Semantic Web

Chapter · January 2011

DOI: 10.1007/978-3-642-19193-0_6

Federico MIchele Facca
Martel Innovate
74 PUBLICATIONS **1,222** CITATIONS
SEE PROFILE

Elena Simperl
University of Southampton
321 PUBLICATIONS **3,899** CITATIONS
SEE PROFILE

Ioan Toma
University of Innsbruck
135 PUBLICATIONS **1,431** CITATIONS
SEE PROFILE

Semantic Web: Why, What, and How?

Dieter Fensel

Jim Hendler

Henry Lieberman

Wolfgang Wahlster

Brief Table of Contents

Table of Contents

Preface

Tim Berners-Lee,Director of the World-Wide Web Consortium

Based on a talk presented at the W3C meeting, London, 1997/12/3

1. The original dream

The Web was designed to be a universal space of information, so when you make a bookmark or a hypertext link, you should be able to make that link to absolutely any piece of information that can be accessed using networks. The universality is essential to the Web: it looses its power if there are certain types of things to which you can’t link.

There are a lot of sides to that universality. You should be able to make links to a hastily jotted crazy idea and to link to a beautifully produced work of art. You should be able to link to a very personal page and to something available to the whole planet. There will be information on the Web which has a clearly defined meaning and can be analysed and traced by computer programs; there will be information, such as poetry and art, which requires the full human intellect for an understanding which will always be subjective.

And what was the purpose of all this? The first goal was to work together better. While the use of the Web across all scales is essential to the concept, the original driving force was collaboration at home and at work. The idea was, that by building together a hypertext Web, a group of whatever size would force itself to use a common vocabulary, to overcome its misunderstandings, and at any time to have a running model - in the Web - of their plans and reasons.

For me, the forerunner to the Web was a program called ‘Enquire’, which I made for my own purposes. I wrote it in 1980, when I was working at the European Particle Physics Lab (CERN), to keep track of the complex web of relationships between people, programs, machines and ideas. In 1989, when I proposed the Web, it was as an extension of that personal tool to a common information space.

When we make decisions in meetings, how often are the reasons for those decisions (which we so carefully elaborated in the meeting) then just typed up, filed as minutes and essentially lost? How often do we pay for this, in time spent passing on half-understandings verbally, duplicating effort through ignorance and reversing good decisions from misunderstanding? How much lack of co-operation can be traced to an inability to understand where another party is ‘coming from’? The Web was designed as an instrument to prevent misunderstandings.

For this to work, it had to be not only easy to ‘browse’, but also easy to express oneself. In a world of people and information, the people and information should be in some kind of equilibrium. Anything in the Web can be quickly learned by a person and any knowledge you see as being missing from the

Web can be quickly added. The Web should be a medium for the communication between people: communication through shared knowledge. For this to work, the computers, networks, operating systems and commands have to become invisible, and leave us with an intuitive interface as directly as possible to the information.

2. Re-enter machines

There was a second goal for the Web, which is dependent on the first. The second part of the dream was that, if you can imagine a project (company, whatever) which uses the Web in its work, then there will be an map, in cyberspace, of all the dependencies and relationships which define how the project is going. This raises the exciting possibility of letting programs run over this material, and help us analyze and manage what we are doing. The computer renters the scene visibly as a software agent, doing anything it can to help us deal with the bulk of data, to take over the tedium of anything that can be reduced to a rational process, and to manage the scale of our human systems.

3. Where are we now?

The Web you see as a glorified television channel today is just one part of the plan. Although the Web was driven initially by the group work need, it is not surprising that the most rapid growth was in public information. Web publishing, when a few write and many read, profited most from the snowball effect of exponentially rising numbers of readers and writers. Now, with the invention of the term ‘intranet’, Web use is coming back into organisations. (In fact, it never left. There have always been since 1991, many internal servers, but as they were generally invisible from outside the companies’ firewalls they didn't get much press!). However, the intuitive editing interfaces which make authoring a natural part of daily life are still maturing. I thought that in 12 months we would have generally available intuitive hypertext editors. (*I have stuck to that and am still saying the same thing today!)*

It is not just the lack of simple editors that has prevented use of the Web as a collaborative medium. For a group of people to use the Web in practice, they need reliable access control, so that they know their ideas will only be seen by those they trust. They also need access control and archival tools that, like browsing, don't require one to get into the details of computer operating systems.

There is also a limit to what we can do by ourselves with information, without the help of machines. A familiar complaint of the newcomer to the Web, who has not learned to follow links only from reliable sources, is the about the mass of junk out there. Search engines flounder in the mass of undifferentiated documents that range vastly in terms of quality, timeliness and relevance. We need information about information, ‘metadata’, to help us organise it.

As it turns out, many of these long-term needs will hopefully be met by technology, which for one reason or another is being developed by the technical community, and agreed upon by groups such as the World-Wide Web Consortium (W3C), in response to various medium-term demands.

4. The World Wide Web Consortium - W3C

The Consortium exists as a place for those companies for whom the Web is essential to meet and agree on the common underpinnings that will allow everyone to go forward. (There are currently over 230 member organisations.)

Whether developing software, hardware, networks, information for sale, or using the Web as a crucial part of their business life, these companies are driven by current emerging areas such as Web publishing, intranet use, electronic commerce, and Web-based education and training. From these fields medium-term needs arise and, where appropriate, the Consortium starts an Activity to help reach a consensus on computer protocols for that area. Protocols are the rules that allow computers to talk together about a given topic. When the industry agrees on protocols, then a new application can spread across the world, and new programs can all work together as they all speak the same language. This is key to the development of the Web.

5. Where is the Web Going Next?

5.1. Avoiding the World Wide Wait

You've heard about it, you may have experienced it, but can anything be done about it?

One reason for the slow response you may get from a dial-up Internet account simply follows from the ‘all you can eat’ pricing policy. The only thing which keeps the number of Internet users down is unacceptable response, so if we were to suddenly make it faster, there would almost immediately be more users until it was slow again. I've seen it: when we speeded up an overloaded server by a factor of five, it once again rose to 100% utilisation as the number of users increased by a factor of five.

Eventually, there will be different ways of paying for different levels of quality. But today there some things we can do to make better use of the bandwidth we have, such as using compression and enabling many overlapping asynchronous requests. There is also the ability to guess ahead and push out what a user may want next, so that the user does not have to request and then wait. Taken to one extreme, this becomes subscription-based distribution, which works more like email or newsgroups.

One crazy thing is that the user has to decide whether to use mailing lists, newsgroups, or the Web to publish something. The best choice depends on the demand and the readership pattern. A mistake can be costly. Today, it is not always easy for a person to anticipate the demand for a page. For example, the pictures of the Schoemaker-Levy comet hitting Jupiter taken on a mountain top and just put on the nearest Mac server or the decision Judge Zobel put onto the Web - both these generated so much demand that their servers were swamped, and in fact, these items would have been better delivered as messages via newsgroups. It would be better if the ‘system’, the collaborating servers and clients together, could adapt to differing demands, and use pre-emptive or reactive retrieval as necessary.

5.2. Data about Data - Metadata

It is clear that there should be a common format for expressing information about information (called metadata), for a dozen or so fields that needed it, including privacy information, endorsement labels, library catalogues, tools for structuring and organising Web data, distribution terms and annotation. The Consortium's Resource Description Framework (RDF) is designed to allow data from all these fields to be written in the same form, and therefore carried together and mixed.

That by itself will be quite exciting. Proxy caches, which make the Web more efficient, will be able to check that they are really acting in accordance with the publisher's wishes when it comes to redistributing material. A browser will be able to get an assurance, before imparting personal information in a Web form, on how that information will be used. People will be able, if the technology is matched by suitable tools, to endorse Web pages that they perceive to be of value. Search engines will be able to take such endorsements into account and give results that are perceived to be of much higher quality. So a common format for information about information will make the Web a whole lot better.

5.3. The Web of trust

In cases in which a high level of trust is needed for metadata, digitally signed metadata will allow the Web to include a ‘Web of trust’. The Web of trust will be a set of documents on the Web that are digitally signed with certain keys, and contain statements about those keys and about other documents. Like the Web itself, the Web of trust does not need to have a specific structure like a tree or a matrix. Statements of trust can be added exactly so as to reflect actual trust. People learn to trust through experience and though recommendation. We change our minds about who we trust for different purposes. The Web of trust must allow us to express this.

Hypertext was suitable for a global information system because it has this same flexibility: the power to represent any structure of the real world or a created imagined one. Systems that force you to express information in trees or matrices are fine so long as they are used for describing trees or matrices. The moment you try to use one to hold information that does not fit the mold, you end up twisting the information to fit, and so misrepresenting the situation. Similarly, the W3C's role in creating the Web of trust will be to help the community have common language for expressing trust. *The Consortium will not seek a central or controlling role in the content of the Web.*

5.4. ‘Oh, yeah?’

So, signed metadata is the next step. When we have this, we will be able to ask the computer not just for information, but why we should believe it. Imagine an ‘Oh, yeah?’ button on your browser. There you are looking at a fantastic deal that can be yours just for the entry of a credit card number and the click of a button. "Oh, yeah?", you think. You press the ‘Oh, yeah?’ button. You are asking your browser why you should believe it. It, in turn, can challenge the server to provide some credentials: perhaps, a signature for the document or a list of documents that expresses what that key is good for. Those documents will be signed. Your browser rummages through with the server, looking for a way to convince you that the page is trustworthy for a purchase. Maybe it will come up with an endorsement from a magazine, which in turn has been endorsed by a friend. Maybe it will come up with an endorsement by the seller's bank, which has in turn an endorsement from your bank. Maybe it won't find any reason for you to actually believe what you are reading at all.

5.5. Data about things

All the information mentioned above is information about information. Perhaps the most important aspect of it is that it is machine-understandable data, and it may introduce a new phase of the Web in which much more data in general can be handled by computer programs in a meaningful way. All these ideas are just as relevant to information about the real world: about cars and people and stocks and shares and flights and food and rivers.

The Enquire program assumed that every page was about something. When you created a new page it made you say what sort of thing it was: a person, a piece of machinery, a group, a program, a concept, etc. Not only that, when you created a link between two nodes, it would prompt you to fill in the relationship between the two things or people. For example, the relationships were defined as ‘A is part of B’ or ‘A made B’. The idea was that if Enquire were to be used heavily, it could then automatically trace the dependencies within an organisation.

Unfortunately this was lost as the Web grew. Although it had relationship types in the original specifications, this has not generally become a Web of assertions about things or people. Can we still build a Web of well-defined information?

My initial attempts to suggest this fell on stony ground, and not surprisingly. HTML is a language for communicating a document for human consumption. SGML (and now XML) gives structure, but not semantics. Neither the application, nor the language, called for it.

With metadata we have a need for a machine-understandable language that has all the qualities we need. Technically, the same apparatus we are constructing in the Resource Description Framework for describing the properties of documents can be used equally well for describing anything else.

5.6. A crying need for RDF

Is there a real need for this metadata and is there a market in the medium term that will lead companies to develop in this direction? Well, in the medium term, we see the drivers already - web publishing, education and training, electronic commerce and intranets.

I have mentioned the vicious circle that caused the Web to take off initially. The increasing amount of information on the Web was an incentive for people to get browsers, and the increasing number of browsers created more incentive for people to put up more Web sites. It had to start somewhere and it was bootstrapped by making ‘virtual hypertext’ servers. These servers typically had access to large databases - such as phone books, library catalogues and existing documentation management systems. They had simple programs which would generate Web pages ‘on the fly’ corresponding to various views and queries on the database. This has been a very powerful ‘bootstrap’ as there is now a healthy market for tools to allow one to map one's data from its existing database form on to the Web.

Now here is the curious thing. There is so much data available on Web pages, that there is a market for tools that ‘reverse engineer’ that process. These are tools that read pages, and with a bit of human advice, recreate the database object. Even though it takes human effort to analyse the way different Web sites are offering their data, it is worth it. It is so powerful to have a common, well defined interface to all the data so that you can program on top of it. So the need for well defined interface to Web data in the short term is undeniable.

What we propose is that, when a program goes out to a server looking for data, say a database record, that the same data should be available in RDF, in such a way that the rows and columns are all labelled in a well-defined way. That it may be possible to look up the equivalence between field names at one Web site and at another, and so merge information intelligently from many sources. This is a clear need for metadata, just from looking at the trouble libraries have had with the numbers of very similar, but slightly different ways of making up a catalogue card for a book.

5.7. Interactive Creativity

I want the Web to be much more creative than it is at the moment. I have even had to coin a new word - Intercreativity - which means building things together on the Web. I found that people thought that the Web already was ‘interactive’, because you get to click with a mouse and fill in forms! I have mentioned that better intuitive interfaces will be needed, but I don’t think they will be sufficient without better security.

It would be wrong to assume that digital signature will be mainly important for electronic commerce, as if security were only important where money is concerned. One of my key themes is the importance of the Web being used on all levels from the personal, through groups of all sizes, to the global population.

When you are working in a group, you do things you would not do outside the group, You share half-baked ideas, reveal sensitive information. You use a vernacular that will be understood; you can cut corners in language and formality. You do these things because you trust the people in the group, and that others won't suddenly have access to it. To date, on the Web, it has been difficult to manage such groups or to allow one to control access to information in an intuitive way.

5.8. Letting Go

So, where will this get us? The Web fills with documents, each of which has pointers to help a computer understand it and relate it to terms it knows. Software agents acting on our behalf can reason about this data. They can ask for and validate proofs of the credibility of the data. They can negotiate as to who will have what access to what and ensure that our personal wishes for privacy level be met.

The world is a world of human beings, as it was before, but the power of our actions is again increased. The Web already increases the power of our writings, making them accessible to huge numbers of people and allowing us to draw on any part of the global information base by a simple hypertext link. Now we image the world of people with active machines forming part of the infrastructure. We only have to express a request for bids, or make a bid, and machines will turn a small profit matching the two. Search engines, from looking for pages containing interesting words, will start indexes of assertions that might be useful for answering questions or finding justifications.

I think this will take a long time. I say this deliberately, because in the past I have underestimated how long something will take to become available (e.g. good editors in 12 months).

Now we will have to find how best to integrate our warm fuzzy right-brain selves into this clearly defined left-brain world. It is easy to know who we trust, but it might be difficult to explain that to a computer. After seeding the semantic Web with specific applications, we must be sure to generalise it cleanly, leaving it clean and simple so that the next generation can learn its logical concepts along with the alphabet.

If we can make something decentralised, out of control, and of great simplicity, we must be prepared to be astonished at whatever might grow out of that new medium.

5.9. It’s up to us

One thing is certain. The Web will have a profound effect on the markets and the cultures around the world: intelligent agents will either stabilise or destabilise markets; the demise of distance will either homogenise or polarise cultures; the ability to access the Web will be either a great divider or a great equaliser; the path will either lead to jealousy and hatred or peace and understanding.

The technology we are creating may influence some of these choices, but mostly it will leave them to us. It may expose the questions in a starker form than before and force us to state clearly where we stand.

We are forming cells within a global brain and we are excited that we might start to think collectively. What becomes of us still hangs crucially on how we think individually.

1. Introduction

Dieter Fensel[1], Jim Hendler[2], Henry Lieberman[3], and Wolfgang Wahlster[4]

The World-wide Web (WWW) has drastically changed the availability of electronically available information. Currently there are around one billion static documents in the WWW which are used by more than 200 million users internationally. In addition, this number is growing astronomically. In 1990, the WWW began with a small number of documents as an in-house solution for around thousand users at CERN. By 2002, the standardization committee for the WWW (called W3C[5]) expects around a billion web users and a even higher number of available documents. However, this success and exponential grow makes it increasingly difficult to find, to access, to present, and to maintain the information of use to a wide variety of users. Currently, pages on web must use representation means rooted in format languages such as HTML or SGML and make use of protocols that allow browser to present information to human readers. The information content, however, is mainly presented by natural language. Thus, there is a wide gap between the information available for tools and the information kept in human readable form. This causes serious problems in accessing and processing the available information.

- *Searching for information*: Already, finding the right piece of information is often a nightmare. One gets lost in huge amounts of irrelevant information and may often miss the relevant ones. Searches are imprecise, often returning pointers to many thousands of pages (and this situation worsens as the web grows). In addition, a user must read retrieved documents in order to extract the desired information—so even once the page is found, the search may be difficult or the information obscured. Thus, the same piece of knowledge must often be presented in different contexts and adapted to different users' needs and queries. However, the web lacks automated translation tools to allow this information to automatically be transformed between different representation formats and contexts.
- *Presenting information:* A related problem is that the maintenance of web sources has become very difficult. Keeping redundant information consistent and keeping information correct is hardly supported by current web tools, and thus the burden on a user to maintain the consistency is often overwhelming. This leads to a plethora of sites with inconsistent and/or contradictory information.
- *Electronic commerce:* Automatization of electronic commerce is seriously hampered by the way information is currently presented. Shopping agents use wrappers and heuristics to extract product informations from weakly structured textual

information. However, development and maintenance costs are high and provided services are limited. B2B market places offer new possibilities for electronic commerce, however, are hampered by the large and increasing mapping costs required to integrate heterogeneous product descriptions.

There is an emerging awareness that providing solutions to these problems requires that there be a machine understandable semantics for some or all of the information presented in the WWW. Achieving such a *semantic web* [Berners-Lee, 1999] requires:

- Developing languages for expressing machine understandable meta information for documents and developing terminologies (i.e., name spaces or ontologies) using these languages and making them available on the web.
- Developing tools and new architectures that use such languages and terminologies to provide support in finding, accessing, presenting and maintaining information sources.

Finally, realizing applications that provide a new level of service to the human users of the semantic web.

Developing such languages, ontologies, and tools is a wide ranging problem that touches on the research areas of a broad variety of research communities. Therefore this book brought together colleagues from these different research communities. These include researchers in the areas of databases, intelligent information integration, knowledge representation, knowledge engineering, information agents, knowledge management, information retrieval, natural language processing, meta data, web standards, and others. The book is based on a seminar we had in Dagstuhl, Germany, in March 2000. The contents of the book is organized as follows. First, a number of arising new web standards are discussed that should improve the representation of machine processable semantics of information. Second, ontologies are introduced for representation of semantics (in the sense of formal and real-world semantics) in these formalisms. Third, these semantic annotations allow automatization in information access and task achievement. Therefore, we discussed intelligent information access based on these semantic annotations. Forth, a number of applications of these new techniques are presented.

The purpose of this book is to provide an overall motivation of the subject. We first discuss further the need for a semantic web in Section 1. Mainly its need is motivated by shortcomings of the current state of the WWW. We will show which kind of new services the semantic web will enable, and in Section 2 we will explain how this can be achieved.

1. Why Is There a Need for the Semantic Web and What It Will Provide

The web has brought exiting new possibilities for information access and electronic commerce. It was its simplicity that enabled its quick take up and exponential growth. However, meanwhile this simplicity also seriously hampers its further growth. We will discuss these bottlenecks for knowledge management and electronic commerce (cf. [Fensel, 2001] for more details).

Knowledge Management is concerned with acquiring, maintaining, and accessing knowledge of an organization. It aims to exploit an organizations intellectual assets for greater productivity, new value, and increased competitiveness. Due to globalization and the impact of the Internet, many organizations are increasingly geographically dispersed and organized around virtual teams. With the large number of on-line documents, several document management systems entered the market. However these systems have severe weaknesses:

- Searching information: Existing keyword-based search retrieves irrelevant information which uses a certain word in a different context, or it may miss information where different words about the desired content are used.

- Extracting information: Human browsing and reading is currently required to extract relevant information from information sources, as automatic agents lack all common sense knowledge required to extract such information from textual representations, and they fail to integrate information spread over different sources.

- Maintaining weakly structured text sources is a difficult and time- consuming activity when such sources become large. Keeping such collections consistent, correct, and up-to-date requires a mechanized representation of semantics and constraints that help to detect anomalies.
- Automatic document generation: Adaptive web sites which enable a dynamic reconfiguration according to user profiles or other relevant aspects would be very useful. The generation of semi-structured information presentations from semi-structured data requires a machine-accessible representation of the semantics of these information sources.

Semantic web technology will enable structural and semantic definitions of documents providing completely new possibilities: Intelligent search instead of keyword matching, query answering instead of information retrieval, document exchange between departments via ontology mappings, and definition of views on documents.

Web Commerce (B2C): Electronic Commerce is becoming an important and growing business area. This is happening for two reasons. First, electronic commerce is extending existing business models. It reduces costs and extends existing distribution channels and may even introduce new distribution possibilities. Second, it enables completely new business models or gives them a much greater importance than they had before. What has up to now been a peripheral aspect of a business field may suddenly receive its own important revenue flow. Examples of business field extensions are on-line stores, examples of new business fields are shopping agents, on-line marketplaces and auction houses that make comparison shopping or meditation of shopping processes into a business with its own significant revenue flow. The advantages of on-line stores and the success story of many of them has led to a large number of such shopping pages. The new task for a customer is now to find a shop that sells the product he is looking for, getting it in the desired quality, quantity, and time, and paying as little as possible for it. Achieving these goals via browsing requires significant time and will only cover a small share of the actual offers. Very early, shopbots were developed that visit several stores, extract product information and present to the customer a instant market overview. Their functionality is provided via wrappers that need to be written for each on-line store. Such a wrapper uses a keyword search for finding the product information together with assumptions on regularities in the presentation format of stores and text extraction heuristics. However, this technology has two severe limitations:

- Effort: Writing a wrapper for each on-line store is a time-consuming activity and changes in the outfit of stores cause high maintenance efforts.

- Quality: The extracted product information is limited (mostly price information), error prone and incomplete. For example, a wrapper may extract the direct product price but misses indirect costs such as shipping costs etc.

These problems are caused by the fact that most product information is provided in natural language, and automatic text recognition is still a research area with significant unsolved problems. What is required is a machine-processable semantics of the provided information. The situation will drastically change when standard representation formalisms for the structure and semantics of data are available. Software agents can be build that can "understand" the product information. Meta- on-line stores can be built with little effort and this technique will also enable complete market transparency in the various dimensions of the diverse product properties. The low-level programming of wrappers based on text extraction and format heuristics will be replaced by semantic mappings, which translate different product descriptions into each other and which can be used to navigate and search automatically for the required information.

Electronic Business (B2B): Electronic Commerce in the business to business field (B2B) is not a new phenomena. Initiatives to support electronic data exchange in business processes between different companies existed already in the sixties. In order to exchange business transactions sender and receiver have to agree on a common standard (a protocol for transmitting the content and a language for describing the content) A number of standards arose for this purpose. One of them is the UN initiative Electronic Data Interchange for Administration, Commerce, and Transport (EDIFACT). In general, the automatization of business transactions has not lived up to the expectations of its propagandists. This can be explained by some serious shortcomings of existing approach like EDIFACT: It is a rather procedural and cumbersome standard, making the programming of business transactions expensive, error prone and hard to maintain. Finally, the exchange of business data via extranets is not integrated with other document exchange processes, i.e., EDIFACT is an isolated standard. Using the infrastructure of the Internet for business exchange significantly improve this situation. Standard browsers can be used to render business transactions and these transactions are transparently integrated into other document exchange processes in intranet and Internet environments. However, this is currently hampered by the fact that HTML do not provide a means for presenting rich syntax and semantics of data. XML, which is designed to close this gap in current Internet technology, will already change the situation. B2B communication and data exchange can then be modeled with the same means that are available for the other data exchange processes, transaction specifications can easily be rendered by standard browsers, maintenance will be cheap. XML provides a standard serialized syntax for defining the structure and semantics of data. Therefore, it provides means to represent the semantics of the information as part of defining their structure. However, XML does *not* provide standard data structures and terminologies to describe business processes and exchanged products. Therefore, new semantic web technology will have to play important roles in XML-enabled electronic commerce:

- First, languages with a defined data model and rich modeling primitives have to be defined that provide support in defining, mapping, and exchanging product data.
- Second, standard ontologies have to be developed covering the various business areas. Examples in this area are Common Business Library (CBL), Commerce XML (cXML), ecl@ss, Open Applications Group Integration Specification (OAGIS), RosettaNet, and UN/SPSC.
- Third, efficient translation services is required in areas where standard ontologies do not exist[6] or where a particular client wants to use his own terminology and needs translation service from his terminology into the standard. This translation service must cover structural and semantical as well as language differences.

