (in section 6.2) is implemented by defining the static triples in this schema.

Please note that the approach taken only makes sense for certain types of triple database implementations. In a relational database implementation – given that queries for finding transitive closures cannot be expressed in relational calculus (see, for example, [155]) – it might make more sense to populate the database with additional triples. In an "in-core" implementation like WILBUR, stepping through the graph has a relatively low cost, and therefore the dynamic approach makes sense, particularly when combined with the potential memory savings.

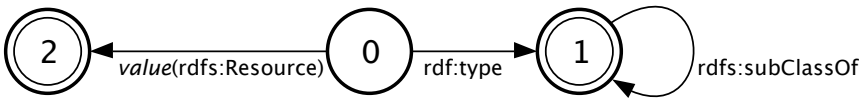

Figure 6.1: DFA corresponding to the complex path from rewrite pattern (6.6)

6.5.2 Subproperty Rules

The partial solution can be extended to provide support for the *subproperty* rules. Referring to equation 6.5, access paths are rewritten as follows: each atomic relation r is rewritten as

$$r \overset{c}{\mapsto} or(r_1, \ldots, r_n) \text{ where } r_i \in A_{lookup}(r, rep(inv(or(p_1, \ldots, p_m))), \mathcal{G}) \tag{6.9}$$

and where $p_1, \ldots, p_m$ are the relation `rdfs:subPropertyOf` and all of its defined subproperties. Please note that this rewriting also applies to all of the atoms of the results of applying the rewrite patterns 6.6 and 6.7. When all values of A are computed the ordering of r_i does not need to be considered. An implementation might, though, apply some specificity ordering to the values based on the graph distance of r_i to r (note that $r_1 = r$).

To avoid redundant lookups, the set of subproperties of `rdfs:subPropertyOf`, $P_{subprop} = \{p_i\}$, can be cached. Each insert into $\mathcal{G}$ where the triple is of the form

$$\langle s, p_i, p_j \rangle \text{ where } p_i \in P_{subprop} \vee p_j \in P_{subprop} \tag{6.10}$$

invalidates and recomputes the cache. The recomputation is effected as follows: assume $P_{subprop_{old}}$ is the current value of the cache, and $P_{subprop_{new}}$ is the recomputed value of the cache; then

$$P_{subprop_{new}} = A_{lookup}(\texttt{rdfs:subPropertyOf}, inv(rep(or(p_1, \ldots, p_n))), \mathcal{G}) \tag{6.11}$$
$$\text{where } \forall i \in [1, n], p_i \in P_{subprop_{old}}$$

In addition to caching subproperty information of `rdfs:subPropertyOf`, the implementation offers other opportunities for caching results. Not only could more of the subproperty information be cached (that is, information about subproperties of *all* relations, not just `rdfs:subPropertyOf`), but other results computed by $A_{closure}$ could be cached as well.

It should be further noted that – similarily to the rewrite rules (6.6) and (6.7) – the property `rdfs:subPropertyOf` itself needs to be rewritten, as follows:

$$\texttt{rdfs:subPropertyOf} \overset{c}{\mapsto} rep(\texttt{rdfs:subPropertyOf}) \tag{6.12}$$

This is done because – according to the definitions given in [86] – `rdfs:subPropertyOf` is both reflexive and transitive.

6.5.3 Domain/Range Rules

The path-rewriting approach works reasonably well for the most part, but as [127, section 4.3] points out, it is not suited to implementing the *domain* and *range* rules (in fact, it proposes a rather awkward solution where certain auxiliary triples are added to the graph to make it possible to have a path from subjects and objects to property nodes of associated statements). Figure 3.2 shows the statement $\langle A, P, B \rangle$ with the corresponding *predicate* (and the associated property node that "names" the predicate), and it can be seen that no path exists between either the subject or the object and the predicate of the statement.

It can be observed, however, that paths (consisting of two arcs) between a subject or object and the property node that names the corresponding predicate are possible when statements are *reified* (i.e., a model of these statements is built using RDF itself). In certain types of implementations these paths can be traversed *without* actually building the graph corresponding to the reifications themselves.

Earlier work related to RDF reification has often dealt with making it more convenient to refer to reified statements and thus making it easier to treat them as first class objects. Such research includes work on extending the RDF model [46], casting Topic Maps as RDF [165], as well as work on several RDF query languages.

Figure 6.2 represents the graph from Figure 3.2 with the reification of the statement added, as defined in [141, section 4.1] and [13, section 7.3]. Now – assuming that you can traverse arcs in the reverse direction – paths exist between the property node P and nodes A and B. The following paths of interest will be considered (these paths are expressed using the

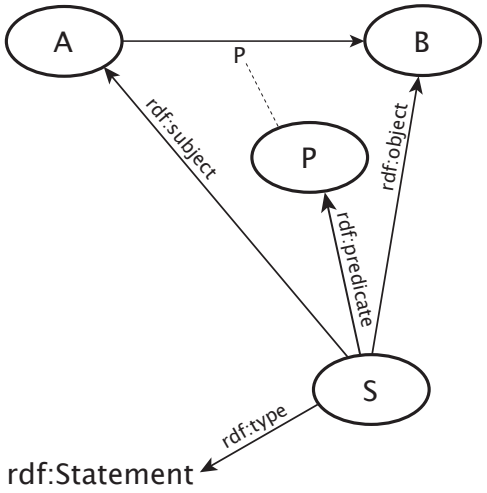

Figure 6.2: Reified Statement S

abstract syntax of query patterns of the Wilbur Query Language):

$$\textit{predicate-of-subject} \equiv \textit{seq}(\textit{inv}(\texttt{rdf:subject}), \texttt{rdf:predicate}) \tag{6.13}$$

$$\textit{predicate-of-object} \equiv \textit{seq}(\textit{inv}(\texttt{rdf:object}), \texttt{rdf:predicate}) \tag{6.14}$$

Since any path in the query language has to be invertible, also the following two paths have to be considered:

$$\textit{inv}(\textit{predicate-of-subject}) \equiv \textit{seq}(\textit{inv}(\texttt{rdf:predicate}), \texttt{rdf:subject}) \tag{6.15}$$

$$\textit{inv}(\textit{predicate-of-object}) \equiv \textit{seq}(\textit{inv}(\texttt{rdf:predicate}), \texttt{rdf:object}) \tag{6.16}$$

Figure 6.3 shows the path matched by the query pattern (6.13). In this dissertation these paths will be called *computed shortcuts* (CSs); in the context of reified statements, CSs are interesting and useful since one can traverse them even if the reified statements themselves do not exist, as long as it is known that they *could* exist and that some other representation that provides information about them exists.

In a "triple-store" implementation, each statement is represented as a tuple $\langle s, p, o \rangle$. Even without reifying *at the graph level*, these tuples are an alternate concrete representation

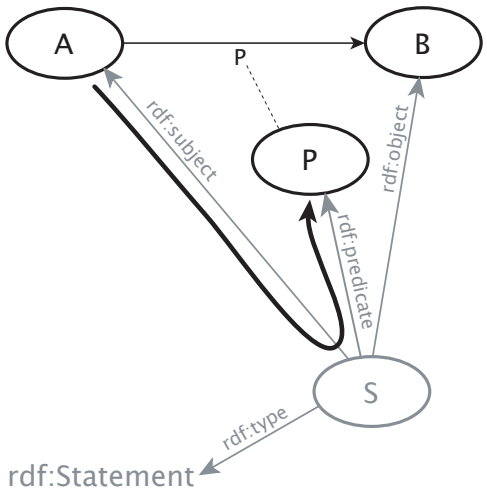

Figure 6.3: Computed shortcut $seq(inv(\texttt{rdf:subject}), \texttt{rdf:predicate})$

of (reified) statements.[32] They can therefore be used to implement the CSs for virtual reification. Using the vocabulary and framework introduced in the previous chapter, CS traversal can be defined as follows:

$$expand(n, \textit{predicate-of-subject}, \mathcal{G}) = \{p \mid \langle s, p, o\rangle \in triple(n, *, *, \mathcal{G})\} \tag{6.17}$$

$$expand(n, \textit{predicate-of-object}, \mathcal{G}) = \{p \mid \langle s, p, o\rangle \in triple(*, *, n, \mathcal{G})\} \tag{6.18}$$

$$expand(n, inv(\textit{predicate-of-subject}), \mathcal{G}) = \{s \mid \langle s, p, o\rangle \in triple(*, n, *, \mathcal{G})\} \tag{6.19}$$

$$expand(n, inv(\textit{predicate-of-object}), \mathcal{G}) = \{o \mid \langle s, p, o\rangle \in triple(*, n, *, \mathcal{G})\} \tag{6.20}$$

A query engine implementation can identify all these CSs while canonicalizing query expressions, and can substitute a special "query atom" for each of them; special cases of the function *expand*, as defined above, then exist for each of these query atoms. Thus the cost of traversing any one of these CSs is the same as the cost of traversing any one arc in the graph, and the same "effect" is achieved as the awkward solution of [127, section 4.3] but without any additional triples. The domain and range rules can now be expressed more elegantly;

[32] Similar approach is taken in [165] to delay concrete reification.

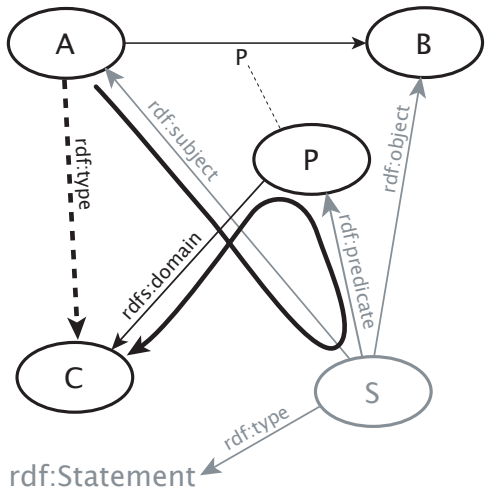

Figure 6.4: Inferring the type of node A using the domain rule

consider the following *rewrite pattern*:

$$\begin{aligned}\texttt{rdf:type} \overset{c}{\mapsto} or(&seq(\texttt{rdf:type}, rep(\texttt{rdfs:subClassOf})),\\ &seq(\textit{predicate-of-object}, \texttt{rdfs:range}, rep(\texttt{rdfs:subClassOf})),\\ &seq(\textit{predicate-of-subject}, \texttt{rdfs:domain}, rep(\texttt{rdfs:subClassOf})),\\ &value(\texttt{rdfs:Resource}))\end{aligned} \tag{6.21}$$

where the definitions of *predicate-of-object* and *predicate-of-subject* are according to (6.14) and (6.13). This rule corresponds directly to [127, rewrite pattern (8)].

Figure 6.4 illustrates how the domain rule (thick solid line), via the traversal of the graph, can be used for inferring the type of a node (thick dashed line). Figure 6.5 illustrates the DFA corresponding to the complex path from rewrite pattern (6.21).[33] This DFA is used for "walking" the RDF graph while making use of the query expansions (in Section 5.3.1), to yield the `rdf:type` of a node (in the process, the following RDF(S) entailment rules [86, section 7.3] are satisfied: *rdfs2*, *rdfs3*, *rdfs4a*, *rdfs8*, *rdfs9*, *rdfs10* and *rdfs11*).

[33]Compare this with the simpler DFA in Figure 6.1.

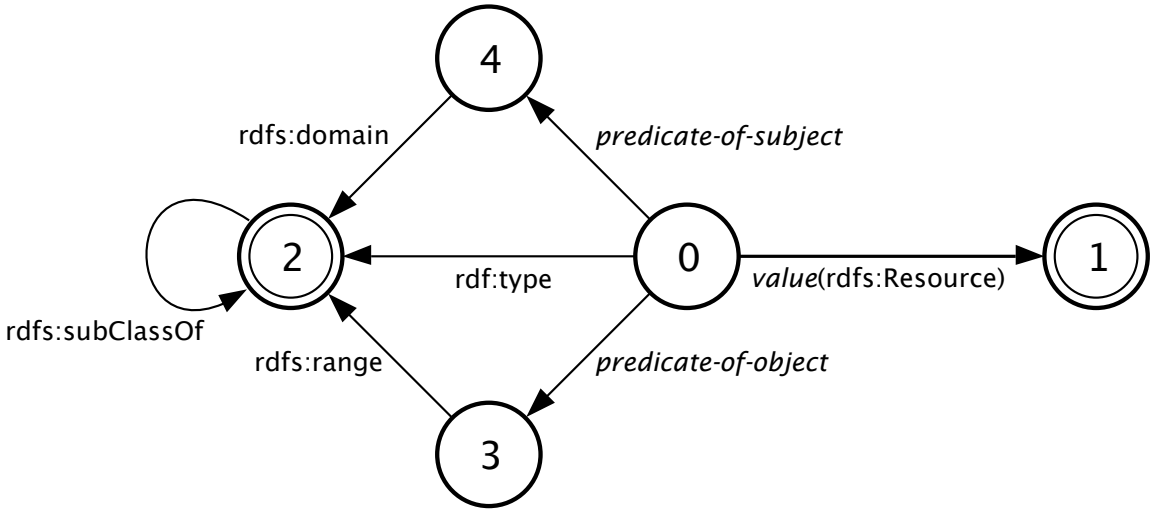

Figure 6.5: DFA corresponding to the complex path from rewrite pattern (6.21)

6.5.4 Inverting Paths with Default Values

In order to allow all possible queries to be transformed, a solution is needed for computing the results of queries that consist of an inverse of a path that gets rewritten to something that contains a default value (e.g., `rdf:type`). Generally, the default values are used to ensure that `rdfs:Resource` is the class of every instance and the superclass of every class. Computing something like $A_{closure}(\texttt{rdfs:Resource}, inv(\texttt{rdf:type}), \mathcal{G})$ results in the set of *all nodes* in $\mathcal{G}$. Since this may only be an intermediary result while computing the results for a more complex query, the use of something akin to *lazy evaluation* will prevent large intermediary data sets from being generated. The special token $*_{all}$ "represents" the set of all nodes during the computation of intermediate results:

$$expand(\texttt{rdfs:Resource}, inv(\texttt{rdf:type}), \mathcal{G}) = \{*_{all}\} \tag{6.22}$$

$$expand(\texttt{rdfs:Resource}, inv(\texttt{rdfs:subClassOf}), \mathcal{G}) = \{*_{all}\} \tag{6.23}$$

$$expand(*_{all}, a, \mathcal{G}) = \{o \mid \langle s, p, o\rangle \in triple(*, a, *, \mathcal{G})\} \tag{6.24}$$

6.5.5 Implementation Summary and Evaluation

Table 6.1 summarizes how the approach described implements the rules of the RDF(S) Model Theory. Going forward, one might consider using forward-chaining rules to generate those parts of the deductive closure that cannot be generated using query rewriting (these "rules" – cases *rdf1*, *rdfs6*, *rdfs12* and *rdfs13* – are currently hard-coded in the database manipulation methods).

Table 6.1: Implementation Summary

Rule	$\overset{c}{\mapsto}$ **Pattern**	**Other Implementation**
rdf1	n/a	during insertions to $\mathcal{G}$
rdf2	n/a	special case of *expand* (5.22)
rdfs1	n/a	special case of *expand* (5.22)
rdfs2	(6.21)	
rdfs3	(6.21)	
rdfs4a	(6.21)	
rdfs4b	(6.21)	
rdfs5	(6.9)	
rdfs6	(6.9)	if (6.10) holds during insertion to $\mathcal{G}$ then invoke (6.11)
rdfs7	(6.9)	
rdfs8	(6.21)	
rdfs9	(6.21)	
rdfs10	(6.7)	
rdfs11	(6.7)	
rdfs12	n/a	during insertions to $\mathcal{G}$
rdfs13	n/a	during insertions to $\mathcal{G}$

The complexity of path traversal and path queries has been studied extensively; for example [197, 210, 155, 167, 82]. Even though the general problems tend to be NP-complete [155], several restricted variations of the problem have lower complexity. Most of the graph processing required for the solution presented in this dissertation is reduced to the computation of transitive closures which can be accomplished in polynomial time [210]. Beyond that, most of the complexity considerations – such as those discussed in [82] – are considered to be out of scope for this dissertation.

6.6 Generalization of Rewriting Approach to Reasoning

As discussed, the aforementioned RDF(S) reasoner is implemented by *rewriting* access queries – expressions of the WilburQL language – to effectively make it look like the database, in addition to the graph stored in it, contains all the entailments of this graph. Rewriting is done by recursively substituting all occurrences of those RDF properties that have a semantic theory (`rdf:type`, `rdfs:subClassOf` and `rdfs:subPropertyOf`) with a more complex query expression that effectively constitutes something loosely resembling backward-chaining rules. The rewritten queries create a "view" into the underlying graph database that contains the RDF(S) closure of the original graph.

The idea of *rewriting* has been used as a general computational vehicle [93], in transforming and optimizing (especially functional) programming languages [203, 202, 116], in various practical software systems (such as the `sendmail` program[34] of UNIX or the Apache web server[35]) as a "customization" mechanism, and in performing service discovery and composition [16, 137]. It has also been used in query systems for semi-structured data for providing augmented or federated views of databases [36, 171, 77].

We will generalize the rewriting mechanism used in the reasoner into a simple rule system.

[34] `http://www.sendmail.org/`
[35] `http://httpd.apache.org/`

Rewrite rules take the general form

$$p \mapsto q \tag{6.25}$$

where p is an *atomic* expression of the query language (i.e., something that conceivably could name an RDF property), and q is an arbitrarily complex query expression that does not contain p (with one exception that's described below).[36] The *rule engine*, upon seeing a query expression, iteratively applies the rewrite rules to the expression until no rule applies. The resulting rewritten query is then presented to the *query engine* that computes and retrieves the result set.

In queries of the form (6.25) it is possible for the expression q to contain subexpressions of the form $norewrite(r)$ where r is an expression that may contain the atomic expression p. The rule engine does not attempt to rewrite these expressions, and the *norewrite* operator is ignored by the query engine, in the sense that from its point of view, $norewrite(p) \equiv p$. This mechanism allows one to augment the behavior of existing RDF properties, for example via the following pattern:

$$p \mapsto or(norewrite(p), q)$$

Referring to (5.3), we can give property p the default value of r by using the following rule:

$$p \mapsto or(norewrite(p), value(r)) \tag{6.26}$$

In comparison to most of the aforementioned existing work on rewriting, our approach to rewriting query expressions is rather simplistic and resembles simple *macro expansion*. Most notably, our rewrite rules do not contain any variables or other types of patterns that would require matching to determine rule applicability (apart from a simple exact matching of names). In that sense, they resemble *entity references* in XML [31, section 4.1].

Virtual properties are rewrite rules that allow the values of new properties to be computed rather than being stored in the underlying database. By expressing rewrite rules as instances

[36] Obviously the "closure rules" of the RDF(S) reasoner (denoted with $\overset{c}{\mapsto}$) described earlier are a special case of the more general rules discussed here.

of a subclass of `rdf:Property` allows us to have virtual properties to have `rdfs:domain` definitions, effectively associating then with classes; given a virtual property p with a domain d, we can use this information to automatically query for values of p whenever, for example, visualizing an instance of the class d.

The rewrite rule system is discussed in more detail in [133].

7 Practicality of RDF(S) in Applications

RDF is a simple representation language capable of representing taxonomies as class hierarchies. The model theory of RDF [86] implies that RDF has a unique deductive closure, making RDF easy to integrate with a procedural programming model since all inferencing can be hidden from the application logic by presenting a view where the deductive closure, instead of the explicit triples in the underlying triple store, is the actual knowledge base.

One of RDF's shortcomings is the inability to use a reasoner to determine anything about object equality. This has implications to the management of object *identity*, particularly with respect to blank nodes. Two specific cases are particularly interesting:

- Explicitly stating that two nodes in a graph are the *same* node. Some might argue that what in fact would be more useful is the *equivalence of classes*, but node equality in RDF(S) is in fact a natural way to think of the *graph*: one merely states that two nodes in the graph are, in fact, a single node.

- Determining node identity based on some property of the node. Instead of determining node identity using the node's URI, the node identity is based on a specific value of a specific property (although the "orthodox" approach to the Semantic Web suggests that all objects that have to be described have a URI, this is hard to achieve in practice: in the "real world", in fact, most objects do *not* have a URI).

OWL [152, 52, 173] can handle both cases; the first by allowing an equality relation between objects (`owl:sameAs`), and the second by allowing properties to be defined as *inverse functional properties* – the latter are like primary keys in a database; for these properties the following holds:

$$p(r_1) = p(r_2) \Rightarrow (r_1 = r_2) \tag{7.1}$$

where p is an inverse functional property and r_i are resources. It would, therefore, be worth

studying whether these two features would change how the language can be used.

This chapter introduces a new language, dubbed RDF^{++} [134], defined as RDF(S) with the addition of the two aforementioned features. This language, like "classical" RDF, also has a unique deductive closure and therefore can be used in the application framework described. Through examples the utility and benefit of the new features as improved expressiveness of the language will also be justified.

7.1 Semantic Theory for RDF^{++}

This section defines the *axiomatic semantics* of RDF^{++}. The formalization is similar to the axiomatization of RDF and RDF(S) as given in [66] and [79], except that the KIF (or "KIF-like") syntax is dispensed with in favor of a more traditional first-order logic syntax.

The definition of RDF^{++} semantics has the following three parts:

1. "Shorthand" axioms, to be used to make later axioms easier to write:[37]

$$type(x, y) \iff holds(\texttt{rdf:type}, x, y)$$

$$class(x) \iff type(x, \texttt{rdfs:Class})$$

$$property(x) \iff type(x, \texttt{rdf:Property})$$

2. Axiomatic triples (definitions of the core classes and properties); these establish an initial, minimal schema (there are other axiomatic triples but they are not strictly

[37]Note that RDF "triples" are expressed using *holds*, a ternary relation such that each RDF statement with any predicate p, any subject s, and any object o is translated into the relational sentence $holds(p, s, o)$. This is a well-known technique for embedding a higher-order *syntax* in first-order logic. An alternative approach would be the use of *relational extensions* as adopted for L_{base} in [79, Section 2.3]; this approach would allow variables in relation positions, making the language appear as higher-order, yet keeping the language semantics as first-order.

necessary from the standpoint of the other axioms presented here):

$$\mathit{class}(\texttt{rdfs:Resource})$$

$$\mathit{property}(\texttt{rdf:type})$$

$$\mathit{holds}(\texttt{rdfs:domain}, \texttt{rdf:type}, \texttt{rdfs:Resource})$$

$$\mathit{holds}(\texttt{rdfs:range}, \texttt{rdf:type}, \texttt{rdfs:Class})$$

$$\mathit{property}(\texttt{rdfs:subClassOf})$$

$$\mathit{holds}(\texttt{rdfs:domain}, \texttt{rdfs:subClassOf}, \texttt{rdfs:Class})$$

$$\mathit{holds}(\texttt{rdfs:range}, \texttt{rdfs:subClassOf}, \texttt{rdfs:Class})$$

$$\mathit{property}(\texttt{rdfs:subPropertyOf})$$

$$\mathit{holds}(\texttt{rdfs:domain}, \texttt{rdfs:subPropertyOf}, \texttt{rdf:Property})$$

$$\mathit{holds}(\texttt{rdfs:range}, \texttt{rdfs:subPropertyOf}, \texttt{rdf:Property})$$

$$\mathit{property}(\texttt{rdfs:domain})$$

$$\mathit{holds}(\texttt{rdfs:domain}, \texttt{rdfs:domain}, \texttt{rdf:Property})$$

$$\mathit{holds}(\texttt{rdfs:range}, \texttt{rdfs:domain}, \texttt{rdfs:Class})$$

$$\mathit{property}(\texttt{rdfs:range})$$

$$\mathit{holds}(\texttt{rdfs:range}, \texttt{rdfs:domain}, \texttt{rdf:Property})$$

$$\mathit{holds}(\texttt{rdfs:range}, \texttt{rdfs:range}, \texttt{rdfs:Class})$$

3. Finally, axioms that establish the semantics of those elements of the RDF^{++} vocabulary that have special semantics. Axioms (7.2)–(7.6) are, in essence, the same as the axioms presented in [66], except that axioms (7.5) and (7.6) use an *implication* instead of *equivalence* as in [66, *range axiom 3* & *domain axiom 2*]; axioms (7.7) and (7.8)

represent the additional semantics for RDF^{++} that has been borrowed from OWL:

$$holds(p, s, o) \implies property(p) \tag{7.2}$$

$$holds(\texttt{rdfs:subClassOf}, x, y) \iff \tag{7.3}$$
$$class(x) \wedge class(y) \wedge [\forall z : type(z, x) \implies type(z, y)]$$

$$holds(\texttt{rdfs:subPropertyOf}, x, y) \iff \tag{7.4}$$
$$property(x) \wedge property(y) \wedge [\forall o, v : holds(x, o, v) \implies holds(y, o, v)]$$

$$holds(\texttt{rdfs:domain}, p, c) \implies \tag{7.5}$$
$$property(p) \wedge class(c) \wedge [\forall x, y : holds(p, x, y) \implies type(x, c)]$$

$$holds(\texttt{rdfs:range}, p, c) \implies \tag{7.6}$$
$$property(p) \wedge class(c) \wedge [\forall x, y : holds(p, x, y) \implies type(y, c)]$$

$$holds(\texttt{owl:sameAs}, x, y) \iff x = y \tag{7.7}$$

$$holds(p, x, z) \wedge holds(p, y, z) \wedge type(p, \texttt{owl:InverseFunctionalProperty}) \implies$$
$$holds(\texttt{owl:sameAs}, x, y) \tag{7.8}$$

By not requiring that RDF semantics be defined independently of RDF(S) semantics, one can state (as the "axiomatic triples") things that otherwise would have to be "bootstrapped" in RDF since the RDF(S) vocabulary is not available.[38] For example, the `rdfs:domain` and `rdfs:range` for `rdf:type`, allowing the elimination of [66, *Type axiom 2*] – this is because axioms (7.5) and (7.6) allows one to infer the same information.

Essentially, the semantic conditions defined for RDF^{++} are those also defined for RDF and RDF Schema [86, sections 3.1 and 4.1], with the specific features that have been added are essentially "borrowed" from OWL, adapted from [173, section 5].

[38] In fact, even [66, *Resource axiom 1*] conflates the two by asserting *class*(`rdf:Resource`).

8 Concrete Semantic Web Platform

> *"Any sufficiently complicated C or Fortran program contains an* ad hoc*, informally-specified, bug-ridden, slow implementation of half of Common Lisp."*
>
> – Philip Greenspun[39]

In this chapter WILBUR, a software library and framework based on the ideas presented in Chapters 5–7, will be described. WILBUR [125, 127, 128, 129] is a platform on which Semantic Web applications can be built.

WILBUR (and its re-implementation WILBUR2) have been written in Common Lisp [193] and were initially based on the BEEF [95, 118, 120] source code. Common Lisp is an expressive programming language particularly well suited to exploring a data-oriented programming paradigm. Its associated *metaobject protocol* [111] allows substantial changes to be introduced to the "inner workings" of the language.

One view of WILBUR is that it takes a low-level approach to integration, by allowing manipulation of RDF graphs (as suggested in Chapter 5). As a fundamental principle, it strives for tight integration between RDF data and the intrinsic features of Common Lisp; the integration focuses on the following areas:

1. Ease of use of Common Lisp data structures with RDF – this is achieved via a simple yet flexible query language WILBURQL, based on the abstract query language described in section 5.3. The query mechanism is thought of as "glue" between the programming language and the representation language.

2. Input and output of RDF data in a "Common Lisp -friendly" manner – WILBUR takes the read/print consistency rules of Common Lisp [193, section 11.1.] very seriously,

[39] This is *Greenspun's Tenth Rule*; see `http://en.wikipedia.org/wiki/Greenspun's_Tenth_Rule`

making it possible to use literal RDF data (nodes and literals) in Common Lisp source files.

3. Use of the Common Lisp condition mechanism for signaling unexpected situations – WILBUR's condition class hierarchy is quite elaborate, allowing precise detection and handling of anomalous situations.

In addition (and perhaps most importantly), WILBUR hides reasoning from the application programmer, using the technique described in Chapter 6.

8.1 Toolkit Concepts

Like other RDF toolkits, WILBUR offers an API for manipulating RDF data (graphs, nodes, etc.). It implements the RDF data model by providing five abstract interfaces (and their concrete implementations):

1. A class is provided to represent *nodes* of an RDF graph. Each node may have a URI (Universal Resource Identifier) string associated with it, in which case the node is considered to be named; nodes without a URI are called blank or anonymous.