These support will significantly extend the degree to which data exchange is automated and will create complete new business models in the participating market segments.

The semantic web deals with important application areas such as knowledge management and electronic commerce (B2C and B2B). It may help to overcome many of the current bottlenecks in these areas. The next section will explain, how this can be achieved.

2. How the Semantic Web Will Be Possible

Above we described new services provided by the semantic web. Here we will discuss how such a new level of service can be achieved. First, we describe new languages that allow to add semantics to the web. Second, we describe important tools and finally we illustrate by some applications the potential utility of the semantic web.

2.1. Languages

Languages for the semantic web include two aspects. First, they need to provide some formal syntax and formal semantics to enable automated processing of their content. Second, they need to provide some standardized vocabulary referring to some real-world semantics enabling automatic and human agents to share information and knowledge. The latter is provided by ontologies.

2.1.1. Formal Languages

Originally, the web grew mainly around the language HTML, that provide a standard for structuring documents that was translated by browsers in a canonical way to render documents. On the one hand, it was the simplicity of HTML that enabled the fast grow of the WWW. On the other hand, its simplicity seriously hampered more advanced web application in many domains and for many tasks. This was the reason to define XML (see Figure 1.1) which allows to define arbitrary domain and task specific extensions (even HTML got redefined as an XML application, see XHTML). Therefore, it is just consequence to define the semantic web as an XML application. The first step in this direction is taken by RDF which define a syntactical convention and a simple data model for representing machine-processable semantics of data. A second step is taken by RDFS that define basic ontological modeling primitives on top of RDF. A full-blown ontology modeling language as extension of RDFS are defined by OIL and DAML-O which conclude our discussion on semantic web languages.

The *Resource Description Framework (RDF)* is a standard for Web meta data developed by the World Wide Web Consortium (W3C) [Lassila 1998]. Expanding from the traditional notion of document meta data (such as something like library catalog information), RDF is suitable for describing any Web resources, and as such provides interoperability between applications that exchange machine-understandable information on the Web. RDF is an XML application and adds a simple data model on top of XML. This data model provides three elements:

Objects, properties, and values of properties applied to a certain object.

The *RDF Schema (RDFS)* candidate recommendation (cf. [Brickley & Guha, 2000]) defines additional modeling primitives on top of RDF. It allows to define classes (i.e., concepts), inheritance hierarchy for classes and properties, and domain and range restrictions for properties. OIL[7] (cf. [Fensel et al., 2001]) takes RDF schema as a starting point and extends it to a full-fledged ontology modeling language. Such an ontology languages must fulfill three important requirements:

It must be highly intuitive to the human user. Given the current success of the frame-based and object-oriented modeling paradigm they should have a frame-like look and feel.

- It must have a well-defined formal semantics with established reasoning properties in terms of completeness, correctness, and efficiency.[8]
- It must have a proper link with existing web languages like XML and RDF ensuring interoperability.

In this respect, many of the existing languages like CycL [Lenat & Guhy, 1990], KIF [Genesereth, 1991], Ontolingua [Farquhar et al., 1997], and SHOE [Luke et al., 1996] fail. However, the Ontology Inference Layer OIL matches the criterion mentioned above. OIL unifies three important aspects provided by different communities: Epistemologically rich modeling primitives as provided by the Frame community, formal semantics and efficient reasoning support as provided by Description Logics, and a standard proposal for syntactical exchange notations as provided by the Web community.

Another candidate for such a web-based ontology modeling language is *DAML-O*[9] funded by DARPA. However, this language is still in its early stage and lack a formal definition of its semantics.]

2.1.2. Ontologies

Ontologies were developed in Artificial Intelligence to facilitate knowledge sharing and reuse. Since the beginning of the nineties ontologies have become a popular research topic investigated by several Artificial Intelligence research communities, including Knowledge Engineering, natural-language processing and knowledge representation. More recently, the notion of ontology is also becoming widespread in fields such as intelligent information integration, cooperative information systems, information retrieval, electronic commerce, and knowledge management. The reason ontologies are becoming so popular is in large part due to what they promise: a shared and common understanding of some domain that can be communicated between people and application systems. Because ontologies aim at consensual domain knowledge their development is often a cooperative process involving different people, possibly at different locations. People who agree to accept an ontology are said to commit themselves to that ontology.

Many definitions of ontologies have been given in the last decade, but one that, in our opinion, best characterizes the essence of an ontology is based on the related definitions by [Gruber, 1993]: An ontology is a formal, explicit specification of a shared conceptualisation. A ‘conceptualisation’ refers to an abstract model of some phenomenon in the world which identifies the relevant concepts of that phenomenon. ‘Explicit’ means that the type of concepts used and the constraints on their use are explicitly defined. ‘Formal’ refers to the fact that the ontology should be machine understandable. Hereby different degrees of formality are possible. Large ontologies like WordNet[10] provide a thesaurus for over 100,000 terms explained in natural language. On the other end of the spectrum is CYC[11], that provides formal axiomating theories for many aspect of common sense knowledge. ‘Shared’ reflects the notion that an ontology captures consensual knowledge, that is, it is not restricted to some individual, but accepted by a group.

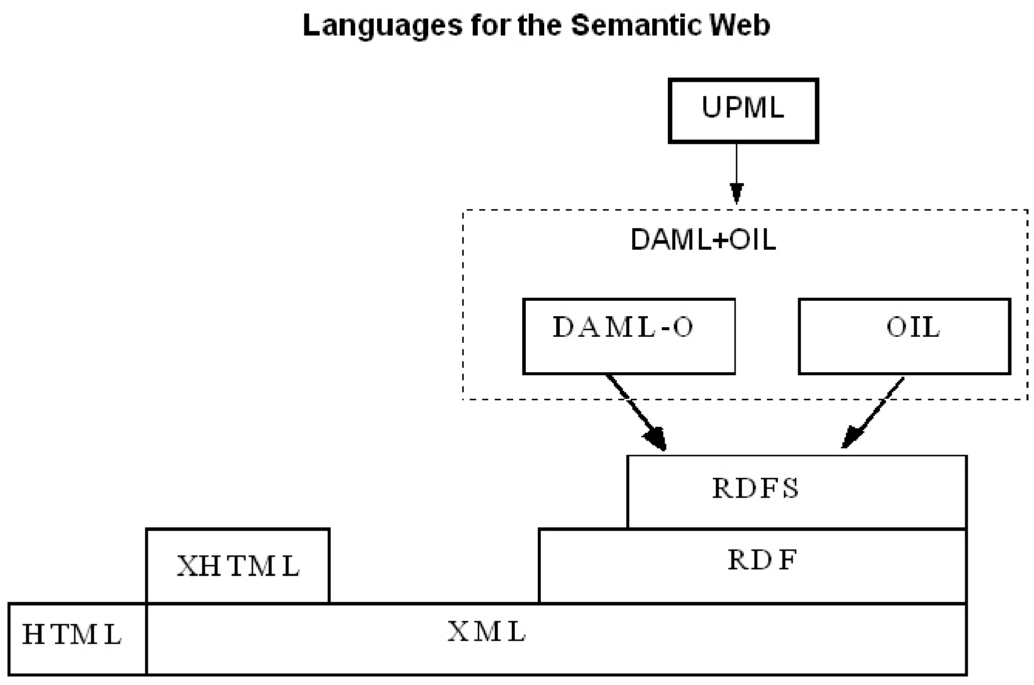


Figure 1.1 The layer language model for the WWW.

2.2. Tools

Effective and efficient work with the semantic web must be supported by advanced tools enabling the full power of this technology. In particular, we need the following elements:

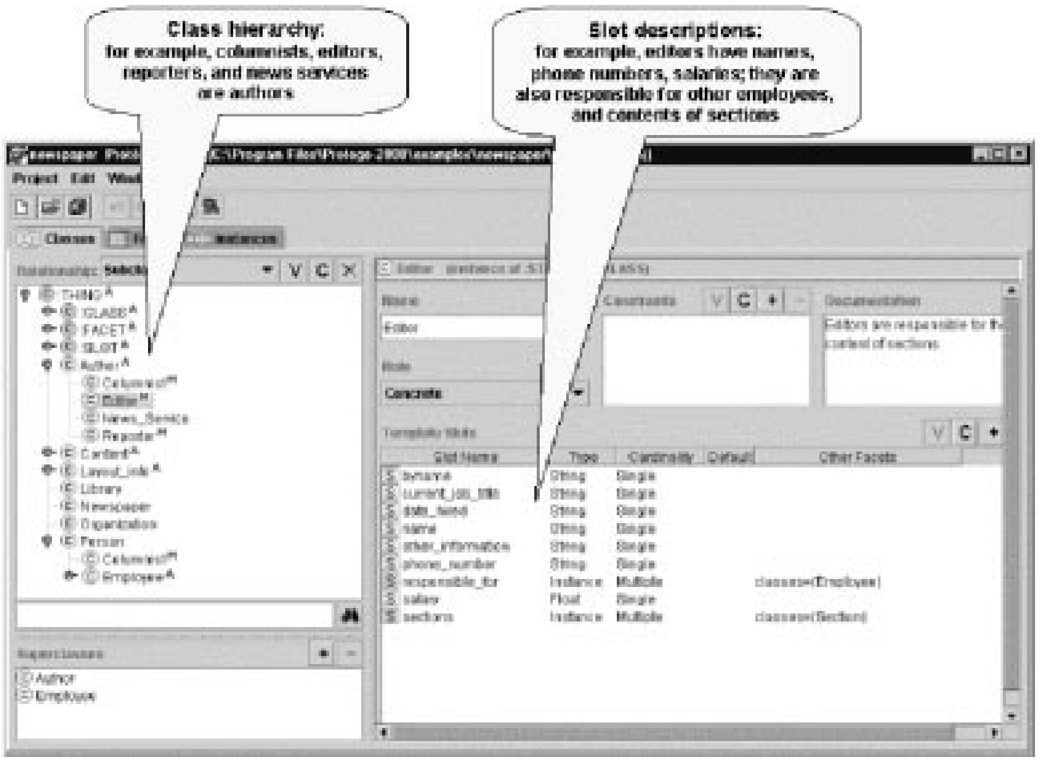
- Formal languages to express and represent ontologies. We already discussed some of them during the last section.
- Editors and semi-automatic construction to build new ontologies.
- Reusing and Merging Ontologies: Ontology Environments that help to create new ontologies by reusing existing ones.
- Reasoning Service: Instance and Schema Inferences enable advanced query answering service, support ontology creation and help to map between different terminologies.

- Annotation tools to link unstructured and semi-structured information sources with meta data.
- Tools for information access and navigation that enable intelligent information access for human users.
- Translation and integration Service between different ontologies that enable multi-standard data interchange and multiple view definitions (especially for B2B electronic commerce).

In the following, we will briefly describe examples for these technologies.

2.2.1. Editors and semi-automatic construction

Ontology editors help human knowledge engineers to build ontologies. Ontology editors support the definition of concept hierarchies, the definition attributes for concepts, and the definition of axioms and constraints. They must provide graphical interfaces and must confirm to existing standards in web-based software development. They enable inspecting, browsing, codifying and modifying ontologies and supports in this way the ontology development and maintenance task. An example system is Protégé [Grosso et al., 1999]. Protégé allows domain experts to build knowledge-based systems by creating and modifying reusable ontologies and problem-solving methods. Protégé generates domain- specific knowledge-acquisition tools and applications from ontologies. Protégé has been used in more than 30 countries. It is an ontology editor which you can use to define classes and class hierarchy, slots and slot- value restrictions, relationships between classes and properties of these relationships (see Figure 1.2). The instances tab is a knowledge-acquisition tool which you can use to acquire instances of the classes defined in the ontology. Protégé is built at the University of Stanford.


Class hierarchy: for example, columnists, editors, reporters, and news services are authors
Slot descriptions: for example, editors have names, phone numbers, salaries; they are also responsible for other employees, and contents of sections
Classes
Instances
Name
Editor
Role
Concrete
Constraints
Documentation
Template Slots
Superclasses
Author
Employee

Editors are responsible for the content of sections.

Slot Name	Type	Cardinality	Default	Other Facets
byname	String	Single		
current_job_title	String	Single		
date_hired	String	Single		
name	String	Single		
other_information	String	Single		
phone_number	String	Single		
responsible_for	Instance	Multiple		classes={Employee}
salary	Float	Single		
sections	Instance	Multiple		classes={Section}

Figure 1.2 The Protégé editor.

Manually building ontologies is a time-consuming task It is very difficult and cumbersome to manually derive ontologies from data. This appears to be true even regardless of the type of data one might consider. Natural language texts exhibit morphological, syntactic, semantic, pragmatic and conceptual constraints that interact in order to convey a particular meaning to the reader. Thus, the text transports information to the reader and the reader embeds this information into his background knowledge. Through the understanding of the text data is associated with conceptual structures and new conceptual structures are learned from the interacting constraints given through language. Tools that learn ontologies from natural language exploit the interacting constraints on the various language levels (from morphology to pragmatics and background knowledge) in order to discover new concepts and stipulate relationships between concepts. Therefore, in addition to editor support, semi- automated tools in ontology development help to improve the overall productivity. These tools combine machine learning, information extraction and linguistic techniques. Main tasks are: extracting relevant concepts, building is-a hierarchies, and extracting relationships between concepts.

An example system is *Text-To-Onto* (cf. [Mädche & Staab, 2000]). The Text-To-Onto system provides an integrated environment for the task of learning ontologies learning from text. The Text Management module enables selecting a relevant corpus. These domain texts may be both, natural language texts and HTML formatted texts. For a meaningful text analysis textual preprocessing has to be performed. The Text Management module serves as an interface to the Information Extraction Server. If there already exists an domain lexicon the information extraction server performs domain specific parsing. The results of the parsing process are stored in XML or feature-value structures.The Management Module offers all existing learning components to the user. Typically these components are parametrizable. Existing knowledge structures (for example a taxonomy of concepts) are incorporated as background knowledge. The learning component discovers on the base of the domain texts new knowledge structures, which are grabbed in the Ontology Modeling Module to expand the existing ontology. Text-To- Onto is developed at the Knowledge Management Group of University of Karlsruhe, Institute AIFB (see Figure 1.3).

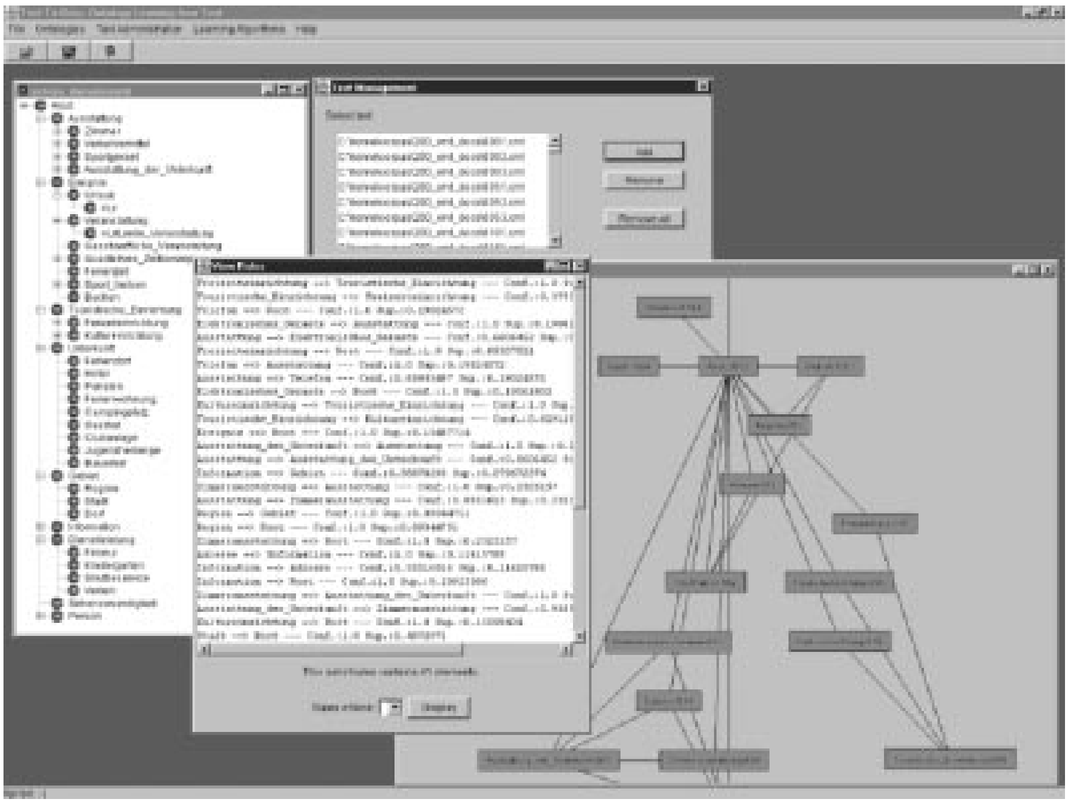


Figure 1.3 Text-To-Onto.

2.2.2. Ontology Environments

Assuming that the world is full of well-designed modular ontologies, constructing a new ontology is a matter of assembling existing ones. Instead of building ontologies from scratch one wants to reuse existing ontologies. Tools that support this approach most provide in adaptation and merging of existing ontologies to make them fitting to new tasks and domains. Operations for combining ontologies are: Ontology inclusion, Ontology restriction, and polymorphic refinement of Ontology. E.g. inclusion of one ontology in another has the effect that the composed ontology consists of the union of the two ontologies (their classes, relations, axioms). The knowledge engineer needs support in merging multiple ontologies together and diagnosing individual or multiple ontologies. He require supports in such tasks as suing ontologies in differing formats, reorganizing taxonomies, resolving name conflicts, browsing ontologies, editing terms, etc. A tool environment in this area is Chimaera which provides support for two important tasks: (1) merging multiple ontologies and (2) diagnosing (and evolving) ontologies [McGuinness et al., 2000]. It has been developed at the University of Stanford. Figure 1.4 illustrates Chimaera.

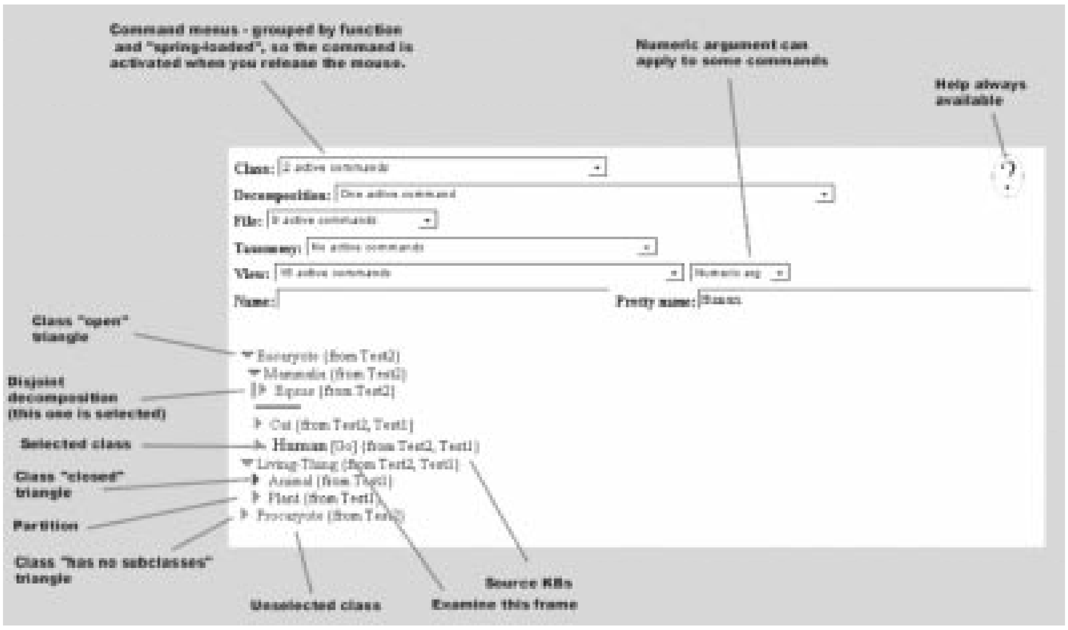


Figure 1.4 Chimaera.

2.2.3. Reasoning Service

Inference engines for ontologies can be used to reason about instances of an ontology or over ontology schemes.

- Reasoning over Instances of an ontology, for example, derive a certain value for an attribute applied to an object. Such inference service can be used to answer queries about explicit and implicit knowledge specified by an ontology. The powerful support in formulating rules, constraints and answering queries over schema information is far beyond existing database technology. These inference services are the equivalent of SQL query engines for databases, however provide stronger support (for example, recursive rules). An example system is Ontobroker ([Fensel et al., 2000b]) which is meanwhile commercialized by the company Ontoprise[12].

- Reasoning over Concepts of an ontology, for example, automatically derive the right position of a new concept in a give concept hierarchy. The FaCT (Fast Classification of Terminologies) [Horrocks & Patel-Schneider, 1999] can be used to automatically derive concept hierarchies. It is a Description Logic (DL) classifier that makes use of the well-defined semantics of OIL. FaCT can be accessed via a Corba interface. It has been developed at the University of Manchester and currently an internet start up may go for implementing a commercial version. FaCT is one of if not THE most efficient reasoner for these kind of tasks.

Both types of reasoners help to build ontologies and to use them for advanced information access and navigation as we will discuss below.

2.2.4. Annotation tools

Ontologies can be used to describe large instance population. Tools help the knowledge engineer to establish such link via:

- Linking an ontology with a database schema or deriving a database schema from an ontology in case of structured data
- Deriving an XML DTD, an XML schema, and an RDF schema from an ontology in case of semi-structured data
- Manually or semi-automatically adding ontological annotation to unstructured data.

More details can be found in [Erdmann & Studer, 2001] and [Klein et al., 2000].

2.2.5. Tools for information access and navigation

Working with the Web is currently done at a very low level: Clicking on links and using key word search for links is the main (if not only) navigation technique. It is like programming with assembler and go-to instructions. The low-level interface may significantly hamper the expected growth of the Web in the future.

- Keyword-based search retrieves irrelevant information that use a certain word in a different meaning or it may miss information where different words are used to describe the desired content. Navigation is only supported by predefined links and does not support clustering and linking of pages based on semantic similarity.
- The query responses require human browsing and reading to extract the relevant information from these information sources. This burdens web users with an additional loss of time and seriously limits information retrieval by automatic agents that miss all common sense knowledge required to extract such information from textual representations
- Keyword-based document retrieval fails to integrate information spread over different sources.
- Finally, each current retrieval service can only retrieve information that is represented by the WWW. No further

inference service is provided for deriving implicit information.