2. A mapping from URI strings to nodes is provided by *dictionaries*. The system uses a single default dictionary where all named nodes are placed. The unique mapping from URI strings to node instances allows the implementation of strict read/print correspondence for nodes.

3. *Triples* represent the labeled arcs of an RDF graph. A triple consists of a subject (a node), a predicate (also a node), and an object (either a node or a literal); each triple also has an associated source (again, a node), designating the file or HTTP URL from which the triple was originally parsed.

4. Collections of triples are stored in *databases*. The system can assume a default database, but also exposes a lower-level API where the database can be specified explicitly. Simple query functionality is provided for selecting triples from a database, similar (at least in spirit) to the "find" interface of the Stanford RDF API [154] or the `SimpleSelector` interface of Jena [151] – the interface corresponds roughly to the function $triple(s, p, o, \mathcal{G})$ described in section 5.3.1. WILBUR databases also implement the full query language based on path patterns; by default (and as illustrated in section 5.3.1), the implementation of the full query language is based on the low-level interface exemplified in $triple(s, p, o, \mathcal{G})$.

5. *Literals* represent literal values (as opposed to description nodes) present in an RDF graph. Literals have an (optional) associated datatype (an XSD primitive datatype) and an (optional) language tag.

WILBUR's high-level query language, WILBURQL, is based on the discussion of a path query language (in Section 5.3). As for its expressive power, it fared well in comparison to the many "mainstream" query languages proposed for RDF, as demonstrated when a recent RDF query language comparison [83] is extended to cover WILBURQL [129].

8.2 Hidden Reasoner

A complementary view of WILBUR is that its databases can be used to hide reasoning from the application developer (as suggested in Chapter 6). A special database class is provided that implements query rewriting and exposes a (virtual) view of the deductive closure of the graphs loaded into the database. The reasoner implements the extended RDF(S) language RDF^{++} described in Chapter 7.

8.3 Input and Output of Data

In order to facilitate *access to data*, WILBUR offers a mechanism for "populating" databases from local (file system) and global (HTTP) sources, as well as a query language for flexible, selective access to the data stored in these databases. Populating databases (called *loading* in WILBUR) involves various parsing functionality (parsers for XML, RDF, etc.) and a simple HTTP client API for accessing remote URLs for the same reason. Both the data source loading functionality as well as the parsing functionality are packaged as extensible "protocols" that allow new data sources and new parsers to be easily added.

The URI shorthand syntax allows one to use XML QNames that are mapped to the corresponding `node` instances. The (normally unassigned) macro character `!` is used. For example, under the default namespace mappings established by the class `dictionary`, writing `!rdf:type` would – at "read time" [193, Section 22.1.5.] – map into a node whose URI is "`http://www.w3.org/1999/02/22-rdf-syntax-ns#type`". It is possible to use this notation in source files, compile those files, and later load the binaries into a Common Lisp system; because the node mappings are then resolved at "load time", two separately compiled binary files that make a reference to similarily named nodes can expect these nodes to be `eq`. This makes it possible to use the `!`-notation in, say, `eql`-specializers [193, Section 28.1.6.2.] and in a way allowing for simple procedural attachment. Note that the node shorthand works even if the nodes are unresolved; resolution happens whenever a namespace mapping is established, and that way "unresolvable" QNames (ones for which a namespace mapping is yet to be established) act as "forward references" to the desired node objects.

8.4 Practical Evaluation of Expressive Power of WilburQL

The query language WILBURQL is the central component of WILBUR. This section will provide an evaluation of its expressive power, specifically in comparison to other RDF query

languages. The comparison is based on an earlier comparison from University of Karlsruhe [83] and its extension to cover WILBUR [129]. In addition, results for SPARQL have been added. This is a practical evaluation of expressive power, without attempts to formalize the semantics of the language(s).

Since the original comparison [129] was published, *filtering* and *restriction* operators were added to WILBURQL, hence the results are now better. Specifically, the use of the idiom:

$$seq(inv(\texttt{rdf:type}), restrict(X)) \tag{8.1}$$

combined with `rdfs:Resource` as the root allows us to force (part of) any query to start from a particular node X. As explained in Section 6.5.4 the computation of the results of this query *does not* generate – as an intermediate result – all the nodes in the graph. As an example, the "union" test ("return the labels of all topics and union the titles of all publications") can be satisfied with the following – admittedly somewhat contrived – query (with `rdfs:Resource` as the root):

$$\begin{aligned} seq(inv(\texttt{rdf:type}), or(seq(restrict(\texttt{acm:Topic}), inv(\texttt{rdf:type}), \texttt{rdfs:label}), \\ seq(restrict(\texttt{q:Publication}), inv(\texttt{rdf:type}), \texttt{q:title}))) \end{aligned} \tag{8.2}$$

The results are shown in Table 8.1, as follows:

1. There are 14 distinct tests; the reader is referred to the original comparison article [83] for a description of these tests.

2. There were eight query languages in the original comparison; the column "Pass" shows the percentage of the query languages that passed a particular test. In subsequent columns, a blank entry denotes failure, • denotes a successful test, and ∘ denotes a partially satisfied test (called "restricted" in the original comparison).

3. RDQL [185] as an example of a relational query language, and Versa [168] as a path query language were included from the original comparison.

4. The results in the column for SPARQL were produced by the author, based on the SPARQL specification [176].

5. Two different "versions" of WilburQL were included: "plain" WilburQL as well as WilburQL enhanced with Common Lisp (one could think of this as a combination of "plain" WilburQL queries with Common Lisp used as a scripting language; this is possible due to the integration of WilburQL with the underlying programming language).

Perhaps unsurprisingly, SPARQL fares better than most query languages. Most notably, SPARQL does a lot better than RDQL, its predecessor. Due to its relational nature, SPARQL fails those tests that involve recursive or repetitive traversal of the graph. The overall average pass rate of the original comparison was 53% – some languages did quite well, but they have not been deployed; it should be noted that no language in the original comparison achieved 100% pass rate

Despite its simplicity, WilburQL does well, comparatively speaking, and when combined with *simple* Common Lisp programs is able to satisfy all 14 tests – this is acceptable and even desirable, considering that the aim of WilburQL was *integration with the underlying programming language*.[40] It should further be noted that even SPARQL could be made to pass all tests when combined with a programming language or scripting language, but the recursive/repetitive tests would require the execution of an arbitrary number of SPARQL queries by the program; this was not the case in any of the combined WilburQL+CL tests, where the programs were merely used to post-process query results – the reader is referred to [129] for details.

[40]You could also think of Common Lisp as a scripting language here, a notion probably quite horrifying to most hard-core Common Lisp aficionados.

Table 8.1: Comparison of RDF Query Languages

Test	**Pass**	**RDQL**	**Versa**	**SPARQL**	**WilburQL**	**WilburQL+CL**
Path Expression	100%	•	•	•	•	•
Optional Path	38%		•	•	•	•
Union	88%	•	•	•	•	•
Difference	50%		○	•		•
Quantification	38%					•
Aggregation	63%		•			•
Recursion	63%		•		•	•
Reification	25%	○	○	•	•	•
Collections & Containers	13%	○	○	○	•	•
Namespace	63%	○		•	•	•
Language	38%			•	•	•
Lexical Space	100%	•	•	•		•
Value Space	50%	○		•		•
Entailment	38%	○			•	•
		21%	**43%**	**64%**	**64%**	**100%**

9 Exploiting Reasoning and Serendipity in Applications

In this section some applications and systems built using WILBUR are described, as illustrative examples of the challenges outlined earlier (and the proposed solutions to those). Because WILBUR is a framework for Common Lisp, all source code examples are naturally written in Common Lisp; the reader is advised to study a textbook on the language – such as [186] – to become familiar with some of the finer details. It should be noted that because of the expressive power of Common Lisp, some aspects of WILBUR may not translate to other programming languages all that well.

9.1 Trivial Example: An RSS Formatter

First, a trivial example is presented for converting syndicated (news) feeds formatted in RSS 1.0[41] into XHTML viewable in a web browser. Basically, the program, when given a URL and an output stream, will read the contents of the URL into a triple store, and then format the RSS profile found therein into the output stream as XHTML.[42]

It should be noted that the program supports reasoning "under the hood", by virtue of using WILBUR's triple-store class that performs RDF^{++} reasoning. The source code is shown as Figure 9.1.

[41] `http://web.resource.org/rss/1.0/spec`

[42] When I originally wrote this example around 2000 it made more sense, but now many Web browsers directly support reading RSS-formatted feeds. It serves, however, to illustrate what a really simple WILBUR program will look like. It has also been updated for WILBUR2 (which has XML output support). An astute reader will also naturally observe that this program will not support newer versions of RSS because they are not based on RDF. Vis-à-vis WILBUR, the general thinking is that these types of situations requiring *syntactic* transformations can be handled using a mechanism like GRDDL [89].

```
(defun rss->xhtml (url stream)
  (let* ((db (make-instance 'edb :populate-with (list url)))
         (channel (first (db-get-values db !rss:channel '(:inv !rdf:type)))))
    (flet ((value (f s) (literal-value (first (db-get-values db f s)))))
      (with-tags (stream "html" "body")
        (format stream "<h1>News: <a href=~S>~A</a></h1>"
                       (value channel !rss:link)
                       (value channel !rss:title))
        (dolist (item (db-get-values db channel '(:seq !rss:items :members)))
          (format stream "<h2><a href=~S>~A</a></h2>~%~A~%"
                         (value item !rss:link)
                         (value item !rss:title))
                         (value item !rss:description))))))
```

Figure 9.1: RSS-to-XHTML formatter

9.2 Browsing Semantic Data

In the case of "spontaneous" information integration – a task well suited to Semantic Web technologies – it is vital for the users to *see* their data. This section will present a tool that allows users to explore Semantic Web data via a mechanism they already know well, namely *browsing*.

As observed earlier, RDF is the fundamental building block of the Semantic Web, and can be thought of as *directed, labeled graphs* that can easily be presented as hypertext; at its simplest the presentation only has to rely on the RDF metamodel, making *any* RDF data "browsable" without any special knowledge of the schema(ta) involved. More specialized, *schema-aware* presentations can then be built on top of the basic solution. Enabling browsing is further encouraged by the fact that, despite its simple data model, users and developers are often put off by RDF's cryptic XML-based syntax.

Many tools have been constructed for searching, exploring and querying complex information spaces such as semistructured data representations or collections with rich metadata. These tools often offer a mixture of features such as faceted search, clustering of search results, etc. Examples of such systems include Lore [74, 75], Flamenco [211], and others. More specifically, recent years have also seen a number of "Semantic Web browsers" emerge; examples of this type of software include mSpace [161], Haystack [177], Magnet [187], DBin [200], semantExplorer [184], Tabulator [19], and others. These systems allow the browsing of Semantic Web data and provide various ways of searching and visualizing this data. Many of them combine semantic data with "classical" Web content or other human-oriented content in order to provide the user a comprehensive access to information.

More recently, the idea of a "Semantic Desktop" has emerged [55, 182, 183]; these systems largely focus on personal information management, and demonstrate the utility of exposing "legacy" data in Semantic Web formalisms, often involving transformations from both file-based data as well as databases. The large body of work on mapping data between relational databases and Semantic Web formalisms is described in [14].

Yet another twist on the browsing theme is Piggy Bank [96]. It allows semantic data to be *collected* while browsing "ordinary" Web pages. This data can originate either in some Semantic Web format, or it can be "scraped" or transformed from a number of known page formats.

9.2.1 Browsing RDF Data

Since the pervasive mainstream adoption of the World Wide Web, browsing has become a natural user interface paradigm. This section introduces a Wilbur application constructed to enable users to browse RDF graphs, a simple tool called OINK[43] [132]. It provides a *node-oriented view* of graphs, visualizing each node as a Web page. These pages show the various

[43]"Open Integration of Networked Knowledge"

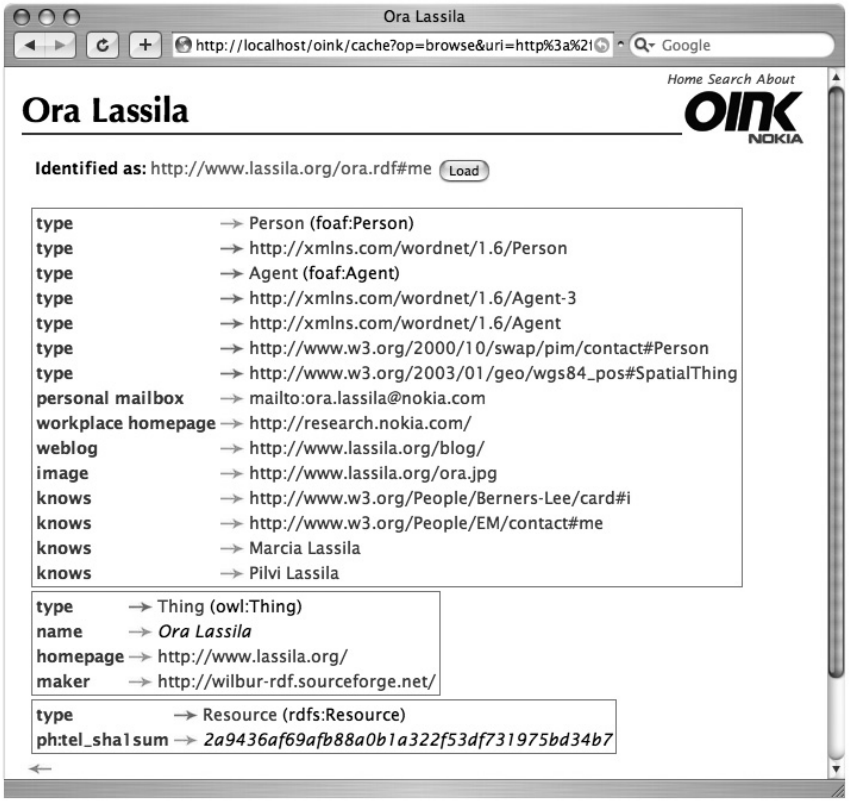

Figure 9.2: A typical page from OINK

ways of identifying the node in question, as well as a list of properties and their values. Figure 9.2 shows a typical page from OINK, as rendered in a Web browser. OINK allows the simultaneous viewing and exploration of any number of RDF graphs, thus supporting the use of RDF in information integration (technically, all documents loaded into OINK form a *single* graph; OINK's reasoner supports not only RDF(S) but also *inverse functional properties* – allowing, say, automatic integration of FoaF [32] profiles).

In addition to visualizing the *outbound* edges, OINK shows the *inbound* edges as well. This allows users to navigate the edges of a graph in the reverse direction (a typical example of this would be when one wants to navigate from an RDF class to any of its subclasses). Since OINK is based on the RDF metamodel – where everything has a representation in RDF

itself – all items on an OINK page may be navigated to (this includes the definitions of the RDF properties *and* their values). The simple metamodel translates directly to a clear and understandable user interface model.

OINK was originally envisioned as a lightweight tool primarily for "debugging" RDF data, but subsequent use of the system has encouraged its development as a platform for building customized browsing solutions for complex data. The use of the underlying reasoning engine also allows automatic integration of data from multiple sources, further reinforcing the idea that Semantic Web technologies can be used for "spontaneous" end-user tasks.

9.2.2 Architecture and Implementation of OINK

The overall architecture of the OINK system is illustrated in Figure 9.3. In broad strokes, OINK is built around a storage and query engine for RDF graphs. An HTTP *servlet* queries the data in the RDF store, and renders it as XHTML. The current prototype implementation of OINK is written in Common Lisp [193] on top of the WILBUR Semantic Web toolkit [125, 127], and uses the Portable AllegroServe web application server [186, chapter 26].

The main-memory RDF database (= "triple store"), acting as a cache, is the central component of OINK.[44] The user registers *data sources* that are loaded into the cache; various facilities are available to the user to maintain this cache:

- Data sources can be "refreshed" manually. OINK relies on the underlying WILBUR mechanism where "old" data is replaced by freshly parsed "new" data as an atomic transaction of the database. Every RDF statement in OINK is linked to the data source(s) that asserted it. Sources are treated as nodes, and any statements about these show up in OINK as well.

[44] So as not to confuse the reader, it should be pointed out that this is a lower-level caching mechanism than the "Semantic cache" reported on earlier [110].

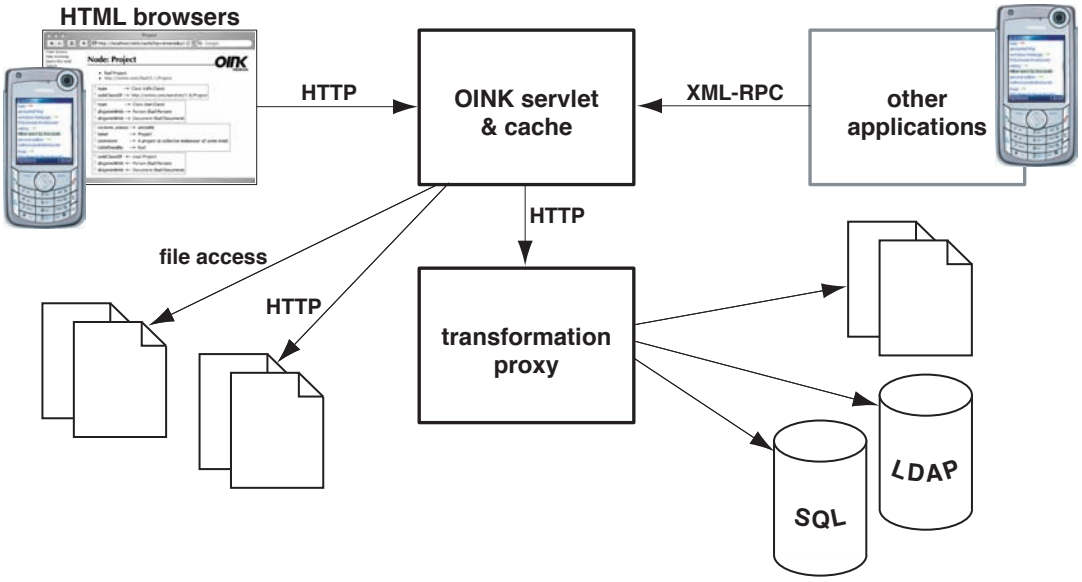

Figure 9.3: The overall architecture of OINK

- Data sources can also be *automatically* refreshed using two different methods. The user or the document itself can designate an interval after which the data in the cache is considered "old"; refreshing can also rely on the HTTP caching semantics, based on the HTTP headers from the response when the data was loaded.

The OINK cache merely remembers *where* the data comes from, so reinitializing the cache causes all the data sources to be loaded anew from their original locations.[45] The cache also provides an XML-RPC interface through which other applications can control the loading of data sources.

The cache can be searched for substring matches on literals or queried using WilburQL path queries [127]. Current work on OINK involves a mechanism where queries are generated automatically from the user's paths through the graph; these queries, in turn, can be used to find similarly related items [133].

[45] In one of the early demonstrations of OINK someone asked to see the XML corresponding to some RDF data. OINK is of little help here, since it does not "care" about any syntactic representations of data, and discards any such representations after they are successfully parsed.

Based on the WILBUR triple store, the OINK cache supports RDF(S) entailments [86] via on-demand generation of the *deductive closure* of the graph stored in the cache. Reasoning is based on a technique where access queries to the triple store are rewritten so that they reflect the "virtual" deductive closure [127]. In addition to RDF(S), the reasoner supports integration-related features such as owl:sameAs and *inverse functional properties* [173].

The OINK servlet merely responds to HTTP requests by querying the cache and producing pages – one per each node in the graph, on demand – that reflect the graph structure. Essentially, the generation of a page for an RDF resource (node) is divided into three components:

1. The node is identified: if present, the value of rdfs:label property (or some subproperty thereof) is used. A link is also provided to the Web resource itself: this is useful, for example, if the node described is a real Web page or some other document that can be rendered in the Web browser.

2. The properties are visualized. The properties of the node are split into clusters corresponding to the most specific non-overlapping classes of the node, using associated domain specifications. Any "leftover" properties are assumed to be associated with rdfs:Resource and are visualized last. The classes that are considered include both asserted and inferred classes of the node. Finally, inbound arcs are visualized separately.

   Each cluster of properties can be associated with a special markup generator that can emit class-specific visualizations. The markup generator associated with rdfs:Resource is assumed to be the default. A subset of the Fresnel vocabulary [23] for customized presentations will be implemented later.

3. OINK tracks the path the user has taken to navigate to any specific node; this history is shown and allows the user to invoke queries generated automatically from the navigational path (Figure 9.4).

Any statements that have been *reified* [141, section 4] have an indicator next to them (as if a footnote or endnote had been associated with them). Clicking the indicator takes one to the page representing the reified statement.

Finally, the servlet will be able to generate structurally simplified pages when it detects that the requesting client is a limited, small device, such as a mobile phone.[46] Browsing semantic data on a low-bandwidth, small-screen device is appealing, since the information is in a more compact and possibly more succinct form.

In order to be able to load various "legacy" data into OINK, a transformation engine is used that allows data format transformations to be written in XSLT or as CGIs. In a loose sense the engine takes the form of an HTTP *proxy*. OINK makes HTTP requests to the proxy, designating not only the original data source but also the desired transformation. The proxy loads the data, applies the transformation, and hands the resulting RDF to OINK. The purpose of the transformation engine, from the architectural viewpoint, is to separate data format conversions from the rest of the system. OINK is a "pure" RDF application in the sense that it does not know anything about other data formats.

9.2.3 Lessons Learned from OINK

In a metaphorical sense, OINK brings the Semantic Web "closer" to the user. Various tools can be applied to the data in the cache, without concern of the "outside world" (e.g., one could locally close the world with respect to reasoning). Generally, the cache separates issues of data access and cache management from higher-level considerations such as reasoning and other "ontological" mechanisms.

OINK has proven useful in visualizing RDF data. Despite the typical Semantic Web focus on machine interpretation of information, there are benefits to letting users explore semantic

[46] The current prototype only knows about Nokia S60 phones.

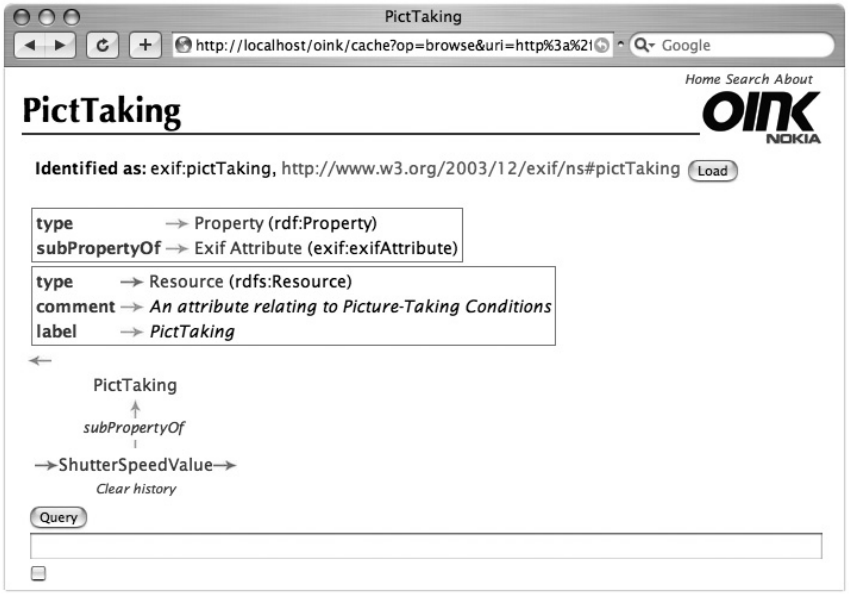

Figure 9.4: Visualization of navigation history and automatically generated queries

representations. These benefits extend beyond tasks like debugging ontologies; with suitable visualization, data integrated from multiple sources can be used for building new applications.

OINK was originally created as an easy-to-use debugging tool for RDF data; subsequent development towards customized visualizations suggests that it could be seen as an application platform. There is a large class of applications that are *little more* than queries to Semantic Web data, associated with suitable visualization.

9.3 Social Networks and Organizational Structures

There are information integration applications that can be built using Wilbur and OINK that are entirely data-driven and in a sense require no customizations to either software package to make the specific scenarios work – that is, any semantics of the applications are encapsulated within the data they use.

Figure 9.5: Exporting a FoaF profile from the phone

The following application scenario demonstrates the merging of data to identify participants in a teleconference [139]. The following data sources were used:

- *Zakim*,[47] a Semantic Web agent, helps facilitate W3C meetings using IRC and an audio teleconference bridge. When a phone call is made into the conference bridge, Zakim registers the participant's presence and identifies the participant by the caller's telephone number. If the telephone number was previously known to Zakim, it identifies the participant using a previously given name (if available). Zakim makes available as RDF the data it collects about active teleconferences and participants. This data includes the name of each participant and a hash (for privacy reasons) of the caller's telephone number.

- *OinkIt*, an application that creates FoaF [32] profiles from the contact book on a Nokia S60 phone. OinkIt uses the `foaf:knows` relationship for the entries in the contact book and uses the same hash function as Zakim to include the contacts' telephone numbers. OinkIt exports the resulting RDF from the phone to OINK via the XML-RPC interface. OINK's XML-RPC interface returns a URI of a page in OINK from which the user can start browsing the loaded data (see Figure 9.5).

[47] `http://www.w3.org/2001/12/zakim-irc-bot.html`

OINK's integrated view of data from Zakim and the FoaF profiles (from some of the participants) helps in identifying participants who were not previously known to Zakim. The RDF^{++} reasoning support in OINK makes this possible; the semantics of the hashed telephone number property are declared to be OWL inverse functional; that is, two contacts who share the same hashed telephone number are declared to be the same individual.[48] OINK can therefore merge data about individuals from the phone contact books with data about individuals in Zakim's teleconference participant list.

In another experiment, an extension of the FoaF schema was created, dubbed "CoaC" (for "Colleague-of-a-Colleague"), suited for representing corporate organizational structures. Subsequently, corporate LDAP directory for employees' contact information was queried, and the query results transformed (using a simple script) into RDF data that uses the CoaC schema; URIs, based on "employee ID" numbers, were generated for all people. WILBUR's query language WILBURQL now allows (arbitrarily deep) organizational hierarchies to be queried. For example, assuming that the URI `http://www.nokia.com/coac?id=10050531` represents a person with the name "Ora Lassila", the following WILBUR query can be used to retrieve the names of all people who report to Ora (directly or indirectly):

```
(all-values !"http://www.nokia.com/coac?id=10050531"
            '(:seq (:rep+ (:inv !coac:reportsTo)) !foaf:name))
```

Similarily, the following query retrieves the names of the secretaries of all the people above Ora in the reporting line:

```
(all-values !"http://www.nokia.com/coac?id=10050531"
            '(:seq (:rep+ !coac:reportsTo) !coac:hasSecretary !foaf:name))
```

The combination of data transformations (embodied in a *transformation proxy* as illustrated in Figure 9.3) and WILBURQL queries allows legacy data to be queries in ways most traditional tools do not allow.

[48] This works well enough in practice today and is getting more accurate as more and more people rely exclusively on their mobile phones for voice communications.

9.4 Service Composition and Substitution

The ability to automatically discover, compose and invoke Web services is an important component (and *benefit*) of the Semantic Web, and a key enabler of the anticipated "serendipity" of agent behavior on the Semantic Web. Upper ontologies for semantic annotation of Web services have started to appear, making it possible to apply Web services in the Semantic Web context. One of these is OWL-S[49] [8] which in itself is quite complex, and motivates the question whether a simpler ontology could be sufficient for some uses of Semantic Web services [137].

The attempts to simplify the OWL-S/DAML-S model have predominantly been based on a single assumption, namely that the concept of *complex process* could be eliminated. In practice, this means that all described services are in effect "black boxes", i.e., they have input and output parameters, but nothing more is known of their internal workings, and their potential components cannot be invoked separately. Services were further restricted to only have a single input parameter and a single output parameter (extending to multiple inputs will make the search algorithm a bit more complicated, but will generally not change the principle presented).

An approach to Web service composition for simple workflows can now be proposed, consisting of the aforementioned services with a single input and a single result – typically these services would be what [153] refers to as "information-providing services" (conversions, queries, etc.) but the possibility of some type of (real world) side effects is not excluded. The principal aim is to provide for *service substitution* in the face of changing availability of services used.

It is assumed that the following services and ontologies exist and are available:

- A service discovery mechanism that accepts *partial* OWL-S service descriptions as

[49] Previously called "DAML-S".

queries, and returns *complete* descriptions of services discovered. No assumptions are made about how this mechanism actually finds services nor where the descriptions of these services would be located; details of this mechanism are beyond the scope of this thesis.

- One or more ontologies for *classifying* services; that is, classification taxonomies describing *what* services actually do.