Ontologies help to overcome this bottlenecks in information access. They support information retrieval based on the actual content of a page. They help to navigate the information space based on semantic concepts. They enable advanced query answering and information extraction service, integrating heterogeneous and distributed information sources enriched by inferred background knowledge. This provides two main improvements:

- Semantic information visualization: does not group information on location but group information on contents providing semantic-based navigation support. Example are the hyperbolic browsing interface of Ontoprise (see Figure 1.5) and the page content visualization tool of Aidministrator[13] (see Figure 1.6).
- Direct query answering services based on semi-structured information sources.

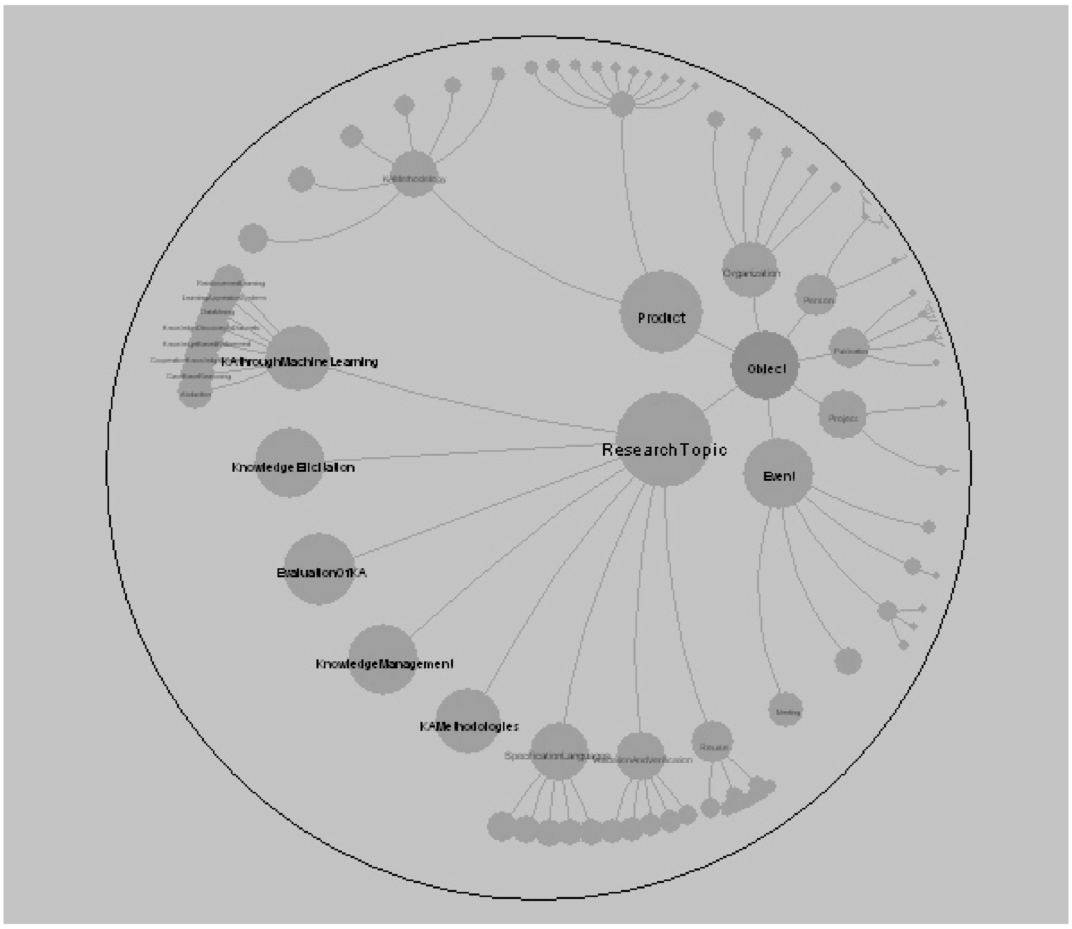


Figure 1.5 A hyperbolic browsing interface.

2.2.6. Translation and Integration Service

Around 80% of the electronic business will be in the B2B area. All experts expect exponential growth of this area. Many studies estimate around 10,000 B2B market places that are set up during the next years. However, there is one serious obstacle: The heterogeneity of product descriptions and the exponential growing effort in mapping these heterogeneous descriptions. Therefore, effective and efficient content management of heterogeneous product catalogues is the critical point for the B2B success. Traditional B2B did not change the business model. It only helped to reduce transactions costs.

It required one mapping from one supplier to one customer or *N* mappings from one supplier to *N* customers. The new business model of B2B market places change the business model bringing electronic commerce to its full economical potential: *individual product search, corporative product search, market transparency, easy access, and negotiation*[14].

An Internet-based marketplace can help significantly to bring both sides together. It provides instant market overview and offers comparison shopping. This marketplace will significantly change the business model of this market segment. Basically, it will replace or at least compete with traditional mediation agents, like wholesale traders. However, the number of required mappings explode. *M* companies exchange business transactions electronically with *N* companies in a fragmented market. In consequence one would need *M*N* mappings. These mappings arise at two levels:

- Different representations of product catalogues must be merged:Different vendors may use different representation of their catalogue data.
    - A product catalogue in EXPRESS with a product catalogue in XML.
    - A product catalogue in XML with DTD1 with a product catalogue in XML with DTD2.
- Different vocabularies used to describe products must be merged. Differences appear in:

the languages that are used to describe the products (English, Spanish, French, German etc.);

- the concepts that are used to define products;
- the attributes that are used to define products;
- the values and value types that are used to define products;
- the overall structure that is used to define products.

We need intermediate architectures that reduces drastically the inherent complexity of the process for each mapping and for reducing the number of mappings. Compared to the prominent need on flexible mappings tools between ontologies, not much actual tools can be reported. A promising approach based on a meta-level architecture is descried in [Bowers & Delcambre, 2000].

2.3. Applications

In the beginning we sketched three application areas for semantic web technologies: knowledge management, B2C web commerce, and B2B electronic business. This section provide some prototypical examples for such applications. It is not meant as a representative survey of the field because this would require much more space and would be a paper (if not a book) by its own.

In On-To-Knowledge[15] [Fensel et al., 2000a], an environment for knowledge management in large intranets and web sites is built. Unstructured and semi-structured data will be automatically annotated, and agent-based user interface techniques and visualization tools help the user to navigate and query the information space. Here, On-To- Knowledge continues a line of research that was set up with SHOE [Luke et al., 1996] and Ontobroker [Fensel et al., 1998]: using ontologies to model and annotate the semantics of information resources in a machine- processable manner. On-To-Knowledge is carrying out three industrial case studies with SwissLife[16], British Telecom[17], and Enersearch[18] to evaluate the tool environment for ontology-based knowledge management. In this context, CognIT[19] extended their information extraction tool Corporum to generate ontologies from semi-structured or unstructured natural language documents. Important concepts and their relationships are extracted from these documents and used to build up initial ontologies. Figure 1.6 shows an automatically generated *semantic* structure maps of the EnerSearch web site using Aidministrator technology[20].

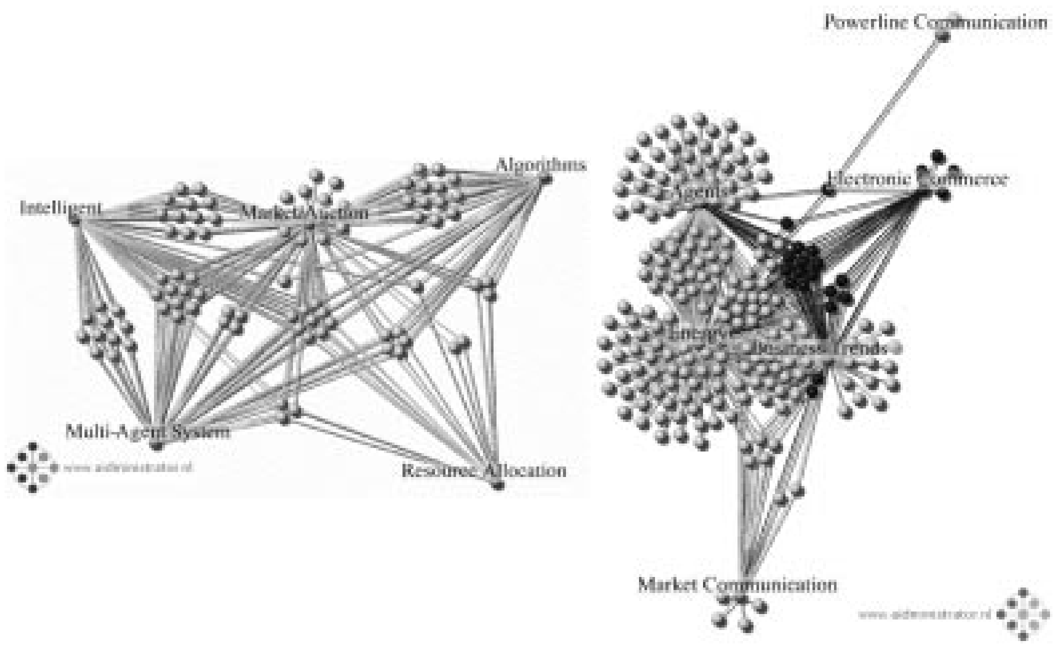


Figure 1.6 Automatically generated semantic structure maps.

Application of semantic web technology in the B2C area is developed by Semantic Edge[21] that develop voice-based and natural language frond end for accessing distributed and heterogeneous product informations. Instead of manually browsing large volumes of product information the human user will be enabled to ask simple questions like: "I am looking for a cheap color printer for my Mac."

Finally, the B2B area may become the most important application area of semantic web technology in terms of the market volume. Companies like VerticalNet[22] that build many vertical market places or ContentEurope[23] that provide content management solutions for B2B electronic commerce all face the same problem: Integrating heterogeneous and distributed product information. Naturally such companies make use of ontology- based integration techniques to reduce their effort in providing integrated solutions for B2B market places.

References

[Bowers & Delcambre, 2000] S. Bowers and L. Delcambre: Representing and Transforming Model- Based Information. In Electronic Proceedings of the ECDL 2000 Workshop on the Semantic Web, 21 September 2000, Lisbon Portugal.

[Berners-Lee, 1999] T. Berners-Lee: Weaving the Web, Orion Business Books, London, 1999.

[Brickley &. Guha, 2000] D. Brickley and R. Guha. Resource Description Framework (RDF) Schema Specification 1.0. Candidate recommendation, World Wide Web Consortium, Mar. 2000. See http://www.w3.org/TR/2000/CR-rdf- schema-20000327.

[Erdmann & Studer, 2001] M. Erdmann and R. Studer: How to Structure and Access XML Documents With Ontologies. Data and Knowledge Engineering 36 (2001), pp. 317-335.

[Farquhar et al., 1997] A. Farquhar, R. Fikes, and J. Rice, The Ontolingua Server: A Tool for Collaborative Ontology Construction, International Journal of Human- Computer Studies, 46:707-728, 1997.

[Fensel, 2001] D. Fensel: Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce, Springer-Verlag, Berlin, 2001.

[Fensel et al., 1998] D. Fensel, S. Decker, M. Erdmann und R. Studer: Ontobroker: The Very High Idea. In Proceedings of the 11th International Flairs Conference (FLAIRS-98), Sanibal Island, Florida, USA, 131-135, Mai 1998.

[Fensel et al., 2001] D. Fensel, I. Horrocks, F. Van Harmelen, D. McGuiness, and P. Patel-Schneider: OIL: Ontology infrastructure to Enable the semantic Web, IEEE Intelligent systems, March/April, 2001.

[Fensel et al., 2000a] D. Fensel, F. van Harmelen, M. Klein, H. Akkermans, J. Broekstra, C. Fluit, J. van der Meer, H.-P. Schnurr, R. Studer, J. Hughes, U. Krohn, J. Davies, R. Engels, B. Bremdal, F. Ygge, U. Reimer, and I. Horrocks: On- To-Knowledge: Ontology-based Tools for Knowledge Management. In Proceedings of the eBusiness and eWork 2000 (EMMSEC 2000) Conference, Madrid, Spain, October 2000.

[Fensel et al., 2000b] D. Fensel, J. Angele, S. Decker, M. Erdmann, H.-P. Schnurr, R. Studer and, A. Witt: Lessons Learned from Applying AI to the Web, Journal of Cooperative Information Systems, 9(4), 2000.

[Genesereth, 1991] M. R. Genesereth: Knowledge Interchange Format. In Proceedings of the Second International Conference on the Principles of Knowledge Representation and Reasoning (KR-91), J. Allenet al., (eds), Morgan Kaufman Publishers, 1991. See also http://logic.stanford.edu/kif/kif.html.

[Grosso, 1999] W. E. Grosso, H. Eriksson, R. W. Fergerson, J. H. Gennari, S. W. Tu, and M. A. Musen. Knowledge Modeling at the Millennium (The Design and Evolution of Protege-2000). In the Proceedings of the Twelfth Workshop on Knowledge Acquisition, Modeling and Management (KAW-1999), Banff, Alberta, Canada, October 16-21, 1999.

[Gruber, 1993] T. R. Gruber: A Translation Approach to Portable Ontology Specifications, Knowledge Acquisition, 5:199—220, 1993.

[Horrocks & Patel-Schneider, 1999] I. Horrocks and P. F. Patel-Schneider: Optimizing description logic subsumption. Journal of Logic and Computation, 9(3):267–293, 1999.

[Klein et al., 2000] M. Klein, D. Fensel, F. van Harmelen, and I. Horrocks: The Relation between Ontologies and Schema-Languages: Translating OIL- Specifications to XML-Schema In: Proceedings of the Workshop on Applications of Ontologies and Problem-solving Methods, 14th European Conference on Artificial Intelligence ECAI-00, Berlin, Germany August 20-25, 2000.

[Lasilla, 1998] O. Lassila: Web Metadata: A Matter of Semantics, IEEE Internet Computing, 2(4), 1998

[Lenat & Guha, 1990] D. B. Lenat and R. V. Guha: Building large knowledge-based systems. Representation and inference in the Cyc project, Addison-Wesley, Reading, Massachusetts, 1990.

[Luke et al., 1996] S. Luke, L. Spector, and D. Rager: Ontology-Based Knowledge Discovery on the World-Wide Web. In Working Notes of the Workshop on Internet-Based Information Systems at the 13th National Conference on Artificial Intelligence (AAAI96), 1996.

[Mädche & Staab, 2000] A. Mädche and S. Staab: Mining Ontologies from Text. In Knowledge Acquisition, Modeling, and Management, Proceedings of the European Knowledge Acquisition Conference (EKAW-2000), R. Dieng et al. (eds.), Lecture Notes in Artificial Intelligence, LNAI, Springer-Verlag, October 2000.

[McGuiness et al., 2000] D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder: The Chimaera Ontology Environment. In the Proceedings of the The Seventeenth National Conference on Artificial Intelligence (AAAI 2000), Austin, Texas, July 30 - August 3, 2000.

Footnotes

[1]Division of Mathmatics & Computer Science,Vrije Universiteit Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, NL, dieter@cs.vu.nl, http://www.cs.vu.nl/~dieter

[2]Department of Computer Science, University of Maryland,College Park, MD 20742, USA, hendler@cs.umd.edu, http://www.cs.umd.edu/~hendler

[3]MIT Media Laboratory,20 Ames St. 305 A,Cambridge, MA 02139 USA, lieber@media.mit.edu, http://lieber.www.media.mit.edu/people/lieber

[4]DFKI GmbH,Stuhlsatzenhausweg 3,D-66123 Saarbrücken, Germany, wahlster@dfki.de, http://www.dfki.de/~wahlster/

[5]http://www.w3c.org

[6]Given the current situation, there will be many "standards" requiring interchange.

[7]http://www.ontoknowledge.org/oil

[8]Notice that we speak about the *semantic* web.

[9]http://www.daml.org

[10]http://www.cogsci.princeton.edu/~wn

[11]http://www.cyc.com/

[12]http://www.ontoprise.de

[13]http://www.aidministrator.nl

[14]Fixed prices turned up at the beginning of the 20th century to lower transaction costs. However, negotiations and auctions help to allocate resources more optimally. Still, the negotiation effort may outweigh the advantages and lead to unreasonably high demands on time (and transaction costs). Automated negotiation agents and auction houses dispel the argument of high transaction costs and allow optimized resource allocation.

[15]On-To-Knowledge is an European IST project, http://www.ontoknowledge.org.

[16]http://www.swisslife.ch

[17]http://www.bt.com/innovations/

[18]See further http://www.enersearch.se. Enersearch research affiliates and shareholders are spread over many countries: its shareholding companies include IBM (US), Sydkraft (Sweden), ABB (Sweden/Switzerland), PreussenElektra (Germany), Iberdrola (Spain), ECN (Netherlands), and Electricidade do Portugal

[19]http://www.cognit.no

[20]http:/www.aidministrator.nl

[21]http://www.semanticedge.com

[22]http://www.verticalnet.com

[23]http://www.contenteurope.com

I. Languages & Ontologies

2. SHOE: A Blueprint for the Semantic Web

Jeff Heflin[1], James Hendler[2], and Sean Luke[3]

1. Introduction

The World Wide Web is a vast repository of information, but its utility is restricted by limited facilities for searching and integrating this information. The problem of making sense of the Web has engaged the minds of numerous researchers from fields such as databases, artificial intelligence, and library science; and these researchers have applied numerous approaches in an attempt to solve it. Tim Berners-Lee, inventor of the Web, has coined the term *Semantic Web* to describe a vision of the future in which the "web of links" is replaced with a "web of meaning." In this paper, we examine the thesis that the "the Semantic Web can be achieved if we describe web resources in a language that makes their meaning explicit."

Any language for the Semantic Web must take into account the nature of the Web. Let's consider some of the issues that arise:

The Web is distributed. One of the driving factors in the proliferation of the Web is the freedom from a centralized authority. However, since the Web is the product of many individuals, the lack of central control presents many challenges for reasoning with its information. First, different communities will use different vocabularies, resulting in problems of synonymy (when two different words have the same meaning) and polysemy (when the same word is used with different meanings). Second, the lack of editorial review or quality control means that each page's reliability must be questioned. An intelligent web agent simply cannot assume that all of the information it gathers is correct and consistent. There are quite a number of well-known "web hoaxes" where information was published on the Web with the intent to amuse or mislead. Furthermore, since there can be no global enforcement of integrity constraints on the Web, information from different sources may be in conflict. Some of these conflicts may be due to philosophical disagreement; different political groups, religious groups, or nationalities may have fundamental differences in opinion that will never be resolved. Any attempt to prevent such inconsistencies must favor one opinion, but the correctness of the opinion is very much in the "eye of the beholder."

The Web is dynamic. The web changes at an incredible pace, much faster than a user or even a "softbot" web agent can keep up with. While new pages are being added, the content of existing pages is changing. Some pages are fairly static, others change on a regular basis and still others change at unpredictable intervals. These changes may vary in significance: although the addition of punctuation, correction of spelling errors or reordering of a paragraph does not affect the semantic content of a document, other changes may completely alter meaning, or even remove large amounts of information. A web agent must assume that its data can be, and often will be, out of date.

The rapid pace of information change on the Internet poses an additional challenge to any attempt to create standard vocabularies and provide formal semantics. As understanding of a given domain changes, both the vocabulary may change and the semantics may be refined. It is important that such changes do not adversely alter the meaning of existing content.

The Web is massive. Recent estimates place the number of indexable web pages at over 2 billion and this number is expected to double within a year. Even if each page contained only a *single* piece of agent-gatherable knowledge, the cumulative database would be large enough to bring most reasoning systems to their knees. To scale to the size of the ever growing Web, we must either restrict expressivity of the language or use incomplete reasoning algorithms.

The Web is an open world. A web agent is not free to assume it has gathered all available knowledge; in fact, in most cases an agent should assume it has gathered rather little available knowledge. Even the largest search engines have only crawled about 25% of the available pages. However, in order to deduce more facts, many reasoning systems make the closed-world assumption. That is, they assume that anything not entailed in the knowledge base is not true. Yet it is clear that the size and evolving nature of the Web makes it unlikely that any knowledge base attempting to describe it could ever be complete.

In an attempt to deal with these issues, we have designed a language named SHOE, for Simple HTML Ontology Extensions. SHOE is one of the first languages that allows ontologies to be designed and used directly on the World Wide Web [Luke et al., 1997]. In this paper we describe work that influenced SHOE, present an overview of the language, describe its syntax and semantics, and discuss how SHOE addresses the issues posed in this introduction. We then discuss the problem of implementing a system that uses SHOE, describe some tools that enhance the language's usability, and discuss the application of these tools to two different domains. Finally, we provide an overview of related work and some concluding remarks.

2. Background

The success of the Web was made possible by the Hypertext Markup Language (HTML). With HTML, people can easily author sharable documents and link to related documents that might exist on different systems. However, HTML is mostly concerned with presentation (i.e., how a document is displayed in a web browser), and it is difficult to automatically extract content from an HTML document.

In 1998, the World Wide Web Consortium (W3C) officially released the Extensible Markup Language (XML) [Bray et al., 1998], a simplified version of the Standard Generalized Markup Language (SGML) [ISO86] for the Web. XML allows special codes, called *tags*, to be embedded in a text data stream in order to provide additional information about the text, such as indicating that a certain word should be emphasized. Usually, each tag has a corresponding end-tag, with arbitrary text or other tags contained between them. The combination of a start-tag, end-tag and the data between is called an *element*. Some tags have additional properties, called *attributes*.

Unlike HTML, which precisely defines the structure and usage of a specific set of elements, XML allows users to define their own elements and attributes. Thus, users can create a document using content-specific as opposed to presentation-specific tags. If a document conforms to basic rules of XML, then it is said to be well-formed. If a document conforms to a common grammar, as specified by a document type definition (DTD), then is said to be valid. A DTD specifies valid elements, the contents of these elements, and which attributes may modify an element. Thus a DTD provides a syntax for an XML document, but the semantics of a DTD are implicit. That is, the meaning of an element in a DTD is either inferred by a human due to the name assigned to it, is described in a natural-language comment within the DTD, or is described in a document separate from the DTD. Humans can then build these semantics into tools that are used to interpret or translate the XML documents, but software tools cannot acquire these semantics independently. Thus, an exchange of XML documents works well if the parties involved have agreed to a DTD beforehand, but becomes problematic when one wants to search across the entire set of DTDs or to spontaneously integrate information from multiple sources [Heflin & Hendler, 2000c].

A number of approaches including information retrieval, wrappers, semi-structured databases, machine learning, and natural language processing have been applied to the problem of querying and understanding HTML and/or XML web pages. However, the lack of semantics in the sources and the lack of common human knowledge in the tools greatly limits the quality of the techniques. We support an alternative approach: that authors should explicitly associate semantics with the content they provide.

In order to provide meaning for data, the knowledge must be represented in some way. Knowledge representation is a sub-field of artificial intelligence concerned with such matters. The goal of knowledge representation is to provide structures that allow information to be stored, modified, and reasoned with, all in an efficient manner. Over time, numerous knowledge representation languages with different properties have evolved, from early languages such as KL-ONE [Brachman & Schmolze, 1985] and KRL [Bobrow & Winograd, 1977] to more recent languages such as LOOM [MacGregor, 1991] , Classic [Brachman et al., 1991], and CYC-L [Lenat & Guha, 1990].