- Ontologies defining concepts and datatypes in the domains relevant to the services used (for example, when interested in services that deal with geographical phenomena – such as various map services – one would assume the existence of an ontology that defines concepts such as *address*, *GPS coordinates*, *region*, *containment of regions*, etc.).

It is further assumed that the ontologies and knowledge representation used could rely on (some form of) logic for simple expression of concept subsumption, specifically because of the assumption that the "quality" of the discovery results (i.e., the degree of the match) can be classified according to the discrete scale presented in [170]: *exact*, *plugin*, *subsumes*, and *fail*.[50] It is anticipated that more complex logical relations will not be needed. In this particular approach, only RDF(S) was used for describing services, albeit it has certain shortcomings.

9.4.1 "DAML-S Lite" – a Subset of OWL-S/DAML-S

For service description a highly simplified subset of OWL-S, written in RDF, is used. The rough idea is that the concept of a complex process is eliminated. Each service described is assumed to be a black box, with inputs and outputs which are described in terms of their datatypes (expressed using RDF classes). Effectively this simplification eliminates process

[50]The additional match category of *intersects* as described in [146] is not considered.

models from the ontology. Each service is also classified using some hypothetical service taxonomy; in practice this means that each service profile is an instance of an appropriate service class (this is a marked difference from OWL-S, since service classification now happens by subclassing the `ServiceProfile` class, as opposed to using a separate value for the `serviceCategory` property of service profiles). Furthermore, for the purposes of the prototype described in this thesis, it is assumed that each service only has one input and one output. Compositions of services like this form linear sequences of operations.

9.4.2 Service Substitution

The particular "use case" of interest is that of *service substitution*, that is, a situation where a service needs to be replaced with an "equivalent" service (given *some* notion of equivalence). For example, imagine a system that makes use of a service x which then becomes unavailable for one reason or another; what if the system was now *automatically* able to discover a set of substituting services $\{y_i\}$ which, when assembled into a linear workflow would provide an equivalent service comparable to x ("linear workflow" is understood to mean a situation where the output of service y_i is "fed" to the input of service y_{i+1}, for $1 \leq i \leq n-1$)? A typical situation is one where many of the services in the workflow perform conversions or other mediation of input and output parameters.

With the emerging mainstream deployment of architectures for *service-oriented computing* [172], the *availability* of services will be of critical importance. In scenarios where near-100% availability cannot be guaranteed, the idea of automatic substitution of services (that have become unavailable) becomes very compelling.

There can naturally be several reasons for a service to become unavailable. In addition to a service going "off-line" for one reason or another (server crash, partial network outage, etc.), the preconditions for invoking a particular service may become invalid. For example, a service

may be tied to a specific geographical location, and when the invoking terminal moves – it could be a mobile phone – a new service has to be discovered. More generally, in a software framework that supports *context awareness*, services could be tied to a particular *usage context*, and substitutions have to be made whenever there are relevant changes in context. Furthermore, service substitution could be used to opportunistically take advantage of the best available services (given some criteria to determine what "best" means – criteria such as cost, quality, speed, etc. could be used).

9.4.3 Composing Workflows

Given the basic use case of service substitution, a single service may be replaced by a slightly different service that potentially needs input and output parameter conversions. These conversions may be performed using either internal functions or by external services; in either case they constitute additional "workflow" to be performed before and after the invocation of the "main" service. From the viewpoint of DAML-S, whether the functionality is internal or external is only a matter of expressing the proper *grounding* of the particular service.

Workflows are composed using a breadth-first search, starting with the description of the service to be replaced. Based on the degree of match – as in [170] – in the results of service discovery, the search proceeds as follows:

- For *exact* and *plugin*, the result of the query is accepted.
- For *subsumes*, the service found is accepted, but the mismatch between the original query and its result will be used to construct further queries to convert input and output parameters, in order to compose a service that exactly matches the original. For example, given a discovered service whose output parameter has a type "temperature" and whose unit is "Celsius" when one was looking for results to be returned in "Fahrenheit", this implies that one will have to find a conversion from "Celsius" to

"Fahrenheit".

- For *fail*, re-perform the query with relaxed input and output parameter descriptions, implying that a resulting successful match will have to be further augmented with input and/or output parameter conversions (a relaxed parameter description is one that is more general in the classification sense).

This analysis is performed for the results of each query – also when these queries are conducted as a result of previous analyses – resulting in a recursive descent tree traversal. The intermediate results of the algorithm consist of workflows expressed in terms of services and queries (the reader is here reminded that queries and concrete services – that is, results of queries – are expressed the same way). As long as an intermediate result contains "unexpanded" queries, the algorithm will keep "rewriting" the result. Any number of parallel intermediate results (= "hypotheses") may be pursued; a null result from a query causes a hypothesis to be eliminated from the search.[51] An example of how a particular weather-related information service is substituted is illustrated in Figure 9.6.

In case of large fan-out of the search – that is, when a large number of services match a query – some type of heuristic pruning could be applied on the intermediate hypotheses. The heuristic applied could be based on similar criteria as when the algorithm is used for opportunistically finding the best available services.

9.4.4 Practical Implementation Using RDF(S)

A proof-of-concept prototype of the composition algorithm has been constructed that uses RDF(S) as its description language. Because of the use of RDF(S), the actual *matching* of service descriptions is partially based on procedurally expressed semantics: service clas-

[51] Effectively, search happens in a breadth-first manner, and resembles some approaches to HTN-planning [60, 59]. The interested reader is referred to [137] for a formal description of the algorithm.

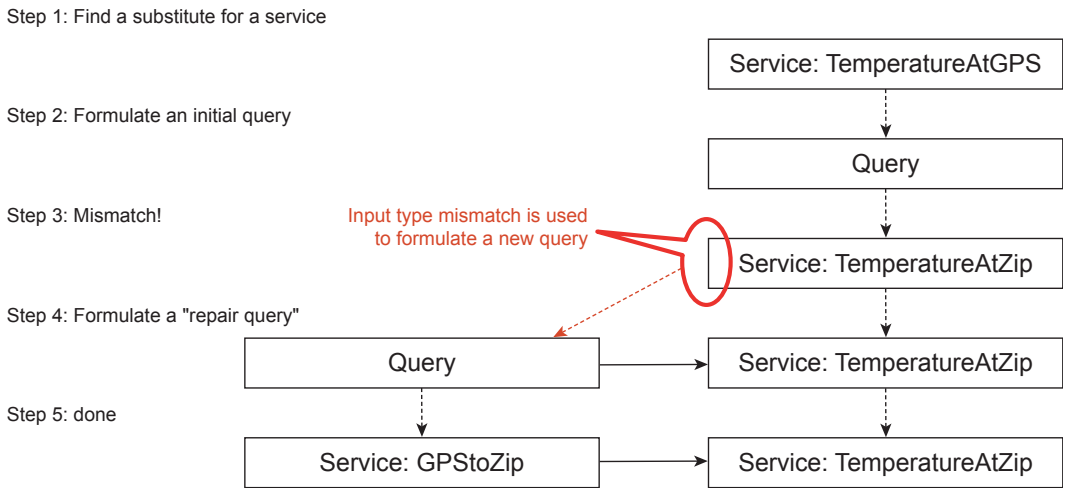

Figure 9.6: Substitution of a Weather Information Service

sification, input and output are considered separately[52] – two service descriptions match if all three components, correspondingly, are "compatible", where compatibility is defined in terms of the degree of match. As matching ultimately comes down to the comparison of RDF datatypes, the following logic is used:

$$match(a,b) = \begin{cases} exact & \text{if } a = b \\ subsumes & \text{if } b \subset a \\ plugin & \text{if } a \subset b \\ fail & \text{otherwise} \end{cases}$$

Relaxation is in this implementation performed by "walking" up the RDF class tree; it can also be done in a single step by replacing any type with the root of the class tree `rdfs:Resource`. Furthermore, given that sequences of conversions can result in loops, the prototype also performs simple pattern matching of hypotheses and eliminates those where loops are seen forming.

[52]The use of a more expressive language (say, OWL DL) would allow one to express service descriptions (including parameters) as single class expressions, in turn allowing one to rely more on the reasoning engine for matching.

9.4.5 Lessons Learned from Service Substitution

The ability to automatically replace services in the event of service failure (due to server or connectivity outage, or change in service invocation preconditions) is an important aspect of building robust and autonomous agents for the Semantic Web. The approach described achieves this for limited services and linear workflows. Useful compositions can be created without sophisticated planning technologies given the limitations described earlier (including the use of a subset of the OWL-S ontology).

The prototype implementation naturally has some shortcomings (perhaps most notably the lack of heuristic pruning during search). The use of (mere) RDF in this experiment reveals that certain DL features would be useful, especially the availability of *class expressions* – this would have allowed expression of entire service descriptions as classes, without the need to resort to a procedurally encoded extra logic for service interface matching.

10 Conclusions

> *"I need new ideas for the web. People are already getting sick of reading the words 'Some Pig!'"*
>
> – Charlotte A. Cavatica[53]

This dissertation has discussed the use of complex representational structures in software; specifically, the "representational structures" are data that uses the formalisms and principles developed for the *Semantic Web*.

The Semantic Web is a field of study inspired – in the long term – to have computers work *on their human users' behalf*. This is a lofty goal and a vision whose realization may take a long time. In the shorter term, technologies associated with the Semantic Web can be used to ease the implementation of automation and automated tasks (on the Web) as well as improve the interoperability of information systems – especially the "unscripted", unanticipated interoperability that today largely eludes automated systems.

In many ways, the "story" of this dissertation applies more generally to *knowledge representation* (KR) techniques and their application; it could also be argued that the Semantic Web is little else besides the application of KR techniques, on a Web-wide scale or merely in the context of Web technologies (i.e., either *for* Web technologies or *using* Web technologies).

The challenge with KR techniques in general – and Semantic Web technologies in particular – is that they are foreign to most developers of mainstream software applications, and consequently the adoption of these technologies presents an uphill battle. The peculiar characteristic of the Semantic Web where any problem, when sufficiently elaborated, may not need Semantic Web technologies as its solution, makes it even harder to convince the software community to adopt these technologies. The approach adopted in this dissertation,

[53] From *Charlotte's Web* [205]; special thanks to Marcia Lassila for coming up with this.

therefore, has been to not only make it easy to programmatically access complex representational structures, but also to the extent possible *hide* associated reasoning mechanisms from application logic.

The approach presented consists of three parts, and is applied in the context of the RDF(S) language and its data model of directed, labeled graphs:

1. A query language is introduced for expressing complex relationships within a graph and consequently useful in "reaching" vertices of interest in the graph (Chapter 5).

2. A mechanism is presented where queries in the aforementioned query language can be transformed in a way that their results reflect not only the original graph but also all the entailments of the graph (Chapter 6).

3. Extensions are introduced to RDF(S) that mitigate certain shortcomings of the language (Chapter 7). These extensions maintain the characteristic of the original language where no contradictions can be introduced via reasoning. Ultimately, this characteristic allows us to present a view into an RDF(S) graph where the program logic accessing the graph does not have to know that entailed edges are also included.

The main shortcoming of RDF(S), as implied by item 3 above, is that Semantic Web representations are largely predicated on objects being identified using URIs. Albeit a convenient mechanism, we are forced to admit that most "real-world" objects that we might want to describe do not have URIs, and that no unambiguous process for the selection of URIs for these objects exists. Data integration, one of the great promises of Semantic Web technology, is largely not possible unless the identity of objects can be established. Our approach essentially introduces the use of unique keys for identifying objects, and consequently operations corresponding to relational joins can be accomplished across independent data sets.

A concrete implementation of the approach has been presented (Chapter 8), as well as

examples of the usage of the approach and the corresponding implementation (Chapter 9).

As for possible directions of future work, there are two clear areas of improvement for the approach (and the corresponding software platform) described in this dissertation, both related to the expressive power of the representation and query languages. They are:

1. The expressive power of the query language could be increased to allow *conjunctive* queries. This, effectively, would push the language past SPARQL as a means of accessing Semantic Web representations. One possible approach could be to express queries as intersections of sub-queries, then optimize the query engine to not have to compute the intermediate results. More generally, as SPARQL is expected to gain popularity as the "official" query language for RDF, migration towards compatibility with this language could be seen as desirable.

2. An approach should be devised that would allow programs to deal with *contradictions* in the underlying knowledge representation. Most typically, representing *disjointness* of concepts is a very powerful modeling construct that would benefit many applications (e.g., the class "man" could be defined as being disjoint from the class "woman"). As a possible solution, the reasoner could raise an *exception* for every contradiction detected; the application program could then choose to reconcile the contradiction(s) – or accept the ramifications.

We should also consider developing similar programming abstractions for languages other than Common Lisp as the ones introduced in this dissertation.

Hypothetically, as the Semantic Web gains popularity and acceptance, the underlying mechanisms will become better known in the software community. With respect to the challenges for adoption of Semantic Web technologies, whether the outcome of the increased popularity is that the proverbial "bar" is lowered (as one might consider obvious) or raised (e.g., developers migrating towards representation languages of higher expressive power) is unclear.

References

[1] Serge Abiteboul. 1997. Querying Semi-Structured Data. In: ICDT '97: Proceedings of the 6th International Conference on Database Theory, pages 1–18. Springer-Verlag, London, UK. ISBN 3-540-62222-5.

[2] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. 1997. The Lorel Query Language for Semistructured Data. International Journal on Digital Libraries 1, no. 1, pages 68–88.

[3] Rakesh Agrawal. 1988. Alpha: An Extension of Relational Algebra to Express a Class of Recursive Queries. IEEE Transactions on Software Engineering 14, no. 7, pages 879–885.

[4] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. 1986. Compilers: Principles, Techniques and Tools. Addison-Wesley.

[5] Alfred V. Aho and Jeffrey D. Ullman. 1979. Universality of data retrieval languages. In: POPL '79: Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, pages 110–119. ACM Press.

[6] Faisal Q. Alkhateeb, Jean-François Baget, and Jérôme Euzenat. 2005. Complex Path Queries for RDF Graphs. In: Riichiro Mizoguchi (editor), ISWC 2005 Poster & Demonstration Proceedings.