One of the oldest knowledge representation formalisms is semantic networks. A semantic net represents knowledge as a set of nodes connected by labeled links. In such a representation, meaning is implied by the way a concept is connected to other concepts. Frame systems are another representation that is isomorphic to semantic networks. In the terminology of such systems, a frame is a named data object that has a set of slots, where each slot represents a property or attribute of the object. Slots can have one or more values; these values may be pointers to other frames.

Advanced semantic networks and frame systems typically include the notion of abstraction, which is represented using *is-a* and *instance-of* links. An *is-a* link indicates that one class is included within another, while an *instance-of* link indicates that a concept is a member of a class. These links have correlations in basic set theory: *is-a* is like the subset relation and *instance-of* is like the element of relation. The collection of *is-a* links specifies a partial order on classes; this order is often called a taxonomy or categorization hierarchy. The taxonomy can be used to generalize a concept to a more abstract class or to specialize a class to its more specific concepts. As demonstrated by the popularity of Yahoo and the Open Directory, taxonomies clearly aid users in locating relevant information on the Web.

Many researchers in knowledge representation have become interested in the use of ontologies [Gruber, 1993]. The term ontology, which is borrowed from philosophy, is defined as "a particular theory about being or reality." As such, an ontology provides a particular perspective onto the world, or some part there of. Where a knowledge representation system specifies how to represent concepts, an ontology specifies what concepts to represent and how they are interrelated. Most researchers agree that an ontology must include a vocabulary and corresponding definitions, but it is difficult to achieve consensus on a more detailed characterization. Typically, the vocabulary includes terms for classes and relations, while the definitions of these terms may be informal text, or may be specified using a formal language like predicate logic. The advantage of formal definitions is that they allow a machine to perform much deeper reasoning; the disadvantage is that these definitions are much more difficult to construct.

Numerous ontologies have been constructed, with varying scopes, levels of detail, and viewpoints. Noy and Hafner [Noy & Hafner, 1997] provide a good overview and comparison of some of these projects. One of the more prominent themes in ontology research is the construction of reusable components. The potential advantages of such components are that large ontologies can be quickly constructed by assembling and refining existing components, and integration of ontologies is easier when the ontologies share components. One of the most common ways to achieve reusability is to allow the specification of an inclusion relation that states that one or more ontologies are included in the new theory [Farquhar, Fikes & Rice, 1997], [Lenat & Guha, 1990]. If these relationships are acyclic and treat all elements of the included ontology as if they were defined locally then an ontology can be said to extend its included ontologies.

One attempt to apply ideas from knowledge representation to the Web is the Resource Description Framework (RDF) [Lassila, 1998]. The RDF data model is essentially a semantic network without inheritance: it consists of nodes connected by labeled arcs, where the nodes represent web resources and the arcs represent attributes of these resources. RDF can be embedded in Web documents using an XML serialization syntax, although its designers emphasize this is only one of many possible representations of the RDF model.

To allow for the creation of controlled, sharable, extensible vocabularies (i.e., ontologies) the RDF working group has developed the RDF Schema Specification [Brickley & Guha, 1999]. This specification defines a number of properties that have specific semantics. RDF Schema defines properties that are equivalent to the *instance-of* and *is-a* links commonly used in knowledge representation. It also provides properties for describing properties, including the specification of a property's domain and range, as well as any properties of which it is a subproperty.

Although RDF is an improvement over HTML and XML, it is insufficient for a Semantic Web language [Heflin & Hendler, 2000c]. In particular, it provides a very small set of semantic primitives and has relatively weak mechanisms for managing schema evolution. It is desirable that a Semantic Web language provides semantics that allow inferences beyond what is capable in RDF, but it is also important that the reasoning procedures for the language can scale to the volumes of data available on the Internet.

The field of deductive databases deals with combining inferential capabilities with the scalability of database systems. The datalog model is commonly used as the basis for describing deductive databases. It is similar to Prolog in that it consists entirely of Horn clauses, but differs in that it does not allow function symbols and is a strictly declarative language.[4] Datalog is based on the relational model but defines two types of relations: *extensional database* (EDB) relations are those predicates which are physically stored in the database, while *intensional database* (IDB) relations are those that can be computed from a set of logical rules.

Datalog restricts its Horn clauses to be *safe*, meaning that all of its variables are *limited*. Datalog defines "limited" as follows: variables are limited if they appear in an ordinary predicate of the rule's body, appear in an ‘=’ comparison with a constant, or appear in an ‘=’ comparison with another limited variable. Datalog's Horn clauses may depend on each other recursively. Datalog allows negation in a limited form called *stratified negation*, which we will not discuss here.

Datalog is relevant to the design of a Semantic Web language because it allows important classes of rules to be expressed while inference algorithms such as *magic sets*, which combine the best features of forward and backward chaining, provide efficient reasoning. Additionally, it seems reasonable to expect that the Web will consist of a large EDB and a comparatively small IDB, which is an ideal situation for a deductive database system.

3. The SHOE Language

SHOE combines features of markup languages, knowledge representation, datalog, and ontologies in an attempt to address the unique problems of semantics on the Web. It supports knowledge acquisition on the Web by augmenting it with tags that provide semantic meaning. The basic structure consists of *ontologies*, which define rules that guide what kinds of assertions may be made and what kinds of inferences may be drawn on ground assertions, and *instances* that make assertions based on those rules. As a knowledge representation language, SHOE borrows characteristics from both predicate logics and frame systems.

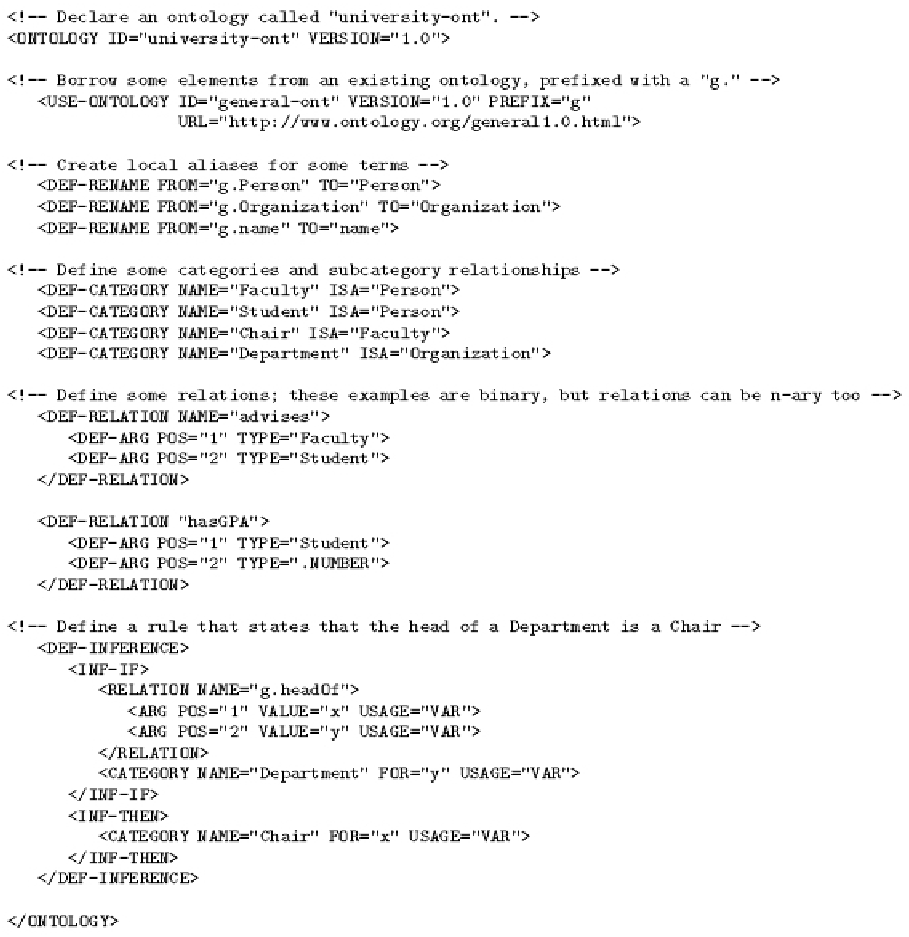
SHOE can be embedded directly in HTML documents or used in XML documents. There are a number of advantages to using an

XML syntax for SHOE. First, although more standard knowledge representation syntaxes, such as first-order logic or S-expressions, could be embedded between a pair of delimiting tags, Web authors are comfortable with XML-like, tag-based languages. Second, the XML syntax allows SHOE information to be analyzed and processed using the Document Object Model (DOM), allowing software that is XML-aware, but not SHOE-aware to still use the information in more limited but nevertheless powerful ways. For example, some web browsers are able to graphically display the DOM of a document as a tree, and future browsers will allow users to issue queries that will match structures contained within the tree. The third reason for using an XML syntax is that SHOE documents can then use the XSLT stylesheet standard [Clark, 1999] to render SHOE information for human consumption. This is perhaps the most important reason for an XML syntax, because it can eliminate the redundancy of having a separate set of tags for the human-readable and machine-readable knowledge.

In this section, we provide a brief description of the syntax of the language followed by a formal model and a discussion of SHOE's key features. The interested reader can find a detailed description of SHOE's syntax in the SHOE Specification [Luke & Heflin, 2000].

3.1. SHOE Ontologies

SHOE uses ontologies to define the valid elements that may be used in describing entities. Each ontology can reuse other ontologies by extending them. An ontology is stored in an HTML or XML file and is made available to document authors and SHOE agents by placing it on a web server. This file includes tags that identify the ontology, state which ontologies (if any) are extended, and define the various elements of the ontology. Figure 2.1 shows an example of a SHOE ontology.



```
<!-- Declare an ontology called "university-ont". -->
<ONTOLOGY ID="university-ont" VERSION="1.0">

<!-- Borrow some elements from an existing ontology, prefixed with a "g." -->
   <USE-ONTOLOGY ID="general-ont" VERSION="1.0" PREFIX="g"
                 URL="http://www.ontology.org/general1.0.html">

<!-- Create local aliases for some terms -->
   <DEF-RENAME FROM="g.Person" TO="Person">
   <DEF-RENAME FROM="g.Organization" TO="Organization">
   <DEF-RENAME FROM="g.name" TO="name">

<!-- Define some categories and subcategory relationships -->
   <DEF-CATEGORY NAME="Faculty" ISA="Person">
   <DEF-CATEGORY NAME="Student" ISA="Person">
   <DEF-CATEGORY NAME="Chair" ISA="Faculty">
   <DEF-CATEGORY NAME="Department" ISA="Organization">

<!-- Define some relations; these examples are binary, but relations can be n-ary too -->
   <DEF-RELATION NAME="advises">
      <DEF-ARG POS="1" TYPE="Faculty">
      <DEF-ARG POS="2" TYPE="Student">
   </DEF-RELATION>

   <DEF-RELATION "hasGPA">
      <DEF-ARG POS="1" TYPE="Student">
      <DEF-ARG POS="2" TYPE=".NUMBER">
   </DEF-RELATION>

<!-- Define a rule that states that the head of a Department is a Chair -->
   <DEF-INFERENCE>
      <INF-IF>
         <RELATION NAME="g.headOf">
            <ARG POS="1" VALUE="x" USAGE="VAR">
            <ARG POS="2" VALUE="y" USAGE="VAR">
         </RELATION>
         <CATEGORY NAME="Department" FOR="y" USAGE="VAR">
      </INF-IF>
      <INF-THEN>
         <CATEGORY NAME="Chair" FOR="x" USAGE="VAR">
      </INF-THEN>
   </DEF-INFERENCE>

</ONTOLOGY>
```

Figure 2.1 An example ontology.

In SHOE syntax, an ontology appears between the tags <ONTOLOGY ID=*id* VERSION=*version*> and <\ONTOLOGY>, and is identified by the combination of the *id* and *version*. An ontology can define categories, relations and other components by including special tags for these purposes.

The tag <DEF-CATEGORY> can be used to make *category definitions* that specify the categories under which various instances could be classified. Categories may be grouped as subcategories under one or more supercategories, essentially specifying the *is-a* relation that is commonly used in semantic networks and frame systems. The use of categories allows taxonomies to be built from the top down by subdividing known classes into smaller sets. The example ontology defines many categories, including Chair, which is a subcategory of Faculty.

The tag <DEF-RELATION> (which is closed by a <\DEF-RELATION> tag) can be used to make *relational definitions* that specify the format of n-ary relational claims that may be made by instances regarding instances and other data. One of the relationships defined by the example ontology is advises, which is between an instance of category Faculty and an instance of category Student. A relation argument can also be one of four basic types (string, number, date, or boolean value) as is the case with the second argument of the relationship hasGPA.

SHOE uses inference rules, indicated by the <DEF-INFERENCE> tag, to supply additional axioms. A SHOE inference rule consists of a set of antecedents (one or more subclauses describing claims that entities might make) and a set of consequents (consisting of one or more subclauses describing a claim that may be inferred if all claims in the body are made). The <INF-IF> and <INF-THEN> tags indicate the antecedents and consequents of the inference, respectively. There are three types of inference subclauses: category, relation and comparison. The arguments of any subclause may be a constant or a variable, where variables are indicated by the keyword VAR. Constants must be matched exactly and variables of the same name must bind to the same value. The ontology in the example includes a rule stating that the head of a department is a chair.

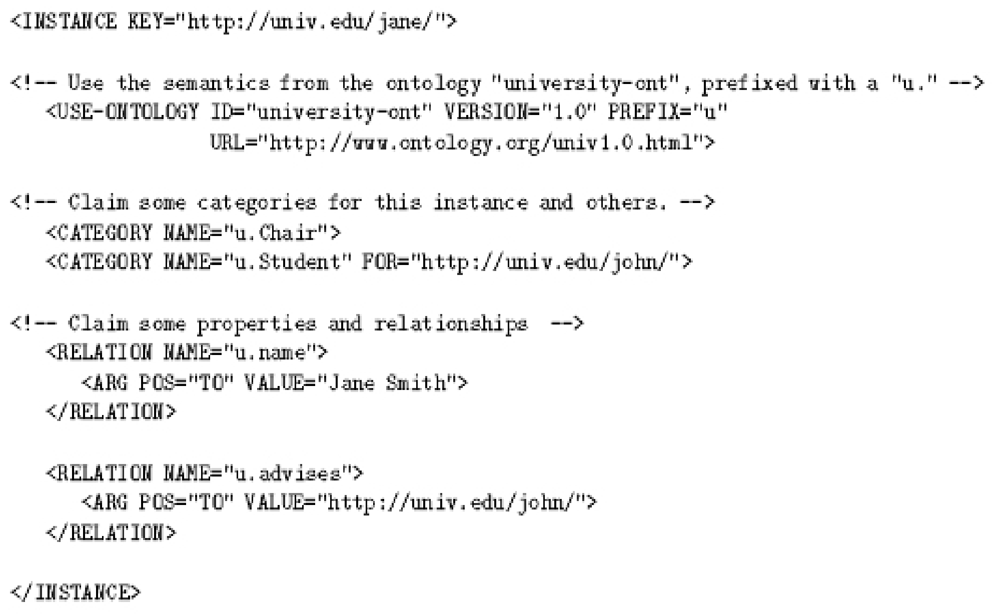
As is common in many ontology efforts, such as Ontolingua and Cyc, SHOE ontologies build on or extend other ontologies, forming a lattice with the most general ontologies at the top and the more specific ones at the bottom. Ontology extension is expressed in SHOE with the <USE-ONTOLOGY> tag, which indicates the id and version number of an ontology that is extended. An optional URL attribute allows systems to locate the ontology if needed and a PREFIX attribute is used to establish a short local identifier for the ontology. When an ontology refers to an element from an extended ontology, this prefix and a period is appended before the element's name. In this way, references are guaranteed to be unambiguous, even when two ontologies use the same term to mean different things. By chaining the prefixes, one can specify a path through the extended ontologies to an element whose definition is given in a more general ontology.

Sometimes an ontology may need to use a term from another ontology, but a different label may be more useful within its context. The <DEF-RENAME> tag allows the ontology to specify a local name for a concept from any extended ontology. This local name must be unique within the scope of the ontology in which the rename appears. Renaming allows domain specific ontologies to use the vocabulary that is appropriate for the domain, while maintaining interoperability with other domains.

3.2. SHOE Instances

Unlike RDF, SHOE makes a distinction between what can be said in an ontology and what can be said on an arbitrary web page. Ordinary web pages declare one or more instances that represent SHOE entities, and each instance describes itself or other instances using categories and relations. An example of a SHOE instance is shown in Figure 2.2.

The syntax for instances includes an <INSTANCE> element that has an attribute for a KEY that uniquely identifies the instance. We recommend that the URL of the web page be used as this key, since it is guaranteed to identify only a single resource. An instance commits to a particular ontology by means of the <USE-ONTOLOGY> tag, which has the same function as the identically named element used within ontologies. To prevent ambiguity in the declarations, ontology components are referred to using the prefixing mechanism described earlier. The use of common ontologies makes it possible to issue a single logical query to a set of data sources and enables the integration of related domains. Additionally, by specifying ontologies the content author indicates exactly what meaning he associates with his claims, and does not need to worry that an arbitrary definition made in some other ontology will alter this meaning.



```
<INSTANCE KEY="http://univ.edu/jane/">

<!-- Use the semantics from the ontology "university-ont", prefixed with a "u." -->
   <USE-ONTOLOGY ID="university-ont" VERSION="1.0" PREFIX="u"
                URL="http://www.ontology.org/univ1.0.html">

<!-- Claim some categories for this instance and others. -->
   <CATEGORY NAME="u.Chair">
   <CATEGORY NAME="u.Student" FOR="http://univ.edu/john/">

<!-- Claim some properties and relationships  -->
   <RELATION NAME="u.name">
      <ARG POS="TO" VALUE="Jane Smith">
   </RELATION>

   <RELATION NAME="u.advises">
      <ARG POS="TO" VALUE="http://univ.edu/john/">
   </RELATION>

</INSTANCE>
```

Figure 2.2 An example instance.

An instance contains ground *category claims* and *relation claims.* A category claim is specified with the <CATEGORY NAME=*y* FOR=*x*> tag, and says that the instance claims that an instance *x* is an element of a category *y*. If the FOR attribute is omitted, then the category claim is about the instance making the claim. In the example, the instance http://univ.edu/jane/ claims that http://univ.edu/jane/ is a Chair and http://univ.edu/john/ is a Student.

A relational claim is enclosed by the <RELATION NAME=*foo*> and <\RELATION> tags, and says that the instance claims that an n-ary relation *foo* exists between some *n* number of appropriately typed arguments consisting of data or instances. In the example, the instance http://univ.edu/jane/ claims that there exists the relation advises between http://univ.edu/jane/ and her student http://univ.edu/john/ and that that the name of http://univ.edu/jane/ is Jane Smith.

3.3. SHOE's Semantics

In order to describe the semantics of SHOE, we will extend a standard model-theoretic approach for definite logic with mechanisms to handle distributed ontologies. For simplicity, this model intentionally omits some minor features of the SHOE language.

We define an ontology *O* to be a tuple *<V, A>* where *V* is the vocabulary and *A* is the set of axioms that govern the theory. Formally, *V* is a set of predicate symbols, each with some arity > 0 and distinct from symbols in other ontologies,[5] while *A* is a set of definite program clauses that have the standard logical semantics.[6] We now discuss the contents of *V* and *A*, based upon the components that are defined in the ontology:

A <USE-ONTOLOGY> statement adds the vocabulary and axioms of the specified ontology to the current ontology. Due to the assumption that names must be unique, name conflicts can be ignored.

A <DEF-RELATION> statement adds a symbol to the vocabulary and for each argument type that is a category, adds an axiom that states that an instance in that argument must be a member of the category. If the tag specifies a name *R* and has *n* arguments then there is an *n*-ary predicate symbol *R* in *V*. If the type of the *i*th argument is a category *C*, then $[R(x1,...,xi,...xn) \rightarrow C(xi)] \in A$. This rule is a consequence of SHOE's open-world policy: since there is no way to know that a given object in a relation claim is *not* a member of a category appropriate for that relation, it is better to assume that this information is yet undiscovered than it is to assume that the relation is in error. However, when arguments are basic data types, type checking is performed to validate the relation. Basic data types are treated differently because they *are* different: they have syntax which can be checked in ways that category types cannot, which allows us to impose stringent input-time type checking on basic data types.

A <DEF-CATEGORY> adds a unary predicate symbol to the vocabulary and possibly a set of rules indicating membership.

If the name is *C*, then $C \in V$. For each super-category *Pi* specified, $[C(x) \rightarrow Pi(x)] \in A$.

A <DEF-INFERENCE> adds one or more axioms to the theory. If there is a single clause in the <INF-THEN>, then there is one axiom with a conjunction of the <INF-IF> clauses as the antecedent and the <INF-THEN> clause as the consequent. If there are *n* clauses in the <INF-THEN> then there are *n* axioms, each of which has one of the clauses as the consequent and has the same antecedent as above.

A <DEF-RENAME> provides an alias for a non-logical symbol. It is meant as a convenience for users and can be implemented using a simple pre-processing step that translates the alias to the original, unique non-logical symbol. Therefore, it can be ignored for the logical theory.

A formula *F* is well-formed with respect to *O* if 1) *F* is an atom of the form *p(t1,...,tn)* where *p* is a *n*-ary predicate symbol such that $p \in V$ or 2) *F* is a Horn clause where each atom is of such a form. An ontology is well-formed if every axiom in the ontology is well-formed with respect to the ontology.

Now we turn our attention to data sources, such as one or more web pages, that use an ontology to make relation and category claims. Let *S=<OS, DS>* be such a data source, where *OS = <VS, AS>* is the ontology and *DS* is the set of claims. *S* is well-formed if *OS* is well-formed and each element of *DS* is a ground atom that is well-formed with respect to *OS*. The terms of these ground atoms are constants and can be instance keys or values of a SHOE data type.

We wish to be able to describe the meaning of a given data source, but it is important to realize that on the Web, the same data could have different meanings for different people. An agent may be able to draw useful inferences from a data source without necessarily agreeing with the ontology intended by the data's author. A common case would be when an agent wishes to integrate information that depends on two overlapping but still distinct ontologies. Which set of rules should the agent use to reason about this data? There are many possible answers, and we propose that the agent should be free to choose. To describe this notion, we define a *perspective* $P = \langle S, O \rangle$ as a data source $S = \langle OS, DS \rangle$ viewed in the context of an ontology $O = \langle V, A \rangle$. If $O = OS$ then P is the *intended perspective*, otherwise it is an *alternate perspective*. If there are elements of DS that are not well-formed with respect to O, these elements are considered to be irrelevant to the perspective. If WS is the subset of DS that is well-formed with respect to O, then P is said to result in a definite logic theory $T = WS \cup A$.

Finally, we can describe the semantics of a perspective P using a model theoretic approach. An interpretation of the perspective consists of a domain, the assignment of each constant in S to an element of the domain, and an assignment of each element in *V* to a relation from the domain. A model of *P* is an interpretation such that every formula in its theory *T* is true with respect to it. We define a query on *P* as a Horn clause with no consequent that has the semantics typically assigned to such queries for a definite logic program *T*.

We also introduce one additional piece of terminology that will be used later in the paper. If every ground atomic logical consequence of perspective *P* is also a ground atomic logical consequence of perspective P' then P' is said to *semantically subsume P*. In such cases, any query issued against perspective P' will have at least the same answers as if the query was issued against *P*. If two perspectives semantically subsume each other, then they are said to be equivalent.

3.4. Interoperability in Distributed Environments

SHOE was designed specifically with the needs of distributed internet agents in mind. A key problem in distributed systems is interoperability; SHOE attempts to maximize interoperability through the use of shared ontologies, prefixed naming, prevention of contradictions, and locality of inference rules. This section discusses each of these in turn.

Figure 2.3 shows how the ontology extension and renaming features of the language promote interoperability. When two ontologies need to refer to a common concept, they should both extend an ontology in which that concept is defined.

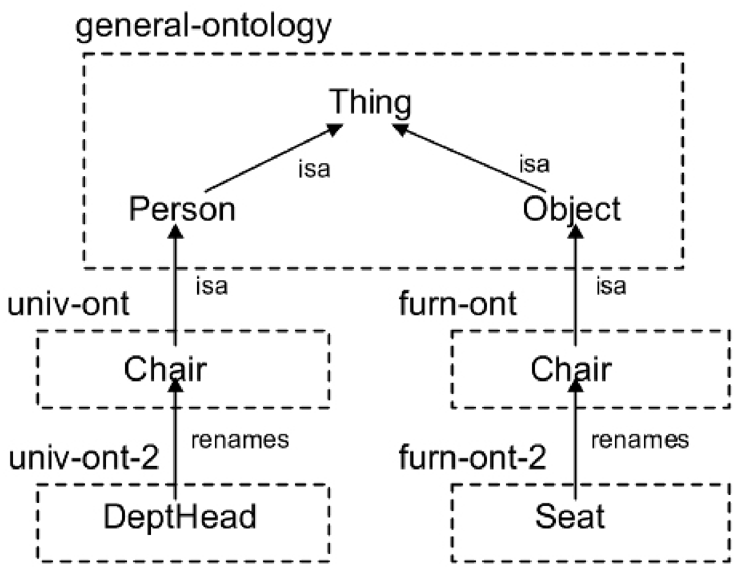


Figure 2.3 Interoperability is based on shared ontologies.

In this way, consistent definitions can be assigned to each concept, while still allowing communities to customize ontologies to include definitions and rules of their own for specialized areas of knowledge. These methods allow the creation of high-level, abstract unifying ontologies extended by often-revised custom ontologies for specialized, new areas of knowledge. There is a trade-off between trust of sources far down in the tree (due to their fleeting nature) and the ease of which such sources can be modified on-the-fly to accommodate new important functions (due to their fleeting nature). In a dynamic environment, an ontology too stable will be too inflexible; but of course an ontology too flexible will be too unstable. SHOE attempts to strike a balance using simple economies of distribution.

The problems of synonymy and polysemy are handled by the extension mechanism and <DEF-RENAME> tag. Using this tag, ontologies can create aliases for terms, so that domain-specific vocabularies can be used. For example, in Figure 2.3, the term DeptHead in univ-ont2 means the same thing as Chair in univ-ont due to a <DEF-RENAME> tag in univ-ont2. Although the extension and aliasing mechanisms solve the problem of synonymy of terms, the same terms can still be used with different meanings in different ontologies. This is not undesirable, a term should not be restricted for use in one domain simply because it was first used in a particular ontology. As shown in

Figure 2.3, in SHOE different ontologies may also use the same term to define a different concept. Here, the term Chair means different things in univ-ont and furn-ont because the categories have different ancestors. To resolve any ambiguity that may arise, ontological elements are always referenced using special prefixes that define unique paths to their respective enclosing ontologies. Instances and ontologies that reference other ontologies must include statements identifying which ontologies are used and each ontology is assigned a prefix which is unique within that scope. All references to elements from that ontology must include this prefix, thereby uniquely identifying which definition is desired.

Recall that each SHOE instance must be assigned a key, and that this key is often the URL of the web page describing the instance. In SHOE, it is assumed that each key identifies exactly one entity, but no assumptions are made about whether two distinct keys might identify the same entity. This is because many different URLs could be used to refer to the same page (due to the facts that a single host can have multiple domain names and operating systems may allow many different paths to the same file). To solve these problems in a practical setting, a canonical form can be chosen for the URL; an example rule might be that the full path to the file should be specified, without operating systems shortcuts such as '~' for a user's home directory. Even then, there are still problems with multiple keys possibly referring to the same conceptual object. At any rate, this solution ensures that the system will only interpret two objects as being equivalent when they truly are equivalent. Ensuring that two object references are matched when they conceptually refer to the same object is an open problem.

In distributed systems, a contradiction cannot be handled by simply untelling the most recent assertion, otherwise the system would give preference to those authors who provided their information first, regardless of whether it was true, false, or a matter of opinion. Rather than delve into complex procedures for maintaining consistency, we chose to keep SHOE easy to understand and implement. Therefore, we have carefully designed the language to eliminate the possibility of contradictions between agent assertions. SHOE does this in four ways:

- SHOE only permits assertions, not retractions.
- SHOE does not permit logical negation.
- SHOE does not have single-valued relations, that is, relational sets which may have only one value (or some fixed number of values).
- SHOE does not permit the specification of disjoint classes.

Although this restricts the expressive power of the language, in our practical experience, we have not yet found it to be a significant problem. It should be noted that SHOE does not prevent "contradictions" that are not logically inconsistent. If claimant *A* says *father(Mark, Katherine)* and claimant *B* says *father(Katherine, Mark)*, the apparent contradiction is because one claimant is misusing the *father* relation. However, this does not change the fact that *A* and *B* made those claims.

A similar problem may occur in an ontology where an inference rule derives a conclusion whose interpretation would be inconsistent with another ontology. Therefore, it is the ontology designer's responsibility to make sure that the ontology is correct and that it is consistent with all ontologies that it extends. It is expected that ontologies which result in erroneous conclusions will be avoided by users, and will thus be weeded out by natural selection.

Yet another problem with distributed environments is the potential interference of rules created by other parties: a rule created by one individual could have unwanted side-effects for other individuals. For these reasons, SHOE only allows rules to be defined in ontologies, and the only rules that could apply to a given claim are those which are defined in the ontologies used by the instance making claim. Since rules can only be expressed in ontologies, the process of determining when a rule is applicable is simplified, and page authors can use this to control the side-effects of their claims. If a user wishes to view an instance in a different context or use it in ways originally unintended by the author, then the user can use an alternate perspective for the instance that is based on a different, but compatible ontology.

3.5. Ontology Evolution

The Web's changing nature means that ontologies will have to be frequently changed to keep up with current knowledge and usage. Since physically revising an ontology can invalidate objects that reference it for vocabulary and definitions, it is useful to think of a revision as a new ontology that is a copy of the original ontology with subsequent modifications.

In fact, this is exactly what SHOE does: each version of an ontology is a separate Web resource and is assigned a unique version number, while all references to an ontology must denote a specific version. How then, is a revision different from an ontology with a different identifier? The answer is that a revision can specify that it is backwardly-compatible with earlier version (using the backward-compatible-with attribute of the ontology), which allows interoperability between sources that use different versions of an ontology.

Before we define backward-compatibility, we will first characterize and compare different types of revisions using the formal model developed in Section 3.3. To be succinct, we will only discuss revisions that add or remove components; the modification of a component can be thought of as a removal followed by an addition. In the rest of this section, O will refer to the original ontology, O' to its revision, P and P' to the perspectives formed by these respective ontologies and an arbitrary source $S = <O, DS>$, and T and T' to the respective theories for these perspectives.

If a revision O' adds an arbitrary rule to ontology O, then for any source S, the perspective P' semantically subsumes P. Since the revision only adds a sentence to the corresponding theory $T'\supset T$, and since first-order logic is monotonic any logical consequence of T is also a logical consequence of T'. Thus, when a revision that adds rules provides an alternate perspective of a legacy data source, there may be additional answers that were not originally intended by the author of the data. Similar reasoning is used to ascertain that if the revision removes rules, then *P* semantically subsumes *P′*.

If *O′* consists of the removal of categories or relations from *O*, then *P* semantically subsumes *P′*. This is because there may be some atoms in *S* that were well-formed w.r.t. *O* that are not well-formed w.r.t. *O′*. Informally, if categories or relations are removed, predicate symbols are removed from the vocabulary. If the ground atoms of *S* depended on these symbols for well-formedness then when the symbols are removed the sentences are no longer well-formed. Thus, $T' \subseteq T$ and due to the monotonicity of definite logic, every logical consequence of *T′* is a logical consequence of *T*. Revisions of this type may mean that using the revised ontology to form a perspective may result in fewer answers to a given query.

Finally, if the revision only adds categories or relations, the corresponding perspective *P′* is equivalent to *P*. Since $T' \supset T$ it is easy to show that *P′* semantically subsumes *P*. The proof of the other direction depends on the nature of the axioms added: $R(x1,...,xi,...xn) \rightarrow C(xi)$ for relations and $C(x) \rightarrow Pi(x)$ for categories. It also relies on the fact that due to the definitions of categories and relations, the predicate of each antecedent is a symbol added by the new ontology and must be distinct from symbols in any other ontology. Therefore any atoms formed from these predicates are not well-formed with respect to any preexisting ontology. Thus, there can be no such atoms in *S*, since *S* must be well-formed with respect to some ontology $\neq$ O′. Since the antecedents cannot be fulfilled, the rules will have no new logical consequences that are ground atoms. Since *P* semantically subsumes *P′* and vice versa, *P* and *P′* are equivalent. This result indicates that we can safely add relations or categories to the revision, and maintain the same perspective on all legacy data sources.

We can now define backward-compatibility: an ontology revision *O′* can be said to be backward-compatible with an ontology *O* if for any data source *S* = *<O, DS>*, the perspective *P′* = *<S, O′>* semantically subsumes the perspective *P* = *<S, O>*. Put simply, if every logical consequence of the original is also a consequence of the revision, then the revision is backward-compatible. By our analysis above, if a revision only adds categories, relations, or rules then it is backward compatible with the original, while if it removes any of these components then it is not backward compatible.

With this notion of backward compatibility, agents can assume with some degree of confidence that a perspective that uses the backward compatible revision will not alter the original meaning of the data source, but instead supplement it with information that was originally implicit. Agents that don't wish to assume anything, may still access the original version because it still exists at the original URL. However, it should be noted that this versioning mechanism is dependent on the compliance of the ontology designers. Since an ontology is merely a file on a web server, there is nothing to prevent its author from making changes to an existing ontology version. This is the price we pay for have having a system that is flexible enough to cope with the needs of diverse user communities while being able to change rapidly. However, we presume that users will gravitate towards ontologies from sources that they can trust and ontologies that cannot be trusted will become obsolete.

Although, ideally integration in SHOE is a byproduct of ontology extension, a distributed environment in which ontologies are rapidly changing is not always conducive to this. Even when ontology designers have the best intentions, a very specialized concept may be simultaneously defined by two new ontologies. To handle such situations, periodic ontology integration must occur. Ontologies can be integrated using a new ontology that maps the related concepts using inference rules, by revising the relevant ontologies to map to each other, or by creating a new more general ontology which defines the common concepts, and revising the relevant ontologies to extend the new ontology. We discuss each of these solutions in more detail in [Heflin & Hendler, 2000a].

3.6. Scalability

The scalability of a knowledge representation depends on the computational complexity of the inferences that it sanctions. We intentionally omitted from SHOE features such as disjunction and negation that typically make knowledge representation systems intractable. Since SHOE is essentially a set of Horn clauses, a naive forward-chaining inference algorithm can be executed in polynomial time and space in the worst case. Of course, the expected size of an extensional database built from the Web makes this an undesirable option.

We carefully chose SHOE's set of semantic primitives so that it could be reduced to datalog, thus allowing the use of optimized algorithms such as magic sets [Ullman, 1988]. The semantics of SHOE categories can be easily described using a datalog rule. For example, category membership such as the fact that a Person is a Mammal may be expressed by using unary predicates and a rule of the form:

Mammal (X) :- Person (x)

Since SHOE's inferential rules are basically Horn clauses, they also map directly to datalog. Furthermore, SHOE's more restrictive variable join rule ensures that all SHOE inference rules are safe. Thus, SHOE is equivalent to safe datalog without negation.

Obviously, SHOE systems can benefit from the deductive database research, but the massive size of the resulting KBs may at times yield unacceptable performance. Therefore SHOE has a modular design that allows systems to cleanly provide differing degrees of inferential capability. For example, a system may choose only to implement transitive category membership, or may choose to implement no inference at all, thus providing only access to the extensional database. Although such systems might be incomplete reasoners with respect to the intended perspective (see section 3.3), they can be complete reasoners with respect to an alternate perspective (i.e., one that mirrors the intended perspective but omits all inference rules and/or category definitions).

4. Implementation

In the previous section we described the semantics of SHOE and discussed its treatment of some of the specific challenges for a Semantic Web language. In this section we describe a suite of tools and methods for using the language in practice, and describe its application to the domains of computer science departments and food safety.

Although there are many possible ways to use the SHOE language, the simplest is one that parallels the way the Web works today. There are numerous tools that can be used to produce content, and this content is published on the Web by making it accessible via a web server. A web-crawler can then gather relevant pages and store it in a repository, which can then be queried by some type of user interface. The key component of a SHOE system is that both the content and the ontologies which provide semantics for the content are published on the Web. Since this information is structured and has semantics, the repository should be a knowledge base rather than an information retrieval system. This basic architecture is shown in Figure 2.4

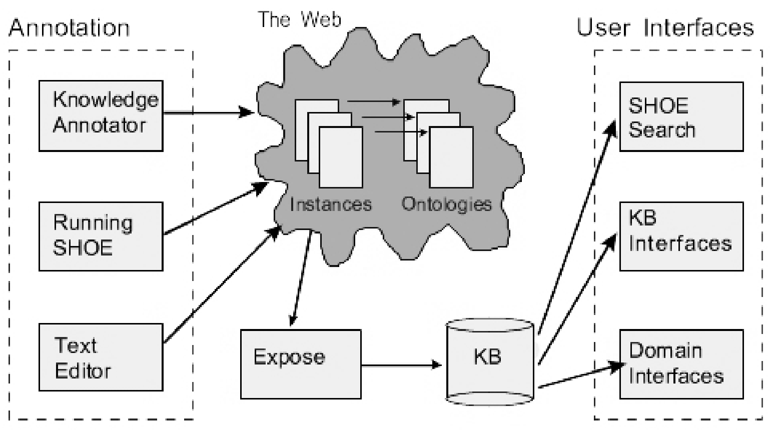


Figure 2.4 The SHOE architecture.

The first step in using SHOE is to locate or design an appropriate ontology. To assist in this process, there should be a central repository of ontologies. A simple repository could be a set of web pages that categorize ontologies, while a more complex repository may associate a number of characteristics with each ontology so that specific searches can be issued. A web-based system that uses the later approach is described in [Vega, 1999]. If it is determined that no suitable ontology is available, any newly created ontology should always extend available ontologies that contain related concepts.

The process of adding semantic tags to a web page is called annotation. There are a number of tools that can be used in this process, from simple text editors, to GUI-based editors, to semi- or fully-automated techniques. Text editors have the advantage that they are common place, but require that users become familiar with the syntax of SHOE. We have developed a GUI editor for SHOE called the Knowledge Annotator that allows the user to create markup by selecting from lists and filling in forms. However, there are cases where the user may want to generate markup that corresponds to information from large lists or tables in pre-existing web pages. For such situations, our Running SHOE tool can be used to quickly create a wrapper for an existing document by specifying tokens that indicate records and fields, and then mapping these records and fields to classes and relations from some ontology. This tool can be used to generate a large set of semantic markup from regular documents in minutes. Other approaches to generating markup can include machine learning, information extraction, and, in limited domains, natural language processing techniques.

Once the necessary content and ontologies documents are published on the Web, they can be harvested by a web-crawler. Exposé, the SHOE web-crawler, searches for web pages with SHOE markup and stores the information in a knowledge base.

Whenever a page commits to an ontology unknown to the system, this ontology is also retrieved and stored in the knowledge base. The biggest drawback to a web-crawler approach is that the information is only as recent as the last time the crawler visited the source web page. In certain applications, such as comparison shopping, this time period may be unacceptable. An interesting direction for future research is the development of focused crawlers that seek answers to particular questions in real-time.

As mentioned above, the web-crawler needs a knowledge base in which to store the results of its efforts. A knowledge base provides permanent storage for information and the ability to use knowledge to draw inferences from that which was explicitly stored in it. An important trade-off for candidate knowledge base systems is the degree to which it can make inferences sanctioned by the language (completeness) and the response time to queries (performance). We believe that users of SHOE systems should be able to specify their preferences with regard to this scale. Thus, many different knowledge base systems could be used for SHOE, with the appropriate tradeoffs in mind.

At the completeness end of the scale sit systems such as XSB [Sagonas et al., 1994], a logic programming and deductive database system. XSB is more expressive than datalog and can thus be used as a complete reasoner for SHOE. At the performance end of the scale sit relational database management systems (RDBMS), which have been traditionally used for querying enormous quantities of data. However, a relational database provides no automated inferential capability, and thus can only answer queries using the information that was explicitly stored. In between the two extremes, sit systems such as Parka [Evett, Andersen & Hendler, 1993], [Stoffel et al., 1997], a high-performance knowledge representation system whose roots lie in semantic networks and frame systems, but which makes use of certain database technology such as secondary storage and indexing. Parka has better inferential capabilities than a RDBMS, but less than XSB, while being faster than XSB but slower than an RDBMS.

Once information is stored in a knowledge base, it can be queried by a variety of front ends. SHOE Search [Heflin & Hendler, 2000b] is a generic tool gives users a new way to browse the web by allowing them to submit structured queries and open documents by clicking on the URLs in the results. The user first chooses an ontology against which the query should be issued and then chooses the class of the desired object from a hierarchical list. After the system presents a list of all properties that could apply to that object and the user has typed in desired values for one or more of these properties, the user issues a query and is presented with a set of results in a tabular form. If the user double-clicks on a binding that is a URL, then the corresponding web page will be opened in a new window of the user's web browser.

In order to evaluate the tools and techniques for SHOE, we have used them in two different domains. The first domain is that of computer science departments. We started by creating a simple computer science department ontology[7] by hand. The scope of this ontology includes departments, faculty, students, publications, courses, and research, for a total of 43 categories and 25 relations. The next step was to annotate a set of web pages (i.e., add SHOE semantic markup to them). Every member of the Parallel Understanding Systems (PLUS) Group marked up their own web pages. Although most members used the Knowledge Annotator, a few added the tags using their favorite text editors. To get even more SHOE information, we used the Running SHOE tool on the faculty, users, courses and research groups web pages from the web sites of various computer science departments. Exposé was used to acquire the SHOE knowledge from the web pages. This resulted in a total of 38,159 assertions, which were stored in both Parka and XSB. Although the KB for this domain is very small when compared to the scale of the entire Web, the initial results are promising. For example, a query of the form member(http://www.cs.umd.edu, x) $\wedge$ instance(Faculty, x) takes Parka less than 250 milliseconds to answer.

The possible benefits of a system such as this one are numerous. A prospective student could use it to inquire about universities that offered a particular class or performed research in certain areas. Or a researcher could design an agent to search for articles on a particular subject, whose authors are members of a particular set of institutions, and were published during some desired time interval. Additionally, SHOE can combine the information contained in multiple sources to answer a single query. For example, to answer the query "Find all papers about ontologies written by authors who are faculty members at public universities in the state of Maryland" one would need information from university home pages, faculty listing pages, and publication pages for individual faculty members. Such a query would be impossible for current search engines because they rank each page based upon how many of the query terms it contains.

The SHOE technology was also applied to the domain of food safety. The Joint Institute for Food Safety and Applied Nutrition (JIFSAN), a partnership between the Food and Drug Administration (FDA) and the University of Maryland, is working to expand the knowledge and resources available to support risk analysis in the food safety area. One of their goals is to develop a website that will serve as a clearinghouse of information about food safety risks. This website must serve a diverse group of users, including researchers, policy makers, risk assessors, and the general public, and thus must be able to respond to queries where terminology, complexity and specificity may vary greatly. This is not possible with keyword based indices, but can be achieved using SHOE.

The initial TSE ontology was fleshed out in a series of meetings that included members of JIFSAN and a knowledge engineer. The ontology focused on the three main concerns for TSE Risks: source material, processing, and end-product use. Currently, the ontology has 73 categories and 88 relations.[8] Following the creation of the initial ontology, the team annotated web pages. There are two types of pages that this system uses. Since the Web currently has little information on animal material processing, we created a set of pages describing many important source materials, processes and products. The second set of pages are existing TSE pages that provide general descriptions of the disease, make recommendations or regulations, and present experimental results. Early annotations were difficult because the original ontology did not have all of the concepts that were needed. When the initial set of pages was completed, we ran Exposé, using Parka as the knowledge base system. Since the TSE Ontology currently does not define inference rules, Parka is able to provide a complete reasoning capability for it. The Parka KB can be queried using SHOE Search as discussed earlier, but JIFSAN also wanted a special purpose tool to help users visualize and understand the processing of animal materials.

To accommodate this, we built the TSE Path Analyzer, a graphical tool that can be used to analyze how source materials end up in products that are eventually consumed by humans or animals. This information is extremely valuable when trying to determine the risk of contamination given the chance that a source material is contaminated. It is expected that information on each step in the process will be provided on different web sites (since many steps are performed by different companies), thus using a language like SHOE is essential to integrating this information. The TSE Path Analyzer allows the user to pick a source, process and/or end product from lists that are derived from the taxonomies of the ontology. The system then displays all possible pathways that match the query. Since these displays are created dynamically based on the semantic information in the SHOE web pages, they are kept current automatically, even when the SHOE information on some remote site is changed.

5. Related Work

In recent years, there has been work to use ontologies to help machines process and understand Web documents. Fensel et al. [Fensel et al., 1998] have developed Ontobroker, which proposes minor extensions to the common anchor tag in HTML. The theoretical basis for Ontobroker is frame logic, a superset of Horn logic that treats ontology objects as first class citizens. However, this approach depends on a centralized broker, and as a result, the web pages cannot specify that they reference a particular ontology, and agents from outside the community cannot discover the ontology information. Kent [Kent, 1999] has designed the Ontology Markup Language (OML) and the Conceptual Knowledge Markup Language (CKML), which were influenced by SHOE, but are based on the theories of formal concept analysis and information flow. However, the complexity of these theories make it unlikely that this language will be accepted by the majority of existing web developers and/or users. The Ontology Interchange Language (OIL) [Decker et al., 2000] is a new web ontology language that extends RDF and RDF Schema with description logic capabilities. Jannink et al. [Jannink et al., 1998] suggest a different approach from creating web ontology languages and annotating pages; they propose that an ontology should be built for each data source, and generalization is accomplished by integrating these data sources. In this way, the data dictates the structure of the ontology rather than the other way around.

Querying the Web is such an important problem that a diverse body of research has be directed towards it. Some projects focus on creating query languages for the Web [Arocena et al., 1997], [Konopnicki & Shemueli, 1995], but these approaches are limited to queries concerning the HTML structure of the document and the hypertext links. They also rely on index servers such as AltaVista or Lycos to search for words or phrases, and thus suffer from the limitations of keyword search. Work on semistructured databases [McHugh et al., 1997] is of great significance to querying and processing XML, but the semistructured model suffers the same interoperability problems as XML. Even techniques such as data guides will be of little use when integrating information developed by different communities in different contexts. Another approach involves mediators (or wrappers), custom software that serves as an interface between middleware and a data source [Wiederhold, 1992, Papakonstantinou, 1995, Roth & Schwarz, 1997]. When applied to the Web, wrappers allow users to query a page's contents as if it was a database. However, the heterogeneity of the Web requires that a multitude of custom wrappers must be developed, and it is possible that important relationships cannot be extracted from the text based solely on the structure of the document. Semi-automatic generation of wrappers [Ashish & Knoblock., 1997] is a promising approach to overcoming the first problem, but is limited to data that has a recognizable structure.