[7] Anupriya Ankolekar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David Martin, Sheila A. McIllraith, Srini Narayanan, Massimo Paolucci, Terry Payne, Tran Cao Son, Katia Sycara, and Honglei Zeng. 2002. DAML-S: A Semantic Markup Language for Web Services. In: Isabel F. Cruz, Stefan Decker, Jérôme Euzenat, and Deborah L. McGuinness (editors), The Emerging Semantic Web, Selected Papers from the First

Semantic Web Working Symposium, volume 75 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

[8] Anupriya Ankolekar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, Drew McDermott, David Martin, Sheila A. McIllraith, Srini Narayanan, Massimo Paolucci, Terry Payne, and Katia Sycara. 2002. DAML-S: Web Service Description for the Semantic Web. In: Ian Horrocks and James Hendler (editors), The Semantic Web - ISWC 2002, 1st International Semantic Web Conference, volume 2342 of *Lecture Notes in Computer Science*, pages 348–363. Springer Verlag.

[9] Jari Arkko, Vesa Hirvisalo, Juha Kuusela, Esko Nuutila, and Markku Tamminen. 1989. Rule-Based Expression Mechanisms for Procedural Languages. Computational Intelligence 5, no. 4.

[10] Danny Ayers. 2005. SPARQL Query Language for RDF. Article and discussion on the Programming Languages Weblog "Lambda the Ultimate" (available at http://lambda-the-ultimate.org/node/view/549).

[11] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider (editors). 2003. The Description Logic Handbook – Theory, Implementation and Applications. Cambridge University Press. ISBN 0521781760.

[12] Jean-François Baget. 2005. RDF Entailment as a Graph Homomorphism. In: Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen (editors), The Semantic Web – ISWC 2005, 4th International Semantic Web Conference, number 3729 in Lecture Notes in Computer Science, pages 82–96. Springer-Verlag.

[13] Dave Beckett. 2004. RDF/XML Syntax Specification (Revised). W3C Recommendation, World Wide Web Consortium, Cambridge, MA. URL `http://www.w3.org/TR/rdf-syntax-grammar/`.

[14] Dave Beckett and Jan Grant. 2003. Mapping Semantic Web Data with RDBMSes. SWAD-Europe Deliverable 10.2, World Wide Web Consortium. URL http://www.w3.org/2001/sw/Europe/reports/scalable_rdbms_mapping_report/.

[15] David Beckett. 2001. The Design and Implementation of the Redland RDF Application Framework. In: WWW '01: Proceedings of the 10th international conference on World Wide Web, pages 449–456. ACM Press, New York, NY. ISBN 1-58113-348-0.

[16] Boualem Benatallah, Mohand-Said Hacid, Christophe Rey, and Farouk Toumani. 2003. Request Rewriting-Based Web Service Discovery. In: Dieter Fensel, Katia Sycara, and John Mylopoulos (editors), The Semantic Web - ISWC 2003, 2nd International Semantic Web Conference, volume 2870 of *Lecture Notes in Computer Science*, pages 242–257. Springer-Verlag.

[17] Tim Berners-Lee. 1998. Notation 3. Design Note, World Wide Web Consortium. URL http://www.w3.org/DesignIssues/Notation3.html.

[18] Tim Berners-Lee. 2000. cwm – a general purpose data processor for the semantic web. URL http://www.w3.org/2000/10/swap/doc/cwm.html.

[19] Tim Berners-Lee, Yuhsin Chen, Lydia Chilton, Dan Connolly, Ruth Dhanaraj, James Hollenbach, Adam Lerer, and David Sheets. 2006. Tabulator: Exploring and Analyzing linked data on the Semantic Web. In: Proceedings of the The 3rd International Semantic Web User Interaction Workshop (SWUI06). Athens, GA.

[20] Tim Berners-Lee, Roy Fielding, and Larry Masinter. 1998. Uniform Resource Identifiers (URI): Generic Syntax. Internet Draft Standard RFC 2396, IETF.

[21] Tim Berners-Lee, James Hendler, and Ora Lassila. 2001. The Semantic Web. Scientific American 284, no. 5, pages 34–43.

[22] Robert Biddle and Ewan Tempero. 1996. Explaining Inheritance: A Code Reusability Perspective. In: SIGCSE '96: Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education, pages 217–221. ACM Press, New York, NY. ISBN 0-89791-757-X.

[23] Christian Bizer, Ryan Lee, and Emmanuel Pietriga. 2005. Fresnel – A Browser-Independent Presentation Vocabulary for RDF. In: End User Semantic Web Interaction Workshop at the 4th International Semantic Web Conference. Galway, Ireland.

[24] Alex Borgida and Peter Patel-Schneider. 1994. A Semantics and Complete Algorithm for Subsumption in the CLASSIC Description Logic. Journal of Artificial Intelligence Research 1, pages 277–308.

[25] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. 2000. Simple Object Access Protocol (SOAP) 1.1. W3C Note, World Wide Web Consortium, Cambridge, MA. URL `http://www.w3.org/TR/2000/NOTE-SOAP-20000508`.

[26] Ronald J. Brachman. 1977. A Structural Paradigm for Representing Knowledge. Ph.D. thesis, Harvard University.

[27] Ronald J. Brachman. 1979. On the Epistemological Status of Semantic Networks. In: N.V. Findler (editor), Associative Networks: Representation and Use of Knowledge by Computers, pages 3–50. Academic Press, New York, NY.

[28] Ronald J. Brachman and Hector J. Levesque (editors). 1985. Readings in Knowledge Representation. Morgan Kaufmann, San Mateo, CA.

[29] Ronald J. Brachman and James G. Schmolze. 1985. An Overview of the KL-ONE Knowledge Representation System. Cognitive Science 9, no. 2, pages 171–216.

[30] Tim Bray, Dave Hollander, and Andrew Layman. 1999. Namespaces in XML. W3C Recommendation, World Wide Web Consortium. URL http://www.w3.org/TR/REC-xml-names/.

[31] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. 1998. Extensible Markup Language (XML) 1.0. W3C Recommendation, World Wide Web Consortium. URL http://www.w3.org/TR/1998/REC-xml-19980210.

[32] Dan Brickley and Libby Miller. 2004. FOAF Vocabulary Specification. http://xmlns.com/foaf/0.1/.

[33] Daniel Brickley and R.V. Guha. 2003. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Working Draft, World Wide Web Consortium.

[34] Jeen Broekstra. 2005. Storage, Querying and Inferencing for Semantic Web Languages. Ph.D. thesis, Vrije Universiteit, Amsterdam, Netherlands.

[35] Jeen Broekstra and Arjohn Kampman. 2003. Inferencing and Truth Maintenance in RDF Schema: Exploring a naive practical approach. In: Workshop on Practical and Scalable Semantic Systems (PSSS). Sanibel Island, FL.

[36] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. 1999. Rewriting of Regular Expressions and Regular Path Queries. In: PODS '99: Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pages 194–204. ACM Press, New York, NY, USA. ISBN 1-58113-062-7.

[37] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. 2000. Query Processing using Views for Regular Path Queries with Inverse. In: Proceedings of the Nineteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2000), pages 58–66. ACM Press, New York.

[38] 1986. Knowledge Craft User's Manual. Technical report, Carnegie Group, Inc., Pittsburgh, PA.

[39] Vinay Chaudhri, Adam Farquhar, Richard Fikes, Peter Karp, and James Rice. 1998. OKBC: A Programmatic Foundation for Knowledge Base Interoperability. In: Proceedings of the National Conference on Artificial Intelligence (AAAI).

[40] Weidong Chen, Michael Kifer, and David S. Warren. 1993. HiLog: a Foundation for Higher-Order Logic Programming. Journal of Logic Programming 15, no. 3, pages 187–230.

[41] Shigeru Chiba. 1995. A Metaobject Protocol for C++. In: Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'95), pages 285–299.

[42] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. 2001. Web Services Description Language (WSDL) 1.1. W3C Note, World Wide Web Consortium, Cambridge, MA. URL `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`.

[43] James Clark. 1999. XSL Transformations (XSLT) Version 1.0. W3C Recommendation, World Wide Web Consortium, Cambridge, MA. URL `http://www.w3.org/TR/xslt`.

[44] James Clark and Steve DeRose. 1999. XML Path Language (XPath) Version 1.0. W3C Recommendation, World Wide Web Consortium. URL `http://www.w3.org/TR/1999/REC-xpath-19991116.html`.

[45] E. F. Codd. 1970. A relational model of data for large shared data banks. CACM 13, no. 6, pages 377–387.

[46] Wolfram Conen, Reinhold Klapsing, and Eckhart Köppen. 2001. RDF M&S revisited:

From Reification to Nesting, from Containers to Lists, from Dialect to pure XML. In: Proceedings of the First Semantic Web Working Symposium.

[47] Mariano P. Consens and Alberto O. Mendelzon. 1990. GraphLog: a visual formalism for real life recursion. In: Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. ACM Press.

[48] John Cowan and Richard Tobin. 2004. XML Information Set (Second Edition). W3C Recommendation, World Wide Web Consortium, Cambridge, MA. URL `http://www.w3.org/TR/2004/REC-xml-infoset-20040204/`.

[49] Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. 1987. A graphical query language supporting recursion. In: Proceedings of the ACM SIGMOD Annual Conference on Management of Data, pages 323–330.

[50] Bogdan Czejdo, Christoph F. Eick, and Malcolm Taylor. 1993. Integrating Sets, Rules, and Data in an Object-Oriented Environment. IEEE Expert: Intelligent Systems and Their Applications 8, no. 1, pages 59–66.

[51] Jos de Brujin, Dieter Fensel, Uwe Keller, Michael Kifer, Holger Lausen, Reto Krummenacher, Axel Polleres, and Livia Predoiu. 2005. Web Service Modeling Language (WSML). W3C Member Submission, World Wide Web Consortium, Cambridge, MA. URL `http://www.w3.org/Submission/WSML/`.

[52] Mike Dean, Guus Schreiber, Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. 2004. OWL Web Ontology Language Reference. W3C Recommendation, World Wide Web Consortium, Cambridge, MA. URL `http://www.w3.org/TR/owl-ref/`.

[53] Thomas L. Dean and Drew McDermott. 1987. Temporal Data Base Management. Artificial Intelligence 32, no. 1, pages 1–55.

[54] Stefan Decker, Dan Brickley, Janne Saarela, and Jürgen Angele. 1998. A query and inference service for RDF. In: W3C Query Languages Workshop (QL'98).

[55] Stefan Decker and Martin Frank. 2004. The Social Semantic Desktop. Technical Report DERI-TR-2004-05-02, DERI.

[56] J. Doyle. 1987. A Truth Maintenance System. In: Readings in Nonmonotonic Reasoning, pages 259–279. Morgan Kaufmann Publishers, San Francisco, CA, USA. ISBN 0-934613-45-1.

[57] 2004. Dublin Core Metadata Element Set, Version 1.1: Reference Description. DCMI Recommendation, Dublin Core Metadata Initiative. URL `http://dublincore.org/documents/dces/`.

[58] Edd Dumbill. 2002. XML Watch: Finding friends with XML and RDF. URL `http://www-106.ibm.com/developerworks/xml/library/x-foaf.html`.

[59] Kutluhan Erol, James Hendler, and Dana Nau. 1994. Semantics for Hierarchical Task Network Planning. Technical report, Department of Computer Science, University of Maryland - College Park.

[60] Kutluhan Erol, James Hendler, and Dana S. Nau. 1994. HTN Planning: Complexity and Expressivity. In: Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94), volume 2, pages 1123–1128. AAAI Press/MIT Press, Seattle, Washington, USA. ISBN 0-262-51078-2.

[61] Leonhard Euler. 1736. Solutio problematis ad geometriam situs pertinentis. Comment. Academiae Sci. I. Petropolitanae 8, pages 128–140.

[62] Adam Farquhar, Richard Fikes, and James Rice. 1997. The Ontolingua Server: a Tool for Collaborative Ontology Construction. International Journal of Human-Computer

Studies 46.

[63] Richard Fikes, Pat Hayes, and Ian Horrocks. 2004. OWL-QL: A Language for Deductive Query Answering on the Semantic Web. KSL Technical Report 03-14, Knowledge Systems Laboratory, Stanford University.

[64] Richard Fikes, Pat Hayes, Ian Horrocks, Harold Boley, Mike Dean, Benjamin Grosof, Frank van Harmelen, Sandro Hawke, Jeff Heflin, Ora Lassila, Deb McGuinness, Peter Patel-Schneider, and Lynn Andrea Stein. 2003. DAML Query Language. Technical report, DARPA Agent Markup Language Program. URL `http://www.daml.org/2003/04/dql/`.

[65] Richard Fikes and Tom Kehler. 1985. The Role of Frame-Based Representation in Reasoning. Communications of the ACM 28, no. 9, pages 904–920.

[66] Richard Fikes and Deborah L. McGuinness. 2001. An Axiomatic Semantics for RDF, RDF Schema, and DAML+OIL. Technical Report KSL-01-01, Knowledge Systems Laboratory, Stanford University. URL `http://www.ksl.stanford.edu/KSL_Abstracts/KSL-01-01.html`.

[67] Charles L. Forgy. 1982. A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence 19, no. 1, pages 17–37.

[68] Mark S. Fox. 1983. Constraint-Directed Reasoning: a Case Study of Job-Shop Scheduling. Ph.D. thesis, Computer Science Department, Carnegie Mellon University., Pittsburgh, PA.

[69] Mark S. Fox. 1985. Knowledge Representation for Decision Support. In: Methlie and Sprague (editors), Knowledge Representation for Decision Support Systems. Elsevier.

[70] Michael R. Genesereth. 1995. Knowledge Interchange Format Specification. ANSI X3T2 working draft, American National Standards Institute.

[71] Matthew L. Ginsberg. 1991. Knowledge interchange format: the KIF of death. AI Magazine 12, no. 3, pages 57–63.

[72] Chris Goad. 2001. Describing Computation within RDF. In: Proceedings of the First Semantic Web Working Symposium. Stanford University.

[73] "The Dispersion of the Nations at Babel". Genesis 11:1–9.

[74] Roy Goldman and Jennifer Widom. 1997. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In: VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases, pages 436–445. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 1-55860-470-7.

[75] Roy Goldman and Jennifer Widom. 1999. Interactive Query and Search in Semistructured Databases. In: WebDB '98: Selected papers from the International Workshop on The World Wide Web and Databases, pages 52–62. Springer-Verlag, London, UK. ISBN 3-540-65890-4.

[76] Georg Gottlob, Christoph Koch, and Reinhard Pichler. 2002. Efficient Algorithms for Processing XPath Queries. In: Proc. VLDB 2002.

[77] Gösta Grahne and Alex Thomo. 2003. Query Containment and Rewriting Using Views for Regular Path Queries under Constraints. In: PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pages 111–122. ACM Press, New York, NY, USA. ISBN 1-58113-670-6.