In order to avoid the overhead of annotating pages or writing wrappers, some researchers have proposed machine learning techniques. [Craven et al., 1998] have trained a system to classify web pages and extract relations from them in accordance with a simple ontology. However, this approach is constrained by the time-consuming task of developing a training set and has difficulty in classifying certain kinds of pages due to the lack of similarities between pages in the same class.

Conclusion

In this paper, we have described many of the challenges that must be addressed by research on the Semantic Web and have described SHOE, one of the first languages to explicitly address these problems. SHOE provides interoperability in distributed environments through the use of extensible, shared ontologies, the avoidance of contradictions, and localization of inference rules. It handles the changing nature of the Web with an ontology versioning scheme that supports backward-compatibility. It takes steps in the direction of scalability by limiting expressivity and allowing for different levels on inferential support. Finally, since the Web is an "open-world," SHOE does not allow conclusions to be drawn from lack of information.

To demonstrate SHOE's features, we have described applications that show the use of SHOE. We've developed a freely available ontology for computer science pages, and we've also worked with biological epidemiologists to design an ontology for a key food safety area. These applications show that SHOE can exist on the web, and that tools using SHOE can be built and used.

Although we believe SHOE is good language that has practical use, we do not mean to suggest that it solves all of the problems of the Semantic Web. We are at the beginning of a new and exciting research field and there is still much work to do. Further research must be performed to determine how SHOE's method for interoperability scales to the thousands of ontologies that will be created on the Web and the problem of ontology evolution must be studied more closely. Additionally, the appropriate level of expressivity for a semantic web language must be explored. For example, are negation and cardinality constraints necessary, and if so, how can they be used in a decentralized system such as the Web? Finally, more user-friendly tools need to be developed, so that use of the semantic web can become routine for the layperson.

As early "pioneers" we hope that our experience with SHOE can inspire and inform others. A key goal of this project is to raise the issues that are crucial to the development of the Semantic Web and encourage others to explore them. To this end, we have made SHOE freely available on the Web, including the Java libraries and our prototype tools. Interested readers are urged to explore our web pages at http://www.cs.umd.edu/projects/plus/SHOE/ for the full details of the language and the applications.

Acknowledgments

This work was supported by the Army Research Laboratory under contract number DAAL01-97-K0135 and the Air Force

Research Laboratory under grants F306029910013 and F306020020578.

References

[Arocena et al., 1997] G. Arocena, A. Mendelzon and G. Mihaila, Applications of a Web Query Language, in: Proceedings of ACM PODS Conference, Tuscon, AZ ,1997.

[Ashish & Knoblock., 1997] N. Ashish and C. Knoblock, Semi-automatic Wrapper Generation for Internet Information Sources, in: Proceedings of the Second IFCIS Conference on Cooperative Information Systems (CoopIS), Charleston, SC ,1997.

[Bobrow & Winograd, 1977] D. Bobrow and T. Winograd, An overview of KRL, a knowledge representation language, Cognitive Science 1(1) ,1977.

[Brachman & Schmolze, 1985] R. Brachman and J. Schmolze, An overview of the KL-ONE knowledge representation system, Cognitive Science, 9(2) ,1985.

[Brachman et al., 1991] R. Brachman, D. McGuinness, P.F. Patel-Schneider, L. Resnick,and A. Borgida, Living with Classic: When and how to use a KL-ONE-like language, in: J. Sowa, ed., Explorations in the representation of knowledge (Morgan-Kaufmann, CA, 1991).

[Bray et al., 1998] T. Bray, J. Paoli and C. Sperberg-McQueen, Extensible Markup Language (XML), W3C (World Wide Web Consortium), at: http://www.w3.org/TR/1998/REC-xml-19980210.html. 1998.

[Brickley & Guha, 1999] D. Brickley and R.V. Guha, Resource Description Framework (RDF) Schema Specification (Candidate Recommendation), W3C (World-Wide Web Consortium), 2000. (At http://www.w3.org/TR/2000/CR-rdf-schema-20000327)

[Clark, 1999] J. Clark, XSL Transformations (XSLT) W3C (World-Wide Web Consortium) (1999). (At http://www.w3.org/TR/1999/REC-xslt-19991116)

[Craven et al., 1998] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigram, and S. Slattery, Learning to Extract Symbolic Knowledge From the World Wide Web, in: Proceedings of the Fifteenth American Association for Artificial Intelligence Conference (AAAI-98) (AAAI/MIT Press, 1998).

[Decker et al., 2000] S. Decker, D. Fensel , F. van Harmelen , I. Horrocks, S. Melnik , M. Klein, and J. Broekstra, Knowledge Representation on the Web, in: Proceedings of the 2000 International Workshop on Description Logics (DL2000), Aachen, Germany, August 2000.

[Evett, Andersen & Hendler, 1993] M. Evett, W. Andersen and J. Hendler, Providing Computational Effective Knowledge Representation via Massive Parallelism, in: L. Kanal, V. Kumar, H. Kitano, and C. Suttner, eds., Parallel Processing for Artificial Intelligence, Elsevier Science, Amsterdam, 1993.

[Farquhar, Fikes & Rice, 1997] A. Farquhar, R. Fikes and J. Rice The Ontolingua Server: A tool for collaborative ontology construction, International Journal of Human-Computer Studies 46(6), 1997. 707-727.

[Fensel et al., 1998] D. Fensel, S. Decker, M. Erdmann und R. Studer: Ontobroker: The Very High Idea. In Proceedings of the 11th International Flairs Conference (FLAIRS-98), Sanibal Island, Florida, USA, 131-135, Mai 1998.

[Gruber, 1993] T. Gruber, A Translation Approach to Portable Ontology Specifications, in: Knowledge Acquisition 5, 1993, 199-220.

[Heflin & Hendler, 2000a] J. Heflin, and J. Hendler, Dynamic Ontologies on the Web, in: Proc. of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000), AAAI/MIT Press, Menlo Park, CA, 2000, 443-449.

[Heflin & Hendler, 2000b] J. Heflin and J. Hendler, Searching the Web with SHOE, in: Artificial Intelligence for Web Search. Papers from the AAAI Workshop. WS-00-01, AAAI Press, Menlo Park, CA, 2000, 35-40.

[Heflin & Hendler, 2000c] J. Heflin and J. Hendler, Semantic Interoperability on the Web, in: Proc. of Extreme Markup Languages 2000, Graphic Communications Association, Alexandria, VA, 2000, 111-120.

[ISO86] ISO (International Organization for Standardization) ISO 8879:1986(E). Information processing -- Text and Office Systems -- Standard Generalized Markup Language (SGML), International Organization for Standardization, Genevea, 1986.

[Jannink et al., 1998] J. Jannink, S. Pichai, D. Verheijen, and G. Wiederhold., G. Encapsulation and Composition of Ontologies, in: AI and Information Integration, Papers from the 1998 Workshop, Technical Report WS-98-14, AAAI Press, Menlo Park, CA, 1998, 43-50.

[Kent, 1999] R.E. Kent. Conceptual Knowledge Markup Language: The Central Core, in: Twelfth Workshop on Knowledge Acquisition, Modeling and Management, 1999.

[Konopnicki & Shemueli, 1995] D. Konopnicki and O. Shemueli, W3QS: A Query System for the World Wide Web, in: Proceedings of the 21st International Conference on Very Large Databases, Zurich, Switzerland, 1995.

[Lassila, 1998] O. Lassila, Web Metadata: A Matter of Semantics, IEEE Internet Computing 2(4), 1998, 30-37.

[Lenat & Guha, 1990] D. Lenat and R. Guha, Building Large Knowledge Based Systems, Addison-Wesley, MA, 1990.

[Luke & Heflin, 2000] S. Luke and J. Heflin, SHOE 1.01, Proposed Specification, at: http://www.cs.umd.edu/projects/plus/SHOE/spec.html, 2000.

[Luke et al., 1997] S. Luke, L. Spector, D. Rager, and J. Hendler, Ontology-based Web Agents, in: Proceedings of the First International Conference on Autonomous Agents, Association of Computing Machinery, New York, NY, 1997, 59-66.

[MacGregor, 1991] R. MacGregor, The Evolving Technology of classification-based knowledge representation systems, in: J. Sowa, ed., Explorations in the representation of knowledge, Morgan-Kaufmann, CA, 1991.

[McHugh et al., 1997] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom, Lore: A Database Management System for Semistructured Data, SIGMOD Record 26(3), 1997, 54-66.

[Noy & Hafner, 1997] N. Noy and C. Hafner, C. The State of the Art in Ontology Design. AI Magazine 18(3), 1997, 53-74.

[Papakonstantinou, 1995] Y. Papakonstantinou, et al., A Query Translation Scheme for Rapid Implementation of Wrappers, in: Proceedings of the Conference on Deductive and Object-Oriented Databases(DOOD) Singapore ,1995.

[Roth & Schwarz, 1997] M. Roth, and P. Schwarz, Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources, in: Proceedings of 23rd International Conference on Very Large Data Bases ,1997.

[Sagonas et al., 1994] K. Sagonas, T. Swift, and D. S. Warren, XSB as an Efficient Deductive Database Engine, in: R. T. Snodgrass and M. Winslett, editors, Proc. of the 1994 ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'94), 1994, 442-453.

[Stoffel et al., 1997] K. Stoffel, M. Taylor and J. Hendler, Efficient Management of Very Large Ontologies, in: Proceedings of American Association for Artificial Intelligence Conference (AAAI-97), AAAI/MIT Press, 1997.

[Ullman, 1988] J. Ullman, Principles of Database and Knowledge-Base Systems, Computer Science Press, MD, 1988.

[Vega, 1999] J. Vega, A. Gomez-Perez, A. Tello, and Helena Pinto, How to Find Suitable Ontologies Using an Ontology-Based WWW Broker, in: International Work-Conference on Artificial and Natural Neural Networks, IWANN'99, Proceeding, Vol. II, Alicante, Spain, 1999, 725-739.

[Wiederhold, 1992] G. Wiederhold, Mediators in the Architecture of Future Information Systems, IEEE Computer 25(3), 1992.

Footnotes

[1]Department of Computer Science, University of Maryland,College Park, MD 20742, USA, heflin@cs.umd.edu, http://www.cs.umd.edu/~heflin

[2]Department of Computer Science, University of Maryland,College Park, MD 20742, USA, hendler@cs.umd.edu, http://www.cs.umd.edu/~hendler

[3]Department of Computer Science Science and Technology Building II, 4400 University Drive, MSN 4A5, Fairfax, VA 22030 USA , seanl@cs.umd.edu, http://www.cs.gmu.edu/~sean/

[4]Prolog is not strictly declarative because the order of the rules determines how the system processes them.

[5]In actuality, SHOE has a separate namespace for each ontology, but one can assume that the symbols are unique because it is always possible to apply a renaming that appends a unique ontology identifier to each symbol.

[6]A definite program clause is a Horn clause that has at least one antecedent and exactly one consequent.

[7]This ontology is located at http://www.cs.umd.edu/projects/plus/SHOE/onts/cs1.0.html

[8]Those interested in the details of the ontology can view it at http://www.cs.umd.edu/projects/plus/SHOE/onts/tseont.html

3. DAML-ONT: An Ontology Language for the Semantic Web

Deborah McGuinness[1],Richard Fikes[2],Lynn Andrea Stein[3],

James Hendler[4]

1. Introduction

The DARPA Agent Markup Language (DAML) initiative is aimed at supporting the development of the semantic web. The program funds research in languages, tools, infrastructure, and applications for making web content more accessible and understandable. It is US Government-funded yet represents collaborations between the Department of Defense, US and European academia and business, and international consortia such as the W3C. While the program covers the breadth of issues related to markup language development, deployment, and evolution, this paper focuses only on the markup language itself.

This language is being developed in two pieces. The first portion—covered in this paper—is the ontology language, aimed at capturing definitions of terms—classes, subclasses, their properties, their restrictions, and individual object descriptions. The second portion of the language (called DAML-LOGIC) will address the issue of encoding inference and general logical implications.

In this paper, we review the history and motivations for the development of the initial DAML ontology language, DAML-

ONT. In the following section, we introduce the DAML-ONT language syntax and usage through a pedagogically ordered set of examples derived from the initial DAML walkthrough document [Stein and Connolly, 2000]. In order to fully specify a knowledge representation language, one needs to describe both the syntax and the semantics of the language. The syntax description specifies what strings of characters are legal statements in the language. The semantic description specifies the intended meaning of each legal statement in the language. In the final section of this paper, we explore an axiomatic semantics for DAML-ONT.

The DAML ontology language takes its motivation from many places, most notably the evolving web languages—in particular RDF [Lassila, 1998, Lassila-Swick, 1999] (with the embedded XML) and RDFS [Brickley-Guha, 2000], jointly referred to in this paper as RDF/S. It is important to be backwards compatible with existing web standard for interoperability with the growing user base of content, tools for these languages, and users who are comfortable with the languages. All of our examples below thus have the format of XML-based RDF. DAML-ONT extends RDF/S by capturing semantic relations in machine-readable form through more expressive term descriptions along with precise semantics. This is important for many reasons; arguably the most salient is to facilitate intercommunication between agents. While compatibility with web languages was paramount, we also recognized that markup representational needs went beyond what was conveniently expressible in RDF/S. Thus, an extended language was considered.

The language is also influenced by frame-based systems, including knowledge representation languages such as Ontolingua [Farquhar et. al, 1997] or KEE. Frame systems have enjoyed acceptance and perceived ease of use by broad populations and have been embraced relatively widespread use [Fikes & Kehler 1985, Karp 1992, Chaudhri et. al. 1998]. The goal of our language is to be accessible to the masses and thus it was important to use paradigms that are easy to explain and use.

Finally, DAML-ONT takes motivation from the field of description logics (www.dl.kr.org), which provide a formal foundation for frame-based systems. Some early description-logic based systems include KL-ONE [Brachman-Schmolze, 1985], CLASSIC [Borgida et. al, 1989], and LOOM [MacGregor, 1991], and a more recent example of a description logic-based system is OIL [Fensel et al., 2000, Bechhofer et al., 2000]. Description logics emphasize clear, unambiguous languages supported by complete denotational semantics and tractable reasoning algorithms. Description logics have been heavily analyzed in order to understand how constructors interact and combine to impact tractable reasoning. See for example, [Donini et al., 1991a, Donini et al., 1991b] for early evaluations. Also, reasoning algorithms have been studied producing knowledge about efficient reasoning algorithms (See [Horrocks and Patel-Schneider, 1999] and [Horrocks et al., 1999] for example). DAML-ONT draws on the general field of research in description logics and, in particular, on the latest description logic: OIL. OIL was designed to be an expressive description logic that is integrated with modern web technology.

The resulting DAML ontology language is a combination of these three building blocks along with influence from KIF—the Knowledge Interchange Format—a first order logic-based proposed ANSI standard, SHOE—Simple HTML Ontology Language, and OKBC—Open Knowledge Base Connectivity—a standard applications programming interface for knowledge systems. The initial proposal for the language was written by MIT and W3C DAML contractors [Berners-Lee et al., 2000]. It was subsequently taken over by a DAML language committee which, in turn, expanded to become the Joint US/EU ad hoc Agent Markup Language Committee. The joint committee has had responsibility for all DAML ontology language releases to date. It is also anticipated that there will be a W3C ontology committee (under the Semantic Web Activity ((http://www.w3.org/2001/sw/)) that will have responsibility for future semantic web language releases.

It is worth noting that after DAML-ONT was released, the joint committee undertook a major effort to evolve the language. The initial language was heavily influenced by the web languages. A primary initial goal of DAML-ONT was to provide a web-compatible language expressive enough to handle markup requirements. Constructors were chosen initially from use-case analysis from experience with the current web markup languages—mostly RDF and XML. A second primary goal was to produce a language quickly so that experimentation could inform further development. An ontology library was also formed rapidly so that DAML ontologies could be submitted, stored, and reused by a larger community. Of these competing influences, web language compatibility and timeliness were the first concerns. Usability issues, as informed by frame languages, were a secondary concern, and formal foundations—as found in description logics—came later. In a subsequent effort, Fikes and McGuinness produced an axiomatic semantics for DAML-ONT [Fikes-McGuinness, 2001], providing a more precise foundation for semantic analysis of the language. Simultaneously, the joint committee undertook a concerted effort to improve compatibility of DAML-ONT with more formal foundations. The resulting language, which was essentially a merging of DAML-ONT and OIL, is called DAML+OIL. It places much more emphasis on clear semantics for the language (provided both by our updated axiomatic semantics (http://www.daml.org/2001/03/axiomatic-semantics.html) along with a model-theoretic semantics (*http://www.daml.org/2001/03/model-theoretic-semantics.html*)). DAML+OIL also filters language constructors according to understanding of the impact that they have on reasoning algorithms. The resulting language, DAML+OIL, chooses its constructors carefully following the analysis done on the underlying formal description logic, typically referred to as *SHIQ*.[5]

An update to this paper that begins with DAML+OIL is in preparation and will be available from www.ksl.stanford.edu/people/dlm/.

2. An Introduction Through Examples

A language meant to capture terms and their meanings will need to be able to describe classes of objects, relations between objects, and ground objects in the domain of discourse. We will introduce the basic notions of the DAML ontology language through example. Readers interested in more information can also consider the following: The DAML web site (*www.daml.org*) contains the full specification of the language along with an example file and an annotated walk through. There is also a DAML ontology library along with numerous links to related work. Also, the OIL documentation is useful in particular for the language specification, documentation, semantics, and for use cases. The white paper[Bechhofer et al,2000], available on the OIL site provides a nice introduction to OIL and is available from *http://www.ontoknowledge.org/oil/*. Also, available from the OIL site is the denotational semantics specification for OIL which was the starting point for the denotational semantics for DAML+OIL. Later in this paper, we provide our axiomatic semantics for DAML-ONT (which is of course the starting place for the axiomatic semantics for DAML+OIL.

The following introduction through extended example is motivated by the walkthrough available from the daml home page.

2.1. Defining Classes and Properties

In order to describe objects, it is useful to define types for the objects. For example, we may be interested in describing people and animals. First, some general classes should be defined. The first class defined is named animal

```
<Class ID="Animal">
```

The Class tag is used to state that there is a class known as Animal. It does not say anything else about what an animal is other than specifying that ID. It is also not (necessarily) the sole source of information about Animals. By saying that the ID is Animal, it becomes possible for future sentences to refer to the definition of Animal given here. (This is done using the URI of the containing page followed by #Animal.) Thus, it is possible to add information to the definition of Animal at a later point.

Next, it may be desirable to annotate terms with labels and comments:

```
<label>Animal</label>
```

```
<comment>This class of animals is illustrative of a number of ontological idioms.</comment>
```

These two lines introduce a label -- a brief identifier of the enclosing element, suitable for graphical representations of RDF, etc. -- and a comment -- a natural language (English, in this case) description of the element within which it is included. Neither a label nor a comment contributes to the logical interpretation of the DAML.

```
</Class>
```

This closes the current definition of the Class Animal.

It may be desirable to define types of animals, named Male and Female.

```
<Class ID="Male">
     <subClassOf resource="#Animal"/>
</Class>
```

The subClassOf element asserts that its subject -- Male -- is a subclass of its object -- the resource identified by #Animal.

When we define Female, we may want to state that no one can be simultaneously a Male and a Female. This is done using the disjointFrom tag in combination with the subClassOf tag below.

```
<Class ID="Female">
     <subClassOf resource="#Animal"/>
     <disjointFrom resource="#Male"/>
</Class>
```

2.2. Defining Individuals

The syntax that we’ve used to define classes can also be used to define individuals. For example, imagine that we want to say that Fred is a Male. Here we provide the syntax for defining the individual Fred. This piece of DAML begins with the type of the thing we’re describing, in this case Male:

```
<Male ID="Fred">
     <label>Fred</label>
     <comment>Fred is a Male.</comment>
</Male>
```

The label and comment attributes are attached to this individual, Fred, but neither carries any semantics for the computer.

At this point, we can see the first benefit to be had from inference. Because we know that Fred is Male and because we are aware of the subclass relationship between Male and Animal, we know that Fred is also of type Animal.

2.3. Relating Individuals through Properties

Properties are used to relate items to each other. In this case we will be interested in connecting two animals via the parent property.

```
<Property ID="parent">
```

The property definition begins similarly to the Class: There is a property called parent. Note, however, that this is not a closing tag; there's more to this definition. (There is a matching </Property> tag below.)

We may want to say how many parents an animal can have. In this case we will state that things that have parents have exactly two parents.

```
<cardinality>2</cardinality>
```

Embedded elements, such as this one, are understood to describe their enclosing elements. So, this cardinality element describes the Property whose ID is parent.

We may want to state that parent is a property that applies only to things that are animals.

```
<domain resource="#Animal"/>
```

This element also describes the Property whose ID is parent. It says that the domain of the parent relation is Animal. This is done by asserting that the domain (of the property with ID parent) is the resource known as #Animal. All properties may have domains and ranges specified. The resource attribute is used to refer to a "reference-able" item created with the ID tag.

Each of the names defined so far—Animal, Person, Male, and Female -- refers to a name in this (i.e., the containing) document, since each reference begins with a #.

```
</Property>
```

This closes the property whose ID is parent.

2.4. Describing Attributes of Classes

Now we will define a Class with an attribute.

```
<Class ID="Person">
<subClassOf resource="#Animal"/>
```

A Person is a kind of Animal. (See the definition of Animal, above.)

The next few lines describe a domain-specific range restriction. The parent of a Person is also a Person.

```
<restrictedBy>
     <Restriction>
        <onProperty resource="#parent"/>
        <toClass resource="#Person"/>
     </Restriction>
</restrictedBy>
```

The syntax used here is a cliché, i.e., it is almost always used as shown, except for the name of the resource in the OnProperty element, which will give the name of the Property to be restricted, and the resource associated with the toClass element, which will give the name of the Class to which the Property is restricted. This is also sometimes referred to as a value restriction since the type of the parent value is being restricted.

```
</Class>
```

That is the end of the Person class definition at this point.

Suppose that we have a Person, Joe.

```
<Person ID="Joe">
     <label>Joe</label>
     <comment>Joe is a person.</comment>
</Person>
```

As we know, since Joe is stated to be of type Person, he will also be of type Animal (because of the subClassOf relationship). He will also have exactly two parents because of the cardinality statement. Those objects that fill his parent property also are of type person (thus, they also are of type animal and also have exactly two parents who are of type Person).

We might also want to define Man as a kind of Male Person and Female as a kind of Female Person.

This can be done as follows:

```
<Class ID="Man">
     <subClassOf resource="#Person"/>
     <subClassOf resource="#Male"/>
</Class>
<Class ID="Woman">
     <subClassOf resource="#Person"/>
     <subClassOf resource="#Female"/>
</Class>
```

The upcoming release will allow the exact definition of Man as exactly those things that are both person and male.

Thus if something is known to be an instance of a male and a person, then it would be inferred to be of type Man. The definition above only uses the subClassOf relation which states that a Man is simultaneously a subclass of Person and Male but it does not state that everything that is simultaneously a Person and a Male is a Man.