[78] T. R. Gruber. 1993. A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition 5, no. 2, pages 199–220.

[79] R. V. Guha and Patrick Hayes. 2003. LBase: Semantics for Languages of the Semantic Web. W3C Working Group Note, World Wide Web Consortium, Cambridge, MA. URL http://www.w3.org/TR/lbase/.

[80] R.V. Guha and Tim Bray. 1997. Meta Content Framework Using XML. W3C Member Submission, World Wide Web Consortium, Cambridge, MA. URL http://www.w3.org/TR/NOTE-MCF-XML-970624/.

[81] R.V. Guha, Ora Lassila, Eric Miller, and Dan Brickley. 1998. Enabling Inferencing. In: W3C Query Languages Workshop (QL'98). World Wide Web Consortium. URL http://www.w3.org/TandS/QL/QL98/pp/enabling.html.

[82] Claudio Gutierrez, Carlos Hurtado, and Alberto O. Mendelzon. 2004. Foundations of Semantic Web Databases. In: PODS '04: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pages 95–106. ACM Press, New York, NY, USA. ISBN 158113858X.

[83] Peter Haase, Jeen Broekstra, Andreas Eberhart, and Raphael Volz. 2004. A Comparison of RDF Query Languages. Technical report, Institute AIFB, University of Karlsruhe.

[84] William Rowan Hamilton. 1856. Memorandum respecting a new system of roots of unity. Philosophical Magazine 12.

[85] Patrick Hayes. 2003. RDF Semantics. W3C Working Draft, World Wide Web Consortium. URL http://www.w3.org/TR/2003/WD-rdf-mt-20030123/.

[86] Patrick Hayes. 2004. RDF Semantics. W3C Recommendation, World Wide Web Consortium. URL http://www.w3.org/TR/rdf-mt/.

[87] Patrick J. Hayes. 1974. Some Problems and Non-Problems in Representation Theory. In: Proceedings of the AISB Summer Conference, pages 63–79. University of Sussex.

[88] Patrick J. Hayes. 1979. The Logic of Frames. In: D. Metzing (editor), Frame Conceptions and Text Understanding, pages 46–61. Walter de Gruyter and Co.

[89] Dominique Hazaël-Massieux and Dan Connolly. 2005. Gleaning Resource Descriptions from Dialects of Languages (GRDDL). W3C Team Submission, World Wide Web Consortium, Cambridge, MA. URL http://www.w3.org/TeamSubmission/2005/SUBM-grddl-20050516/.

[90] Jeff Heflin. 2001. Towards the Semantic Web: Knowledge Representation in a Dynamic, Distributed Environment. Ph.D. thesis, University of Maryland, College Park, MD.

[91] James Hendler. 2001. Agents and the Semantic Web. IEEE Intelligent Systems 16, no. 2, pages 30–37.

[92] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. 2004. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission, World Wide Web Consortium. URL http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/.

[93] Gerard Huet and Derek C. Oppen. 1980. Equations and Rewrite Rules: A Survey. Technical Report CS-TR-80-785, Stanford University, Stanford, CA.

[94] Edmund Husserl. 1900/01. Logische Untersuchungen. Niemeyer.

[95] Juha Hynynen and Ora Lassila. 1989. On the Use of Object-Oriented Paradigm in a Distributed Problem Solver. AI Communications 2, no. 3, pages 142–151.

[96] David Hyunh, Stefano Mazzocchi, and David Karger. 2005. Piggy bank: Experience the Semantic Web Inside Your Web Browser. In: Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen (editors), The Semantic Web – ISWC 2005, 4th International Semantic Web Conference. Springer-Verlag.

[97] 1985. KEE^{TM} Software Development System User's Manual. Technical report, IntelliCorp, Inc., Mountain View, CA.

[98] R. J. K. Jacob and J. N. Froscher. 1990. A Software Engineering Methodology for Rule-Based Systems. IEEE Transactions on Knowledge and Data Engineering 2, no. 2, pages 173–189.

[99] Ian Jacobs and Norman Walsh. 2004. Architecture of the World Wide Web, Volume One. W3C Recommendation, World Wide Web Consortium, Cambridge, MA. URL `http://www.w3.org/TR/webarch/`.

[100] H. V. Jagadish. 1989. Incorporating hierarchy in a relational model of data. In: SIGMOD '89: Proceedings of the 1989 ACM SIGMOD international conference on Management of data, pages 78–87. ACM Press. ISBN 0-89791-317-5.

[101] Nicholas R. Jennings and Michael J. Wooldridge. 1998. Agent Technology: Foundations, Applications, and Markets. Springer-Verlag.

[102] ISO/IEC JTC1/SC34. 1999. Topic Maps. ISO/IEC International Standard 13250, International Organization for Standardization.

[103] Lalana Kagal. 2004. A Policy-Based Approach to Governing Autonomous Behavior in Distributed Environments. Ph.D. thesis, Department of Computer Science and

Electrical Engineering, University of Maryland Baltimore County, Baltimore, MD.

[104] Lalana Kagal, Jim Parker, Harry Chen, Anupam Joshi, and Tim Finin. 2003. Security, Privacy and Trust in Mobile Computing Environments, chapter Security and Privacy Aspects. CRC Press.

[105] P. D. Karp. 1992. The Design Space of Frame Knowledge Representation Systems. Technical Report 520, SRI International Artificial Intelligence Center.

[106] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. 2002. RQL: A Declarative Query Language for RDF. In: Proceedings of the 11th International Conference on the World Wide Web.

[107] G. Karvounarakis, A. Magganaraki, S. Alexaki, V. Christophides, D. Plexousakis, M. Scholl, and K. Tolle. 2003. Querying the Semantic Web with RQL. Computer Networks 42, no. 5, pages 617–640.

[108] Yarden Katz, Kendall Clark, and Bijan Parsia. 2005. Pychinko: A Native Python Rule Engine. In: International Python Conference 05.

[109] Rohit Khare. 2006. Microformats: The Next (Small) Thing on the Semantic Web? IEEE Internet Computing 10, no. 1, pages 68–75.

[110] Deepali Khushraj and Ora Lassila. 2005. Ontological Approach to Generating Personalized User Interfaces for Web Services. In: Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen (editors), The Semantic Web – ISWC 2005, 4th International Semantic Web Conference, number 3729 in Lecture Notes in Computer Science, pages 916–927. Springer-Verlag, Galway, Ireland.

[111] Gregor Kiczales, Jim des Rivières, and Daniel G. Bobrow. 1991. The Art of the Metaobject Protocol. MIT Press.

[112] Michael Kifer, Georg Lausen, and James Wu. 1995. Logical Foundations of Object-oriented and Frame-based Languages. Journal of the ACM 42, no. 4, pages 741–843.

[113] Graham Klyne and Jeremy J. Carroll. 2004. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, World Wide Web Consortium, Cambridge, MA. URL `http://www.w3.org/TR/rdf-concepts/`.

[114] Graham Klyne, Franklin Reynolds, Chris Woodrow, Hidetaka Ohto, Johan Hjelm, Mark H. Butler, and Luu Tran. 2004. Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0. W3C Recommendation, World Wide Web Consortium, Cambridge, MA. URL `http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/`.

[115] Tim Krauskopf, Jim Miller, Paul Resnick, and Win Treese. 1996. PICS Label Distribution Label Syntax and Communication Protocols – Version 1.1. W3C Recommendation, World Wide Web Consortium, Cambridge, MA. URL `http://www.w3.org/TR/REC-PICS-labels`.

[116] David Lacey and Oege de Moor. 2001. Imperative Program Transformation by Rewriting. In: CC '01: Proceedings of the 10th International Conference on Compiler Construction, pages 52–68. Springer-Verlag, London, UK. ISBN 3-540-41861-X.

[117] Ora Lassila. 1990. Frames or Objects, or Both? In: Workshop Notes from the 8th National Conference on Artificial Intelligence (AAAI-90): Object-Oriented Programming in AI. American Association for Artificial Intelligence. Also published as Technical Report HTKK-TKO-B67, Department of Computer Science, Helsinki University of Technology.

[118] Ora Lassila. 1991. BEEF Reference Manual – A Programmer's Guide to the BEEF Frame System. Technical Report HTKK-TKO-C46, Department of Computer Sci-

ence, Helsinki University of Technology.

[119] Ora Lassila. 1992. Oliojärjestelmän laajentaminen metaobjektiprotokollan avulla (Extending an Object System using a Metaobject Protocol; in Finnish). Unpublished design document, Helsinki University of Technology.

[120] Ora Lassila. 1992. The Design and Implementation of a Frame System. Master's thesis, Faculty of Technical Physics, Helsinki University of Technology.

[121] Ora Lassila. 1995. PORK Object System Programmers' Guide. Technical Report CMU-RI-TR-95-12, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

[122] Ora Lassila. 1997. Introduction to RDF Metadata. W3C Note, World Wide Web Consortium, Cambridge, MA. URL `http://www.w3.org/TR/NOTE-rdf-simple-intro-971113.html`.

[123] Ora Lassila. 1997. PICS-NG Metadata Model and Label Syntax. W3C Note, World Wide Web Consortium, Cambridge, MA. URL `http://www.w3.org/TR/NOTE-pics-ng-metadata`.

[124] Ora Lassila. 1998. Web Metadata: A Matter of Semantics. IEEE Internet Computing 2, no. 4, pages 30–37.

[125] Ora Lassila. 2001. Enabling Semantic Web Programming by Integrating RDF and Common Lisp. In: Proceedings of the First Semantic Web Working Symposium. Stanford University.

[126] Ora Lassila. 2002. Serendipitous Interoperability. In: Eero Hyvönen (editor), The Semantic Web Kick-off in Finland – Vision, Technologies, Research, and Applications, HIIT Publications 2002-001. University of Helsinki.

[127] Ora Lassila. 2002. Taking the RDF Model Theory Out for a Spin. In: Ian Horrocks and James Hendler (editors), The Semantic Web - ISWC 2002, 1st International Semantic Web Conference, volume 2342 of *Lecture Notes in Computer Science*, pages 307–317. Springer-Verlag.

[128] Ora Lassila. 2003. Common Lisp Support for Semantic Web Programming. Invited talk at the International Lisp Conference (ILC 2003).

[129] Ora Lassila. 2004. Wilbur Query Language Comparison. Technical report, Nokia Research Center. URL `http://wilbur-rdf.sourceforge.net/2004/05/11-comparison.shtml`.

[130] Ora Lassila. 2005. Applying Semantic Web in Mobile and Ubiquitous Computing: Will Policy-Awareness Help? In: Lalana Kagal, Tim Finin, and James Hendler (editors), Proceedings of the Semantic Web Policy Workshop, 4th International Semantic Web Conference, pages 6–11. Galway, Ireland.

[131] Ora Lassila. 2005. Using the Semantic Web in Mobile and Ubiquitous Computing. In: Max Bramer and Vagan Terziyan (editors), Proceedings of the 1st IFIP WG12.5 Working Conference on Industrial Applications of Semantic Web, pages 19–25. Springer.

[132] Ora Lassila. 2006. Browsing the Semantic Web. In: 17th International Conference on Database and Expert Systems Applications (DEXA'06), pages 365–369. IEEE Computer Society, Krakow, Poland.

[133] Ora Lassila. 2006. Generating Rewrite Rules by Browsing RDF Data. In: Proceedings of the Second International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML 2006). IEEE Computer Society.

[134] Ora Lassila. 2006. Identity Crisis and Serendipity. Invited talk at

the W3C Advisory Committee meeting in Edinburgh, Scotland. URL http://www.w3.org/2006/05/IdentityCrisisAndSerendipity.pdf.

[135] Ora Lassila and Mark Adler. 2003. Semantic Gadgets: Ubiquitous Computing Meets the Semantic Web. In: Dieter Fensel, James Hendler, Wolfgang Wahlster, and Henry Lieberman (editors), Spinning the Semantic Web, pages 363–376. MIT Press.

[136] Ora Lassila, Marcel Becker, and Stephen Smith. 1996. An Exploratory Prototype for Air Medical Evacuation Re-Planning. Technical Report CMU-RI-TR-96-03, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

[137] Ora Lassila and Sapna Dixit. 2004. Interleaving Discovery and Composition for Simple Workflows. In: Semantic Web Services, AAAI Spring Symposium Series. AAAI.

[138] Ora Lassila and Deepali Khushraj. 2005. Contextualizing Applications via Semantic Middleware. In: The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous). IEEE Computer Society.

[139] Ora Lassila, Deepali Khushraj, and Ralph R. Swick. 2006. Spontaneous Collaboration via Browsing of Semantic Data on Mobile Devices. In: Stefan Decker, Jack Park, Leo Sauermann, Sören Auer, and Siegfried Handschuh (editors), Proceedings of the Semantic Desktop and Social Semantic Collaboration Workshop (SemDesk 2006), number 202 in CEUR Workshop Proceedings. CEUR-WS.org, Athens, GA.

[140] Ora Lassila and Deborah L. McGuinness. 2001. The Role of Frame-Based Representation on the Semantic Web. Technical Report KSL-01-02, Knowledge Systems Laboratory, Stanford University. URL http://www.ksl.stanford.edu/KSL_Abstracts/KSL-01-02.html.

[141] Ora Lassila and Ralph R. Swick. 1999. Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, World Wide Web Consortium.

[142] Ora Lassila, Markku Syrjänen, and Seppo Törmä. 1991. Coordinating Mutually Dependent Decisions in a Distributed Scheduler. In: Eero Eloranta (editor), Proceedings of the 4th IFIP TC5/WG5.7 International Conference on Advances in Production Management Systems – APMS'90, pages 257–264. Elsevier Science Publishers.

[143] Ora Lassila and Seppo Törmä. 1991. Using a Distributed Frame System to Implement Distributed Problem Solvers. Technical Report HTKK-TKO-B68, Department of Computer Science, Helsinki University of Technology.

[144] Ora Lassila, Seppo Törmä, and Markku Syrjänen. 1992. Designing a Distributed Frame System. In: Eero Hyvönen (editor), New Directions in Artificial Intelligence, volume 1, pages 183–192. Finnish Artificial Intelligence Society, Espoo, Finland.

[145] Andrew Layman, Edward Jung, Eve Maler, Henry S. Thompson, Jean Paoli, John Tigue, Norbert H. Mikula, and Steve De Rose. 1998. XML-Data. W3C Member Submission, World Wide Web Consortium, Cambridge, MA. URL `http://www.w3.org/TR/1998/NOTE-XML-data-0105/Overview.html`.