2.5. Using Properties

The next several annotations illustrate features of properties:

Father is a property that is a kind of parent property, i.e., x's father is also x's parent. In addition, range is used to ensure that x's father must be Male, and that x has only one father.

```
<Property ID="father">
     <subProperty resource="#parent"/>
     <range resource="#Man"/>
     <cardinality>1</cardinality>
</Property>
```

At this point if Joe is stated to be John’s father, then we can determine that Joe is of type Male because of the range restriction. Because Male is of type animal, Joe is also of type animal. Also, parents have a domain of person, thus John is of type person. He has exactly one father, now known to be Joe. Joe must further be of type Person, since Person has a domain specific range restriction.

Mother is defined similarly to father but using a variant notation. A UniqueProperty is one with cardinality 1, so we can omit that sub-element from Mother's definition.

```
<UniqueProperty ID="mother">
     <subProperty resource="#parent"/>
     <range resource="#Woman"/>
</UniqueProperty>
```

Sometimes, synonyms are useful. For example, some applications may want to use the term "mom" rather than "mother". The tag equivalentTo allows us to establish this synonymy:

```
<Property ID="mom">
     <equivalentTo resource="#mother"/>
</Property>
```

Inverse relationships are supported as well using the inverseOf tag. If x's parent is y, then y is x's child.

```
<Property ID="child">
     <inverseOf resource="#parent"/>
</Property>
```

Transitive relationships are supported using the TransitiveProperty tag. The ancestor and descendent properties are transitive versions of the parent and child properties. We would need to introduce additional elements to enforce these connections.

```
<TransitiveProperty ID="ancestor">
</TransitiveProperty>
```

```
<TransitiveProperty ID="descendant"/>
```

The cardinality property is exact. But sometimes we want to bound cardinality without precisely specifying it. A person may have zero or one job, no more:

```
<Property ID="occupation">
     <maxCardinality>1</maxCardinality>
</Property>
```

In the upcoming version of DAML+OIL, cardinality may be stated in a class-dependent manner instead of requiring the cardinality statement to be universally applied to the property.

Classes, too, can be annotated in various ways:

```
<Class ID="Car">
     <comment>no car is a person</comment>
     <subClassOf>
```

The thing that Car is a subClassOf could in principle be specified using a resource= attribute. In this case, however, there is no preexisting succinct name for the thing we want. A car is a kind of non-person. We build this by introducing a new -- anonymous -- Class definition described using the complementOf tag:

```
       <Class>
         <complementOf resource="#Person"/>
```

From the inside out: There's a thing that's the complement of Person, i.e., all non-Persons.

```
       </Class>
```

That thing is a Class.

</subClassOf>

That thing -- the Class of all non-Persons -- has a subClass.

</Class>

The Class with ID Car is the thing that is a subClass of the Class of all non-Persons.

(There's a similar construction from the outside in: Car is a Class that is a specialization of another Class, the Class that is left when you consider everything except Persons).

The next example shows an instance of further specifying a previously defined element by using the about attribute. These new assertions about the thing described above with ID Person have no more and no less authority than the assertions made within the <Class ID="Person"> element. (Of course, if the two assertions were in different documents, had different authors, etc., we might want to accord them different authority, but this would be as a result of information *about* those assertions rather than inherently from the assertions themselves.)

In this case, we identify the Class Person with the disjoint union of the Classes Man and Woman. Note that the disjointUnionOf element contains two subelements, the Class Man and the Class Woman. The parseType= "daml:collection" indicates that these subelements are to be treated as a unit, i.e., that they have special RDF-extending meaning within the disjointUnionOf.

```
<Class about="#Person">
     <comment>every person is a man or a woman</comment>
     <disjointUnionOf parseType="daml:collection">
       <Class about="#Man"/>
       <Class about="#Woman"/>
     </disjointUnionOf>
</Class>
```

A Person has a height, which is a Height. (hasHeight is a Property, or relation; Height is a Class, or kind of thing.)

```
<Property ID="hasHeight">
     <domain resource="#Person"/>
     <range resource="#Height"/>
</Property>
```

Height is Class described by an explicitly enumerated set. We can describe this set using the oneOf element. Like disjointUnionOf, oneOf uses the RDF-extending parsetype="daml:collection".

```
<Class ID="Height">
     <oneOf parseType="daml:collection">
       <Height ID="short"/>
       <Height ID="medium"/>
       <Height ID="tall"/>
     </oneOf>
</Class>
```

3. Notes

Of course by giving an introduction by example, we have not covered all portions of the language. Brief mention is made here to some other topics. See *www.daml.org* for more complete information.

DAML-ONT is consistent with the RDF namespace scheme, thus one can refer to RDF namespaces and create a base namespace for work. Ontologies will begin with an RDF start tag and then include a specification of appropriate namespaces. The following is typical of a namespace declaration:

```
<rdf:RDF
  xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns     ="http://www.daml.org/2000/10/daml-ont#"
  xmlns:daml="http://www.daml.org/2000/10/daml-ont#"
  >
```

In this document, the rdf: prefix should be understood as referring to things drawn from the namespace called http://www.w3.org/1999/02/22-rdf-syntax-ns#. This is a conventional RDF declaration appearing verbatim at the beginning of almost every rdf document[6]. The second declaration says that unprefixed element names in this document refer to terms in the namespace called http://www.daml.org/2000/10/daml-ont#. This is a conventional DAML-ONT declaration that should appear verbatim at the beginning of the current generation of DAML-ONT documents. The third declaration binds the daml: prefix to the same namespace name. So, for example, <Thing/> and <daml:Thing/> refer to the same thing in this document[7].

This set of three namespaces would come at the beginning of the file and then there would be a trailing closing tag at the end of the file: </rdf:RDF>.

One can also state information about version info, import from other resources, etc. for example:

<versionInfo>$Id: daml-ex.daml,v 1.2 2000/10/07 03:21:17 connolly Exp $</versionInfo>

<comment>An example ontology</comment>

<imports resource="http://www.daml.org/2000/10/daml-ont"/>

4. Language Extensions

When DAML-ONT was released, its authors were conscious that some expressive extensions would be made. Integrating a description logic-like methodology into DAML-ONT provided it with a more formal foundation. Subsequently, two major features were added through expressive extensions. First, term definitions were introduced, allowing the definition of a term exactly equivalent to a combination of other terms. For example, one could define a term MAN as exactly equivalent to the conjunction of a PERSON who was also a MALE. Thus, if John were later stated to be a PERSON and a MALE, a system could *deduce* that JOHN was a MAN instead of waiting to be told that JOHN were a MAN.

A second later addition was the introduction of concrete domains. It is important in any web language to be able to make reference to ground objects such as the XML type system in order to appropriately encode things such as numbers and strings. The DAML+OIL extension adds XML types to the language.

5. An Axiomatic Semantics of DAML-ONT

5.1. Overview

In order to fully specify a knowledge representation language, one needs to describe both the syntax and the semantics of the language. The syntax description specifies what strings of characters are legal statements in the language, and the semantic description specifies the intended meaning of each legal statement in the language. The semantics of a representation language can be formally specified in multiple ways. We have chosen here to use the method of specifying a translation of the DAML-ONT language into another representation language that has a formally specified semantics. In particular, we have specified a translation of DAML-ONT into first-order predicate calculus, which has a well-accepted model theoretic semantics. We specify how to translate a DAML-ONT ontology into a logical theory expressed in first-order predicate calculus that is claimed to be logically equivalent to the intended meaning of that DAML-ONT ontology.

There is an additional benefit to this approach. By translating a DAML-ONT ontology into a logically equivalent first-order predicate calculus theory, we produce a representation of the ontology from which inferences can automatically be made using traditional automatic theorem provers and problem solvers. For example, the DAML-ONT axioms enable a reasoner to infer from the two statements "Class Male and class Female are disjoint" and "John is type Male" that the statement "John is type Female" is false.

The translation of DAML-ONT to first-order predicate calculus is done by a simple rule for translating an RDF statement into a first-order relational sentence, and by including in the translation a prespecified set of first-order predicate calculus axioms that restrict the allowable interpretations of the properties and classes that are included in DAML-ONT. This creates a set of first-order sentences that include the specific terms in the ontology along with the pre-specified set of axioms restricting the interpretations. The pre-specified set of axioms and the rules for generating the translation of RDF statements into first order sentences are the focus of this presentation since this is the portion that is leveragable across all DAML-ONT (and RDF/S) ontologies. Since DAML-ONT is simply a vocabulary of properties and classes added to RDF and RDF Schema, and RDF Schema is simply a vocabulary of properties and classes added to RDF, all statements in DAML-ONT are RDF statements and a rule for translating RDF statements is sufficient for translating DAML-ONT statements as well.

We now describe the mapping of DAML-ONT into first-order predicate calculus. A logical theory that is logically equivalent to a set of DAML-ONT descriptions is produced as follows:

Translate each RDF statement with property P, subject S, and object O into a first-order predicate calculus sentence of the form *"(PropertyValue P S O)"*.

Add to this translation the axioms that constrain the allowable interpretations of the properties and classes that are included in RDF, RDF Schema, and DAML-ONT.

Note that it is not necessary to specify a translation for every construct in RDF since any set of RDF descriptions can be translated into an equivalent set of RDF statements (as described in the RDF and RDF Schema specification documents). Thus, the one translation rule above suffices to translate all of RDF and therefore all of DAML-ONT as well.

A notable characteristic of this axiomatization is that it is designed to minimize the constraints on the legal interpretations of the DAML-ONT properties and classes in the resulting logical theory. In particular, the axioms do not require classes to be sets or unary relations, nor do they require properties to be sets or binary relations. Such constraints could be added to the resulting logical theory if desired, but they are not needed to express the intended meaning of the DAML-ONT descriptions being translated.

We now present an informal description of the axioms that are added to the translation of each DAML-ONT ontology. Since DAML-ONT is specified as an extension to RDF and RDF Schema, the axiomatization includes axioms describing the properties and classes in both RDF and RDF Schema, as well as those in DAML-ONT itself.

5.2. The Axiom Language

The axioms are written in ANSI Knowledge Interchange Format (KIF) (*http://logic.stanford.edu/kif/kif.html*), which is a proposed ANSI standard. The axioms use standard first-order logic constructs plus KIF-specific relations and functions dealing with lists and integers. Lists and integers as objects in the domain of discourse are needed in order to axiomatize RDF containers and the DAML-ONT properties dealing with cardinality.

As stated above, each RDF statement "Property P of resource R has value V" is translated into the KIF sentence "(PropertyValue P R V)". Because of the centrality of the "type" property in RDF, we define for convenience an additional binary relation call "Type" to provide a more succinct translation of RDF statements of the form "Property ‘type’ of resource R has value V". The meaning of the relation "Type" is specified by the following axiom:

Ax1. (<=>[8] (Type ?r ?v) (PropertyValue type ?r ?v))

That is, saying that relation "Type" holds for objects R and V is logically equivalent to saying that relation "PropertyValue" holds for objects "type", R, and V.

The axiomatization also restricts the first argument of relation "Type" to be a resource and the second argument to be a class as follows:

Ax2. (=>[9] (Type ?r ?c) (and (Type ?r Resource) (Type ?c Class)))[10]

5.3. Axioms for RDF

RDF is a language for;

- declaring named resources to have type "Property" or "Class",
- declaring resources to have a given class as a type (e.g., "Clyde" is type "Elephant"), and
- stating that a given property of a given resource has a given value (e.g., that property "Color" of "Clyde" has value "Gray").

A property named "type" is used for declaring that the type of a resource R is T so that such a declaration is actually a statement that a given property of a given resource has a given value, specifically that property "type" of resource R has value T. Thus, an RDF file can be considered to consist *entirely* of statements of the form "Property P of resource R has value V".

Our axiomatization provides axioms restricting the interpretation of the classes and properties that are included in RDF. Those classes are "Resource", "Property", "Class", "Literal", "Statement", "Container", "Bag", "Seq", "Alt", and "ContainerMembershipProperty". Those properties are "type", "subject", "predicate", "object", "value", "_1", "_2", "_3", …

5.4. .Axioms for RDF Schema

RDF Schema is simply a vocabulary of properties and classes added to RDF. Our axiomatization provides axioms restricting the interpretation of those classes and properties. The classes are "ConstraintResource" and ConstraintProperty". The properties are "subClassOf", "subPropertyOf", "seeAlso", "isDefinedBy", "comment", "label", "range", and "domain".

5.5. Axioms for DAML-ONT

DAML-ONT is simply a vocabulary of properties and classes added to RDF and RDF Schema. We present in the appendix natural language transcriptions of the axioms that restrict the interpretation of those classes and properties.

5.6. Example Translation and Inference

Consider the following DAML-ONT descriptions of class "Person" and of person "Joe":

```
<Class ID = "Person">
     <subClassOf resource = "#Animal" />
     <restrictedBy>
          <Restriction>
               <onProperty resource = "#parent" />
               <toClass resource = "#Person" />
          </Restriction>
</restrictedBy>
</Class>
<Person ID = "Joe">
     <parent resource = "#John" />
</Person>
```

Those descriptions are equivalent to the following set of RDF statements:

```
(type Person Class)
(subClassOf Person Animal)
(type Restriction R)[11]
(restrictedBy Person R)
(onProperty R parent)
(toClass R Person)
(type Joe Person)
(parent Joe John)
```

Those RDF statements are translated by our axiomatic semantics into the following KIF sentences:

```
(Type Person Class)
(PropertyValue subClassOf Person Animal)
(Type R Restriction)
(PropertyValue restrictedBy Person R)
(PropertyValue onProperty R parent)
(PropertyValue toClass R Person)
(Type Joe Person)
(PropertyValue Parent Joe John)
```

Informally, the "toClass" restriction stated in these sentences is that parents of persons are also persons. Therefore, we should be able to infer from these sentences and the DAML-ONT axioms that "John" is type "Person". That inference can be made using the primary axiom (i.e., Ax30) associated with property "toClass".

Axiom Ax30 is as follows:

```
(=> (and (PropertyValue restrictedBy ?c1 ?r)
         (PropertyValue onProperty ?r ?p)
         (PropertyValue toClass ?r ?c2))
    (forall (?i ?v) (=> (and (Type ?i ?c1)
                             (PropertyValue ?p ?i ?v))
                        (Type ?v ?c2))))
```

The axiom says that if object R is a value of "restrictedBy" for object C1, object P is a value of "onProperty" for R, and object C2 is a value of "toClass" for

R, then for all objects I and V, if I is of type C1 and V is a value of P for I, then V is type C2.

In this axiom, if variable ?c1 is bound to "Person", variable ?r is bound to "R", variable ?p is bound to "parent", and variable ?c2 is bound to "Person", then the sentences describing the "toClass" restriction on the parents of persons satisfy the conjunction that is the antecedent of the implication that is Ax30. One can therefore infer the following corresponding instance of the consequence of that implication:

```
(forall (?i ?v) (=> (and (Type ?i Person)
                         (PropertyValue parent ?i ?v))
                    (Type ?v Person)))
```

If variable ?i is bound to "Joe" and variable ?v is bound to "John", then the sentences describing "Joe" satisfy the conjunction that is the antecedent of the above inferred implication. One can therefore infer the following corresponding instance of the consequence of that implication:

```
(Type John Person)
```

That completes a proof that "John" is type "Person".

5.7. DAML-ONT Classes

This section describes the axioms for the classes that are included in DAML-ONT. For each of these classes C, there is an axiom stating that C is type "Class". The full logical specification of the axioms is available in [Fikes-McGuinness, 2000].

Thing

Ax3. Every object is type Thing.

Nothing

Ax4. Every object is not type "Nothing".

List

Ax5. An object of type "List" is also of type "Sequence".

Disjoint

Ax6. Saying that an object is type "Disjoint" is equivalent to saying that the object is type "List", that every item in the list is type "Class", and that the classes in the list are pairwise disjoint.[12]

Empty

Ax7. "Empty" and "Nothing" are the same class.[13]

TransitiveProperty

Ax8. Saying that an object P is type "TransitiveProperty" is equivalent to saying that P is type "Property", and that if object Y is a value of P for object X and object Z is a value of P for Y, then Z is also a value of P for X.

Unique Property

Ax9. Saying that object P is type "UniqueProperty" is equivalent to saying that P is type property, and that if objects Y and Z are both values of P for object X, then Y and Z are the same object.

UnambiguousProperty

Ax10. Saying that an object P is type "UnambiguousProperty" is equivalent to saying that P is type property, and that if object V is a value of P for both objects X and Y, then X and Y are the same object.

Restriction, Qualification, and Ontology

No axioms other than those stating that each of these are type "Class".

5.8. DAML-ONT Properties

This section describes the axioms for the properties that are included in DAML-ONT. For each of these properties P, there is an axiom stating that P is type "Property". The full logical specification of the axioms is available in [Fikes-McGuinness:2000].

disjointWith

Saying that object C2 is a value of property "disjointWith" for object C1 is equivalent to saying that C1 and C2 are each type "Class", that no object is both type C1 and type C2, and that there is at least one object that is type C1 or type C2.

unionOf

Saying that object L is a value of "unionOf" for object C1 is equivalent to saying that C1 is type "Class", L is type "List", every item in list L is type "Class", and an object is type C1 if and only if it is type of one of the items of list L.

disjointUnionOf

Saying that object L is a value of "disjointUnionOf" for object C is equivalent to saying that L is a value of property "unionOf" for C (i.e., that class C is the union of the classes in list L) and that L is type "Disjoint" (i.e., the classes in list L are pairwise disjoint).

intersectionOf

Saying that object L is a value of "intersectionOf" for object C1 is equivalent to saying that C1 is type "Class", L is type "List", all of the items in list L are type "Class", and an object is type C1 if and only if it is type of all of the items of list L.

complementOf

Saying that object C2 is a value of "complementOf" for object C1 is equivalent to saying that C2 is a value of "disjointWith" for C1 (i.e., C1 and C2 are disjoint classes) and all objects are either type C1 or type C2.

oneOf

Saying that object L is a value of "oneOf" for object C is equivalent to saying that C is type "Class", L is type "List", and the objects that are type C are exactly the items in list L.

asClass

Saying that object C is a value of "asClass" for object L is the equivalent of saying that something is of type C if and only if it is a member of the list L (i.e. it is the first of L or a member of the rest of L.)

first

Saying that object X is a value of "first" for object L is equivalent to saying that L is type "List", L has at least one item, and the first item of L is X.

rest

Saying that object R is a value of "rest" for object L is equivalent to saying that L is type "List", R is type "List", L has at least one item, and L has the same items in the same order as list R with one additional object as its first item.

item

Saying that object X is a value of "item" for object L is equivalent to saying that L is type "List" and either X is a value of "first" for L (i.e., X is the first item in list L) or there is an object R that is a value of "rest" for L (i.e., there is a list R that is the rest of list L) and X is a value of "item" for R (i.e., X is an item in the list R).

cardinality

Saying that object N is a value of "cardinality" for object P is equivalent to saying that P is type "Property" and for all objects X, any list containing no repeated items and containing exactly those objects V such that V is a value of P for X is of length N.

maxCardinality

Saying that object N is a value of "maxCardinality" for object P is equivalent to saying that P is type "Property" and for all objects X, the length of any list containing no repeated items and containing exactly those objects V such that V is a value of P for X is equal to or less than N.

minCardinality

Saying that object N is a value of "minCardinality" for object P is equivalent to saying that P is type "Property" and for all objects X, there exists a list of length at least N that contains no repeated items and contains only objects V such that V is a value of P for X.

inverseOf

Saying that object P2 is a value of "inverseOf" for object P1 is equivalent to saying that P1 is type "Property", P2 is type "Property", and that object X2 is a value of P1 for object X1 if and only if X1 is a value of P2 for X2.

restrictedBy

If object R is a value of "restrictedBy" for object C, then R is type "Resource" and C is type "Class".

onProperty

If object P is a value of "onProperty" for object RQ, then P is type "Property" and RQ is either type "Restriction" or type "Qualification".

toValue

If object V is a value of "toValue" for object R, then R is type "Restriction".

If object R is a value of "restrictedBy" for object C, object P is a value of "onProperty" for R, and object V is a value of "toValue" for R, then for all objects I of type C, V is a value of P for I. (I.e., a "toValue" restriction of V on a property P on a class C constrains each object I of type C to have V as a value of property P.)

toClass

If object C is a value of "toClass" for object R, then R is type "Restriction" and C is type "Class".

If object R is a value of "restrictedBy" for object C1, object P is a value of "onProperty" for R, and object C2 is a value of "toClass" for R, then for all objects I and V, if I is of type C1 and V is a value of P for I, then V is type C2. (I.e., a "toValue" restriction of C2 on a property P on a class C1 constrains each P value of each object of type C1 to be type C2.)

qualifiedBy

If object Q is a value of "qualifiedBy" for object C, then Q is type "Qualification" and C is type "Class".

hasValue

If object C is a value of "hasValue" for object Q, then C is type "Class" and Q is type "Qualification".

If object Q is a value of "qualifiedBy" for object C1, object P is a value of "onProperty" for Q, and object C2 is a value of "hasValue" for Q, then for all objects I of type C1, there exists an object V such that V is a value of P for I and V is type C2. (I.e., a "hasValue" restriction of C2 on a property P on a class C1 constrains each object of type C1 to have a value of type C2.)

versionInfo

If object V is a value of "versionInfo" for object O, then O is type "Ontology".

imports

If object O2 is a value of "imports" for object O1, then O1 is type "Ontology" and O2 is type "Ontology".

equivalentTo

Saying that object Y is a value of "equivalentTo" for object X is equivalent to saying that X and Y are the same object.

default

If object V is a value of "default" for object P, then P is type "Property".

Conclusions

The ontology language for a semantic web needs to be able to express common elements such as classes, properties, restrictions, and objects in the domain. We have provided a historical perspective on the evolution of the initial ontology language for the DAML program. We introduced the language through example. We also provided an axiomatization of the language (and in doing so, also provided an axiomatization for the building blocks of the language—RDF). We showed a short proof using the axioms showing how they can be used to make the inferences mentioned in the introduction by example.

We believe that this language is a useful starting point for describing web content. It builds on decades of research in frame-based systems, description logics, and web languages. It thus has the benefit of years of research on languages, complexity, and usability and may be positioned to be the foundation for the next evolution of web access. It also provided the merging point for the eventual DAML ontology language formed by more tightly integrating the OIL language into DAML-ONT.

Acknowledgements

DAML-ONT is the result work from many people. The US Defense Advanced Research Projects Agency funds the DAML project. The ontology language originated within the program but has been heavily influenced by work beyond DARPA, in particular by work from the European Union funded Semantic Web projects. This paper is authored by two of the three original editors of the language release, the DAML program director, and the co-author of the axiomatic semantics. We would explicitly like to acknowledge the impact of the other original co-editor—Dan Connolly—for his significant impact on

the initial language release. The language has been influenced heavily by others including Tim Berners-Lee, Frank van Harmelen, Ian Horrocks, Dan Brickley, Mike Dean, Stefan Decker, Pat Hayes, Jeff Heflin, Drew McDermott, Peter Patel-Schneider, and Ralph R. Swick. The evolving list of major contributors is maintained on *www.daml.org*. DAML-ONT -has since merged into a new ontology language called DAML+OIL (see *http://www.daml.org/2001/03/daml+oil-index* for the release). We will be producing an updated paper reflecting the larger language. Also, DAML is expected to include a logic language (DAML-L) and a service language (DAML-S).