[146] Lei Li and Ian Horrocks. 2003. A Software Framework for Matchmaking based on Semantic Web Technology. In: Proceedings of the 12th International Conference on the World Wide Web.

[147] Barbara Liskov and Robert Scheifler. 1982. Guardians and Actions: Linguistic Support for Robust, Distributed Programs. In: Proceedings of the Ninth ACM Symposium on the Principles of Programming Languages. Association for Computing Machinery.

[148] Diane J. Litman, Peter F. Patel-Schneider, Anil K. Mishra, James M. Crawford, and Daniel L. Dvorak. 2002. R++: Adding Path-Based Rules to C++. IEEE Transactions on Knowledge and Data Engineering 14, no. 3, pages 638–658.

[149] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila A. McIllraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. 2004. OWL-S: Semantic Markup for Web Services. W3C Member Submission, World Wide Web Consortium. URL http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/.

[150] Akiyoshi Matono, Toshiyuki Amagasa, Masatoshi Yoshikawa, and Shunsuke Uemura. 2005. A Path-based Relational RDF Database. In: CRPIT '39: Proceedings of the sixteenth Australasian conference on Database technologies, pages 95–103. Australian Computer Society, Inc., Darlinghurst, Australia, Australia. ISBN 1-920-68221-X.

[151] Brian McBride. 2001. Jena: Implementing the RDF Model and Syntax Specification. In: Proceedings of the First Semantic Web Working Symposium. Stanford University.

[152] Deborah L. McGuinness and Frank van Harmelen. 2004. OWL Web Ontology Language Overview. W3C Recommendation, World Wide Web Consortium, Cambridge, MA. URL http://www.w3.org/TR/2004/REC-owl-features-20040210/.

[153] Sheila A. McIllraith, T. Son, and H. Zeng. 2001. Mobilizing the Web with DAML-Enabled Web Services. In: The Second International Workshop on the Semantic Web (SemWeb'2001) at WWW-10.

[154] Sergey Melnik. 2001. RDF API Draft. Stanford University working draft. URL http://www-db.stanford.edu/~melnik/rdf/api.html.

[155] Alberto O. Mendelzon and Peter T. Wood. 1995. Finding Regular Simple Paths in Graph Databases. SIAM Journal on Computing 24, no. 6, pages 1235–1258.

[156] 1998. Merriam-Webster's Collegiate Dictionary. Merriam-Webster.

[157] Jim Miller, Paul Resnick, and David Singer. 1996. Rating Services and

Rating Systems (and Their Machine Readable Descriptions) – Version 1.1. W3C Recommendation, World Wide Web Consortium, Cambridge, MA. URL http://www.w3.org/TR/REC-PICS-services.

[158] Marvin Minsky. 1975. A Framework for Representing Knowledge. In: Patrick Henry Winston (editor), Psychology of Computer Vision. McGraw-Hill, New York.

[159] Daniel P. Miranker. 1987. TREAT: A Better Match Algorithm for AI Production Systems. In: Kenneth Forbus and Howard Shrobe (editors), Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87), pages 42–47. AAAI Press.

[160] Daniel P. Miranker. 1990. TREAT: a New and Efficient Match Algorithm for AI Production Systems. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 0-934613-71-0.

[161] monica schraefel, Maria Karam, and Shengdong Zhao. 2003. mSpace: interaction design for user-determined, adaptable domain exploration in hypermedia. In: AH2003: Workshop on Adaptive Hypermedia and Adaptive Web-Based Systems.

[162] Chuck Murray, Nicole Alexander, Souri Das, George Eadon, and Siva Ravada. 2005. Oracle® Spatial Resource Description Framework (RDF), 10g Release 2 (10.2). Technical Report B19307-03, Oracle Corporation.

[163] Nicola Muscettola, P. Pandurang Nayak, Barney Pell, and Brian Williams. 1998. Remote Agent: To Boldly Go Where No AI System Has Gone Before. Artificial Intelligence 103, no. 1–2, pages 5–48.

[164] Guido Naudts. 2003. An Inference Engine for RDF. Master's thesis, Open University of the Netherlands.

[165] Steven R. Newcomb. 2002. Preemptive Reification. In: Ian Horrocks and James

Hendler (editors), The Semantic Web - ISWC 2002, 1st International Semantic Web Conference, volume 2342 of *Lecture Notes in Computer Science*, pages 414–418. Springer-Verlag. ISBN 3-540-43760-6.

[166] Esko Nuutila. 1990. Combining Rule-Based and Procedural Programming in the XC and XE Programming Languages. Licentiate's thesis, Helsinki University of Technology.

[167] Esko Nuutila. 1994. An Efficient Transitive Closure Algorithm for Cyclic Digraphs. Information Processing Letters 52, no. 4, pages 207–213.

[168] Uche Ogbuji. 2002. Versa, the RDF query language. URL `http://uche.ogbuji.net/tech/rdf/versa/`.

[169] 2003. User Agent Profile – Version 20-May-2003. Technical Report OMA-UAProf-v2_0-20030520-C, Open Mobile Alliance.

[170] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. 2002. Semantic Matching of Web Services Capabilities. In: Ian Horrocks and James Hendler (editors), The Semantic Web - ISWC 2002, 1st International Semantic Web Conference, volume 2342 of *Lecture Notes in Computer Science*. Springer Verlag.

[171] Yannis Papakonstantinou and Vasilis Vassalos. 1999. Query Rewriting for Semistructured Data. In: SIGMOD '99: Proceedings of the 1999 ACM SIGMOD international conference on Management of data, pages 455–466. ACM Press, New York, NY, USA. ISBN 1-58113-084-8.

[172] M.P. Papazoglou and D. Georgakopoulos. 2003. Service-Oriented Computing. Communications of the ACM 46, no. 10, pages 25–28.

[173] Peter F. Patel-Schneider, Patrick J. Hayes, and Ian Horrocks. 2004. OWL Web Ontol-

ogy Language Semantics and Abstract Syntax. W3C Recommendation, World Wide Web Consortium, Cambridge, MA.

[174] Terry Payne and Ora Lassila. 2004. Semantic Web Services (guest editors' introduction). IEEE Intelligent Systems 19, no. 4, pages 14–15.

[175] David Provost. 2004. Hurdles in the Business Case for the Semantic Web. Master's thesis, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA.

[176] Eric Prud'hommeaux and Andy Seaborne. 2005. SPARQL Query Language for RDF. W3C Working Draft, World Wide Web Consortium. URL `http://www.w3.org/TR/2005/WD-rdf-sparql-query-20050217/`.

[177] Dennis Quan and David Karger. 2004. How to Make a Semantic Web Browser. In: Proceedings of the 13th International Conference on the World Wide Web, pages 255–265. ACM Press.

[178] M. Ross Quillian. 1967. Word Concepts: A Theory and Simulation of Some Basic Semantic Capabilities. Behavioral Science 12, pages 410–430.

[179] Golden G. Richard III. 2000. Service Advertisement and Discovery: Enabling Universal Device Cooperation. IEEE Internet Computing 4, no. 5.

[180] P. Rogers and A.J. Wellings. 2000. OpenAda: A Metaobject Protocol for Ada 95. Technical Report YCS-2000-331, Department of Computer Science, University of York. URL `citeseer.nj.nec.com/431663.html`.

[181] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Cristoph Bussler, and Dieter Fensel. 2005. Web Service Modeling Ontology. Applied Ontology 1, no. 1, pages 77–106.

[182] Leo Sauermann. 2005. The Gnowsis Semantic Desktop for Information Integration. In: 1st Workshop on Intelligent Office Appliances.

[183] Leo Sauermann and Sven Schwartz. 2005. Gnowsis Adapter Framework: Treating Structured Data as Virtual RDF Graphs. In: Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen (editors), The Semantic Web – ISWC 2005, 4th International Semantic Web Conference, number 3729 in Lecture Notes in Computer Science, pages 1016–1028. Springer-Verlag.

[184] Simon Scerri, Charlie Abela, and Matthew Montebello. 2005. semantExplorer: A Semantic Web Browser. In: Pedro Isaías and Miguel Baptista Nunes (editors), IADIS International Conference WWW/Internet 2005, pages 35–42.

[185] Andy Seaborne. 2004. RDQL - A Query Language for RDF. W3C Member Submission, HP Labs.

[186] Peter Seibel. 2005. Practical Common Lisp. APress, Berkeley, CA. ISBN 1590592395.

[187] Vineet Sinha and David R. Karger. 2005. Magnet: Supporting Navigation in Semistructured Data Environments. In: SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data, pages 97–106. ACM Press, New York, NY, USA. ISBN 1-59593-060-4.

[188] Michael Sintek and Stefan Decker. 2002. TRIPLE—A Query, Inference, and Transformation Language for the Semantic Web. In: Ian Horrocks and James Hendler (editors), The Semantic Web - ISWC 2002, 1st International Semantic Web Conference, volume 2342 of *Lecture Notes in Computer Science*. Springer Verlag.

[189] Barry Smith. 1998. Basic Concepts of Formal Ontologies. In: Nicola Guarino (editor), Formal Ontology in Information Systems. IOS Press.

[190] Stephen F. Smith and Ora Lassila. 1994. Configurable Systems for Reactive Production Management. In: E. Szelke and R.M.Kerr (editors), Knowledge-Based Reactive Scheduling, volume B-15 of *IFIP Transactions*. Elsevier Science Publishers.

[191] Stephen F. Smith, Ora Lassila, and Marcel Becker. 1996. Configurable, Mixed-Initiative Systems for Planning and Scheduling. In: Austin Tate (editor), Advanced Planning Technology: Technological Achievements of the ARPA/Rome Laboratory Planning Initiative. AAAI Press, Menlo Park, CA.

[192] Richard Soley. 2000. Model Driven Architecture. White Paper, Object Management Group.

[193] Guy L. Steele. 1990. Common Lisp the Language, 2nd edition. Digital Press.

[194] Mark J. Stefik, Daniel G. Bobrow, and Kenneth M. Kahn. 1986. Integrating Access-oriented Programming into a Multiparadigm Environment. IEEE Software 3, no. 1, pages 10–18.

[195] Susie Stephens. 2005. Semantic Data Integration in the Life Sciences. White Paper, Oracle Corporation.

[196] Katia P. Sycara. 1997. James Bond and Michael Ovitz: The Secret Life of Agents. In: Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI 97), pages 770–773. American Association for Artificial Intelligence.

[197] Robert Endre Tarjan. 1981. Fast Algorithms for Solving Path Problems. Journal of the ACM 28, no. 3, pages 594–614.

[198] Yannis Theoharis, Vassilis Christophides, and Grigoris Karvounarakis. 2005. Benchmarking Database Representations of RDF/S Stores. In: Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen (editors), The Semantic Web – ISWC

2005, 4th International Semantic Web Conference, number 3729 in Lecture Notes in Computer Science, pages 685–701. Springer-Verlag.

[199] Seppo Törmä, Ora Lassila, and Markku Syrjänen. 1991. Adapting the Activity-Based Scheduling Method to Steel Rolling. In: G.Doumeingts, J.Browne, and M.Tomljanovich (editors), Computer Applications in Production and Engineering: Integration Aspects (CAPE'91). Elsevier Science Publishers.

[200] Giovanni Tummarello, Christian Morbidoni, Paolo Puliti, and Francesco Piazza. 2005. The DBin Semantic Web Platform: An Overview. In: Workshop on the Semantic Computing Initiative (SeC 2005). Chiba, Japan.

[201] Frank van Harmelen, Peter Patel-Schneider, and Ian Horrocks. 2001. Reference description of the DAML+OIL (March 2001) ontology markup language. Technical report, DARPA Agent Markup Language Program. URL `http://www.daml.org/2001/03/reference`.

[202] Eelco Visser. 2001. Stratego: A Language for Program Transformation Based on Rewriting Strategies. In: RTA '01: Proceedings of the 12th International Conference on Rewriting Techniques and Applications, pages 357–362. Springer-Verlag, London, UK. ISBN 3-540-42117-3.

[203] Eelco Visser, Zine-el-Abidine Benaissa, and Andrew Tolmach. 1998. Building Program Optimizers with Rewriting Strategies. In: Proceedings of the third ACM SIGPLAN International Conference on Functional Programming (ICFP'98), pages 13–26. ACM Press. URL `http://citeseer.ist.psu.edu/visser98building.html`.

[204] Daniel J. Weitzner, Jim Hendler, Tim Berners-Lee, and Dan Connolly. 2005. Creating a Policy-Aware Web: Dicretionary, Rule-based Access for the World Wide Web. In: E. Ferrari and B. Thuraisingham (editors), Web and Information Security. IOS Press.

[205] E. B. White. 1952. Charlotte's Web. HarperCollins.

[206] William A. Woods. 1975. What's in a Link: Foundations of Semantic Networks. In: D.G.Bobrow and A.M.Collins (editors), Representation and Understanding: Studies in Cognitive Science, pages 35–82. Academic Press, New York.

[207] Guizhen Yang and Michael Kifer. 2000. FLORA: Implementing an Efficient DOOD System Using a Tabling Logic Engine. In: CL '00: Proceedings of the First International Conference on Computational Logic, number 1861 in Lecture Notes in Computer Science, pages 1078–1093. Springer-Verlag, London, UK. ISBN 3-540-67797-6.

[208] Guizhen Yang and Michael Kifer. 2002. On the Semantics of Anonymous Identity and Reification. In: On the Move to Meaningful Internet Systems, 2002 - DOA/CoopIS/ODBASE 2002 Confederated International Conferences DOA, CoopIS and ODBASE 2002, pages 1047–1066. Springer-Verlag, London, UK. ISBN 3-540-00106-9.

[209] Guizhen Yang and Michael Kifer. 2003. Reasoning about Anonymous Resources and Meta Statements on the Semantic Web. In: Stefano Spaccapietra, Sal March, and Karl Aberer (editors), Journal of Data Semantics I, volume 2800 of *Lecture Notes in Computer Science*, pages 69–97. Springer-Verlag.

[210] Mihalis Yannakakis. 1990. Graph-theoretic methods in database theory. In: Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pages 230–242.

[211] Ka-Ping Yee, Kirsten Swearingen, Kevin Li, and Marti Hearst. 2003. Faceted Metadata for Image Search and Browsing. In: CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems, pages 401–408. ACM Press, New York, NY, USA. ISBN 1-58113-630-7.