References

[Bechhofer et al., 2000] S. Bechhofer et al., "An Informal Description of OIL-Core and Standard OIL: A Layered Proposal for DAML-O, www.ontoknowledge.org/oil/downl/dialects.pdf

[Berners-Lee] Tim Berners-Lee. Semantic Web Road Map. 1998. http://www.w3.org/DesignIssues/Semantic.html

[Berners-Lee et al., 2000]. Tim Berners-Lee, David R. Karger , Lynn Andrea Stein , Ralph R. Swick, and Daniel J. Weitzner. Semantic Web Development. 2000. http://www.w3.org/2000/01/sw/DevelopmentProposal

[Borgida et. al, 1989] Alex Borgida, Ronald J. Brachman, Deborah L. McGuinness, and Lori Alperin Resnick. ``CLASSIC: A Structural Data Model for Objects", Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, Portland, Oregon, June, 1989, pp. 59-67.

[Brachman-Schmolze, 1985] Ronald J. Brachman, James G. Schmolze, `An Overview of the KL-ONE Knowledge Representation System', Cognitive Science, Vol 9(2), pp 171-216, 1985.

[Brickley-Guha, 2000] D. Brickley and R.V. Guha, Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation, World Wide Web Consortium, 2000, www.w3.org/TR/rdf-schema.

[Chaudhri et. al. 1998] Vinay Chaudhri, Adam Farquhar, Richard Fikes, Peter Karp, and James Rice; "OKBC: A Programmatic Foundation for Knowledge Base Interoperability", AAAI 1998.

[Farquhar et. al, 1997] Adam Farquhar, Richard Fikes, and James Rice; "The Ontolingua Server: a Tool for Collaborative Ontology Construction", Intl. Journal of Human-Computer Studies **46**, 1997.

[Fikes & Kehler 1985] Richard Fikes & Tom Kehler, "The Role of Frame-Based Representation in Reasoning", CACM 28(9): 904-920 (1985).

[Karp 1992] Peter D. Karp, "The design space of frame knowledge representation systems", Technical Report 520, SRI International AI Center; available on line as ftp://www.ai.sri.com/pub/papers/karp-freview.ps.Z

[Donini et al., 1991a] F. Donini, M. Lenzerini, D. Nardi, and W. Nutt. `The Complexity of Concept Languages', KR-91, pp 151-162, 1991.

[Donini et al., 1991b] F. Donini, M. Lenzerini, D. Nardi, and W. Nutt. Tractable concept languages', IJCAI-91, pp 458-465, 1991.

[Fikes-McGuinness, 2000] Richard Fikes and Deborah L. McGuinness. "An Axiomatic Semantics for DAML-ONT," December 10, 2000. www.ksl.stanford.edu/people/dlm/daml-semantics .

[Fensel et al., 2000] Dieter Fensel, Ian Horrocks, Frank van Harmelen, Stefan Decker, M. Erdmann, and M. Klein. OIL in a nutshell. In Proceedings of the European Knowledge Acquisition Conference (EKAW-2000), Lecture Notes In Artificial Intelligence. Springer-Verlag, 2000.

[Fikes & McGuinness] Richard Fikes & Deborah L. McGuinness, "An Axiomatic Semantics for RDF, RDF Schema, and DAML+OIL", KSL Technical Report KSL-01-01, Stanford University, 2001; available on-line as http://www.ksl.stanford.edu/people/dlm/daml-semantics/abstract-axiomatic-semantics.html.

[Heflin & Hendler] J. Heflin and J. Hendler, "Semantic Interoperability on the Web," Proc. Extreme Markup Languages 2000, Graphic Communications Assoc., Montreal, Canada 2000, pp. 111–120.

[Horrocks and Patel-Schneider, 1999] Horrocks, I., Patel-Schneider, P., `Optimizing description logic subsumption', Journal of Logic and Computation, Vol 9(3), pp 267-293, 1999.

[Horrocks—et-al, 1999] I. Horrocks and U. Sattler: A description logic with transitive and inverse roles and role hierarchies. Journal of Logic and Computation, 9(3):385-410, 1999.

[Horrocks, Sattler & Tobies] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for very expressive description logics. Logic Journal of the IGPL, 8(3):239-263, 1999.

[Lassila, 1998] O. Lassila, "Web Metadata: A Matter of Semantics," IEEE Internet Computing, vol. 2, no. 4, July/Aug. 1998, pp. 30–37.

[Lassila-Swick,1999] O. Lassila and R. Swick, Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation, World Wide Web Consortium, 1999, www.w3.org/TR/REC-rdf-syntax .

[MacGregor, 1991] Robert M. MacGregor: Inside the LOOM Description Classifier. SIGART Bulletin 2(3): 88-92 (1991)

[Stein & Connolly, 2000] Lynn Andrea Stein and Dan Connolly. Annotated DAML Ontology Markup. 2000. www.daml.org/2000/10/daml-walkthru

Footnotes

[1]Gates Building 2A Stanford University Stanford, CA 94305 USA 650-723-9770 dlm@ksl.stanford.edu

[2]Gates Building 2A Stanford University Stanford, CA 94305 USA 650-723-9770 fikes@ksl.stanford.edu

[3]Franklin W. Olin College of Engineering 1735 Great Plain Avenue Needham, MA 02495 USA 781-292-2525 las@olin.edu

[4]DARPA/ISO 3701 N. Fairfax Dr.Arlington, VA 22203 703-696-2238 jhendler@darpa.mil

[5]SHIQ includes ALC plus transitive roles, inverse roles, and qualified number restrictions. For more on the theoretical analysis and reasoning issues see [Horrocks and Sattler, 1999] and [Horrocks et. al, 1999].

[6]The namespace prefix rdf is arbitrary. The name in this document would be the conventional choice.

[7]Unprefixed attribute names are not associated with the default namespace name the way unprefixed element names are. This is expanded upon in myth 4 available in "Namespace Myths exploded" at: http://www.xml.com/pub/a/2000/03/08/namespaces/index.html?page=2 .

[8]KIF note: "<=>" means "if and only if". Relational sentences in KIF have the form " (<relation name> <argument>*)". Names whose first character is ``?" are variables. If no explicit quantifier is specified, variables are assumed to be universally quantified.

[9]KIF note: "=>" means "implies".

[10]RDF considers "Resource" to be a class and "Class" itself to be a class. The axiomatization specifies that semantics by including the axioms "(Type Resource Class)" and "(Type Class Class)".

[11]The restriction is unnamed in the RDF markup. We use the arbitrary name "R" here to denote the restriction.

[12]The DAML-ONT property "disjointWith" is used to express pairwise disjointness.

[13]The DAML-ONT property "asClass" is used to express equivalence of classes.

4. Ontologies and Schema Languages on the Web

Michel Klein[1], Jeen Broekstra[2], Dieter Fensel[1], Frank van Harmelen[1,2], and Ian Horrocks[3]

1. Introduction

For the past few years, information on the World Wide Web was mainly intended for direct human consumption. However, to facilitate new intelligent applications such as meaning-based search, information brokering and electronic transactions between several independent partners, the semantics of the data on the internet should also be accessible for machines. Therefore, methods and tools to create such a "semantic web" have generated wide interest.

An major aspect of a web of "machine understandable" information are explicit models of the domains of interest, which describe the vocabulary and structure of the information. These models - often called ontologies - may play a key role in advanced information exchange, as they provide a shared and common understanding of a domain. However, it is still an important question how ontologies can be applied fruitfully to online resources.

In this chapter, we will look at the relation between ontologies and two schema languages from the World Wide Web consortium, that – in some sense – both aims at providing a mechanism for describing the common understanding of data in a specific domain. One of them is RDF Schema [Brickley and Guha, 2000], a schema language for the Resource Description Framework (RDF)[Lassila & Swick, 1999]. RDF is a standard from the W3C for representing metadata on the web. RDF Schema provides some modelling primitives that can be used to define a simple model of classes and their relations. This model can then act as a vocabulary for RDF statements, which describe resources on the web. The other schema language that we will consider is XML Schema [Thompson et al., 2000]. This is a proposed standard for describing the structure and semantics of XML documents. It prescribes the way in which elements and attributes in an XML document are combined and can be used to validate the structure and content of a document.

The aim of this chapter is to investigate how these languages can be used to add ontology-based semantics to online resources. We will descibe their different roles in this process, and illustrate the way in which they can play that role. In short, we will argue that XML Schema documents and ontologies refer to different abstraction levels on how to describe information and therefore also to different states in the process of developing on-line information sources. To illustrate this, we will provide a transformation procedure from ontologies to XML Schema documents. RDF Schema documents and ontologies, on the other hand, serve the same purpose to a large extent, but generally differ in the level in the level of expressiveness. We will show a method to capture most of the expressive power of ontologies in RDF Schema documents, by extending the RDF Schema language.

This chapter is organized as follows. In the next section, we will give an abstract introduction to ontologies, schemas and their relationship. In Section 3 we provide a short introduction to a specific ontology language, called OIL [Fensel et al., 2000], that we will use as an example language throughout this chapter. Section 4 introduces XML Schema and compares it to ontology languages. Section 5 does the same for RDF Schema. In Section Conclusions, we give two examples of the application of ontologies to online resources, using respectivily XML Schema and RDF Schema.

2. Ontologies and Schemas

Ontology, which has been a field of philosophy since Aristotle, has become a buzz-word in information and knowledge-based systems research [Guarino & Welty, 2000]. Various publications in knowledge engineering, natural language processing, cooperative information systems, intelligent information integration, and knowledge management report about the application of ontologies in developing and using systems. In general, ontologies provide a *shared and common* understanding of a domain that can be communicated between people and heterogeneous and distributed application systems. They have been developed in Artificial Intelligence to facilitate knowledge sharing and reuse.

Schemas, in particular for databases, have been developed in computer science to describe the structure and semantics of data. A well-known example is the relational database schema that has become the basis for most of the currently used databases [Elmasri & Navathe, 2000]. A database schema defines a set of relations and certain integrity constraints. A central assumption is the atomicity of the elements that are in certain relationships (i.e., first normal form). In a nutshell, an information source (or, more precisely, a data source) is viewed as a set of tables. However, many new information sources now exist that do not fit into such rigid schemas. In particular, the WWW has made predominantly document-centered information based on natural language text available. Therefore, new schema languages have arisen that better fit the needs of richer data models. Some of them, like XML Schema (see [Biron & Malhotra, 2000], [Thompson et al., 2000] and [Walsh, 1999]), integrate schemas for describing documents (like HTML or SGML) with schemas designed for describing data. Other schema languages, such as RDF Schema, abstract from structure and representation issues and try to provide a general basis for data modelling on the web.

And their relationship? Ontologies applied to on-line information source may be seen as explicit conceptualizations (i.e., *meta information*) that describe the semantics of the data. Fensel [Fensel, 2001] points out the following common differences between ontologies and schema definitions:

- A language for defining ontologies is often syntactically and semantically richer than common approaches for databases.
- The information that is described by an ontology consists of semi-structured natural language texts and not tabular information.
- An ontology must be a shared and consensual terminology because it is used for information sharing and exchange.
- An ontology provides a domain theory and not the structure of a data container, like most schema languages do.

However, these statements need to be formulated more precisely when comparing ontology languages with both the XML Schema language as the RDF Schema language, and the purpose of ontologies with the purpose of those schema languages. This will be done in the next sections.

3. The Ontology Language OIL

To investigate the relations between ontology languages and schema languages, we will use one specific ontology language as reference and example. Horrocks et al. [Horrocks et al., 2000] defines the *Ontology Interface Layer (OIL)*. In this section we will only give a brief description of the OIL language. More detailed descriptions can be found elsewhere; a comparison of OIL with other ontology languages can be found in [Horrocks et al., 2000] and [Fensel et al., 2000].

A brief example ontology in OIL is provided in Figure 4.1; the example is based on the country pages of the CIA World Factbook[4], which we will use as an example throughout this chapter. The OIL language has been designed so that: (1) it provides most of the modeling primitives commonly used in frame-based and Description Logic (DL) oriented ontologies; (2) it has a simple, clean, and well defined semantics; (3) automated reasoning support, (e.g., class consistency and subsumption checking) can be provided. It is envisaged that this core language will be extended in the future by sets of additional primitives, with the proviso that full reasoning support may not be available for ontologies using such primitives.

An ontology in OIL is represented by an *ontology container* and an *ontology definition*. We will discuss both elements of an ontology specification in OIL. We start with the ontology container and will then discuss the backbone of OIL, the ontology definition.

For the *ontology container* part, OIL adopts the components defined by the Dublin Core Metadata Element Set, Version 1.1. [5]

Apart from the container, an OIL ontology consists of a set of *ontology definitions*:

- *import* A list of references to other OIL modules that are to be included in this ontology. Specifications can be included and the underlying assumptions is that names of different specifications are different (via different prefixes).

- *class and slot definitions* Zero or more class definitions (*class-def*) and slot definitions (*slot-def*), the structure of which will be described below.

```
 Ontology-container
 title CIA World Fact Book ontology
 creator Michel Klein
 subject country information, CIA, world
factbook
 description  A didactic example ontology
describing
       country information
  description.release 1.02
 publisher CIA
 type ontology
 format pseudo-xml
 identifier http://www.ontoknowledge.org/oil/wfb.xml
 source
http://www.odci.gov/cia/publications/factbo
ok/
 language OIL
 language en-uk

 Ontology-definitions
 slot-def capital
  domain Country
  range City
  inverse capital_of
  properties functional
 Slot-def has_boundary
  domain Country
  range LandBoundary
 Slot-def coastline
  domain Geographical_Location
  range (KilometerLength or MilesLength)
 Slot-def relative_area
  domain Geographical_Location
  range AreaComparison
 Slot-def value
  domain (KilometerLength or
MilesLength)
  range integer
  properties functional
```

```
 class-def
Geographical_Location
  slot-constraint name
    value-type string
 class-def City
  subclass-of
Geographical_Location
  slot-constraint located_in
    value-type Country
 class-def Country
  subclass-of
Geographical_Location
  slot-constraint capital
    has-value City
 class-def LandBoundary
  slot-constraint
neighbor_country
    cardinality 1 Country
  slot-constraint length
    value-type
    (KilometerLength or
MilesLength)
 class-def KilometerLength
  slot-constraint value
    has-value integer
  slot-constraint unit
    has-value km
 class-def MilesLength
  slot-constraint value
    has-value integer
  slot-constraint unit
    has-value mile
 class-def AreaComparison
  slot-constraint
compared_to
    value-type
Geographical_Location
  slot-constraint proportion
    value-type string
```

Figure 4.1 An partial ontology in OIL

A class definition associates a class name with a class description. A *class-def* consists of the following components:

- *type* The type of definition. This can be either *primitive* or *defined*; if omitted, the type defaults to primitive. When a class is primitive, its definition (i.e., the combination of the following subclass-of and slot-constraint components) is taken to be a necessary but not sufficient condition for membership of the class. When a class is defined, its definition is taken to be a necessary *and* sufficient condition for membership of a class.
- *subclass-of* A list of one or more class-expressions, the structure of which will be described below. The class being defined in this class-def must be a subclass of each of the class expressions in the list.
- *slot-constraints* Zero or more slot-constraints, the structure of which will be described below. The class being defined in this class-def must be a subclass of each of the slot-constraints in the list (note that a slot-constraint defines a class).

A *class-expression* can be either a class name, a *slot-constrain*t, or a boolean combination of class expressions using the operators *and*, *or,* or *not*. Note that class expressions are recursively defined, so that arbitrarily complex expressions can be formed.

In some situations it is possible to use a *concrete-type-expression* instead of a class expression. A concrete-type-expression defines a range over some data type. Two data types that are currently supported in OIL are *integer* and *string*.

Ranges can be defined using the expressions *(min* X), *(max* X), *(greater-than* X), *(less-than* X), *(equal* X) and *(range* X Y). For example, *(min* 21) defines the data type consisting of all the integers greater than or equal to 21. As another example, *(equal* "xyz") defines the data-type consisting of the string "xyz".

A *slot-constraint* is a list of one or more constraints (restrictions) applied to a *slot*. A slot is a binary relation (i.e., its instances are pairs of individuals), but a slot-constraint is actually a class definition—its instances are those individuals that satisfy the constraint(s). Typical slot-constraint are:

- *has-value* A list of one or more *class-expressions*. Every instance of the class defined by the slot constraint must be related via the slot relation to an instance of each *class-expression* in the list. For example, the *has-value* constraint:

  slot-constraint eats

  has-value zebra, wildebeest

defines the class each instance of which *eats* some instance of the class zebra and some instance of the class wildebeest. Note that this does not mean that instances of the slot-constraint eat *only* zebra and wildebeest: they may also be partial to a little gazelle when they can get it.

- *value-type* A list of one or more class-expressions. If an instance of the class defined by the slot-constraint is related via the slot relation to some individual *x*, then *x* must be an instance of each class-expression in the list.

- *max-cardinality* A non-negative integer n followed by a class-expression. An instance of the class defined by the slot-constraint can be related to at most n distinct instances of the class-expression via the slot relation.
- *min-cardinality* and, as a shortcut, *cardinality*.

A slot definition *(slot-def*) associates a slot name with a slot description. A slot description specifies global constraints that apply to the slot relation, for example that it is a transitive relation. A slot-def consists of the following main components:

- *subslot-of* A list of one or more slots. The slot being defined in this slot-def must be a subslot of each of the slots in the list. For example,

  slot-def daughter

    subslot-of child

defines a slot *daughter* that is a subslot of child, i.e., every pair of individuals that is an instance of *daughter* must also be an instance of child.

- *domain* A list of one or more class-expressions. If the pair (x; y) is an instance of the slot relation, then x must be an instance of each class-expression in the list.

- *range* A list of one or more class-expressions. If the pair (x; y) is an instance of the slot relation, then y must be an instance of each class-expression in the list.
- *inverse* The name of a slot *S* that is the inverse of the slot being defined. If the pair (x; y) is an instance of the slot *S*, then (y; x) must be an instance of the slot being defined.
- *properties* A list of one or more properties of the slot.

Valid properties are: *transitive*, *functional* and *symmetric*.

An *axiom* asserts some additional facts about the classes in the ontlogy, for example that the classes carnivore and herbivore are disjoint (that is, have no instances in common). Valid axioms are:

- *disjoint (class-expr)+* All of the class expressions in the list are pairwise disjoint.
- *covered (class-expr) by (class-expr)+* Every instance of the first class expression is also an instance of at least one of the class expressions in the list.
- *disjoint-covered (class-expr) by (class-expr)+* Every instance of the first class expression is also an instance of exactly one of the class expressions in the list.
- *equivalent (class-expr)+* All of the class expressions in the list are equivalent (i.e. they have the same instances).

Besides the "presentation syntax" described above, OIL also has a XML and RDF representation. The technical report on OIL [Horrocks et al., 2000] defines a DTD and an XML Schema definition for the XML syntax. The representation of OIL in RDF is described in Section 2, and also in [Broekstra et al., 2000]. We will now take a look at XML Schema its relation to OIL.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
<xsd:complexType name="address">
<xsd:sequence>
<xsd:element name="name">
<xsd:complexType mixed="true">
<xsd:simpleContent>
<xsd:extension base="xsd:string"/>
</xsd:simpleContent>
</xsd:complexType>
</xsd:element>
<xsd:element name="street" type="xsd:string" maxOccurs="2"/>
<xsd:element ref="zip"/>
<xsd:element name="city" type="xsd:string"/>
<xsd:element name="country" type="xsd:string" minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>
<xsd:element name="zip" type="zipCode"/>
<xsd:simpleType name="zipCode">
<xsd:restriction base="xsd:string">
<xsd:pattern value="[0-9]{5}(-[0-9]{4})?"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:schema>
```

Figure 4.2 An example for a schema definition.

4. XML Schema

4.1. Description of XML Schema

XML Schema is a means for defining constraints on the syntax and structure of valid XML documents (cf. [Biron &

Malhotra, 2000], [Thompson et al., 2000], [Walsh, 1999]). A more easily readable explanation of XML Schema can be found in [Fallside, 2000]. XML Schemas have the same purpose as DTDs, but provide several significant improvements:

- XML Schema definitions are themselves XML documents.
- XML Schemas provide a rich set of datatypes that can be used to define the values of elementary tags.
- XML Schemas provide a much richer means for defining nested tags (i.e., tags with subtags).
- XML Schemas provide the namespace mechanism to combine XML documents with heterogeneous vocabulary.

We will discuss these four aspects in more detail.

4.1.1. XML schema definitions are themselves XML documents

Figure 4.2 shows an XML Schema definition of an address. The schema definition for the address tag is itself an XML document, whereas DTDs would provide such a definition in an external second language. The clear advantage is that all tools developed for XML (e.g., validation or rendering tools) can be immediately applied to XML schema definitions, too.

4.1.2. Datatypes

Datatypes are described in [Biron & Malhotra, 2000]. We already saw the use of a datatype (i.e., *string*) in the example. In general, a datatype is defined as a 3-tuple consisting of a set of distinct values, called its *value space*, a set of lexical representations, called its *lexical* *space*, and a set of *facets* that characterize properties of the value space, individual values, or lexical items.

Value space. The value space of a given datatype can be defined in one of the following ways: enumerated outright (extensional definition), defined axiomatically from fundamental notions (intensional definition)[6], defined as the subset of values from a previously defined datatype with a given set of properties, and defined as a combination of values from some already defined value space(s) by a specific construction procedure (e.g., a list).

Lexical space. A lexical space is a set of valid literals for a datatype. Each value in the datatype's value space is denoted by one or more literals in its lexical space. For example, "100" and "1.0E2" are two different literals from the lexical space of float which both denote the same value.

Facets. A facet is a single defining aspect of a datatype. Facets are of two types: fundamental facets that define the datatype and non-fundamental or constraining facets that constrain the permitted values of a datatype.

- Fundamental facets: equality, order on values, lower and upper bounds for values, cardinality (can be categorized as "finite", "countably infinite" or "uncountably infinite"), numeric versus nonnumeric
- Constraining or non-fundamental facets are optional properties that can be applied to a datatype to constrain its value space: length constrains minimum and maximum,

pattern can be used to constrain the allowable values using regular expressions, enumeration constrains the value space of the datatype to the specified list, lower and upper bounds for values, precision, encoding, etc. Some of these facets already constrain the possible lexical space for a datatype.

It is useful to categorize the datatypes defined in this specification along various dimensions, forming a set of characterization dichotomies.

- *Atomic vs. list datatypes*: Atomic datatypes are those having values which are intrinsically indivisible. List datatypes are those having values which consist of a sequence of values of an atomic datatype. For example, a single token which matches NMTOKEN from the XML 1.0 [Bray, 1998] could be the value of an atomic datatype NMTOKEN, whereas a sequence of such tokens could be the value of a list datatype NMTOKENS.
- *Primitive vs. generated datatypes*: Primitive datatypes are those that are not defined in terms of other datatypes; they exist ab initio. Generated datatypes are those that are defined in terms of other datatypes. Every generated datatype is defined in terms of an existing datatype, referred to as the basetype. Basetypes may be either primitive or generated. If type a is the basetype of type b, then b is said to be a subtype of a. The value space of a subtype is a subset of the value space of the basetype. For

example, date is derived from the base type *recurringInstant*.

- *Built-in vs. user-derived datatypes*: Built-in datatypes are those which are defined in the XML schema specification and may be either primitive or generated. User-derived datatypes are those derived datatypes that are defined by individual schema designers by giving values to constraining facets. XML Schema provides a large collection of such built-in datatypes, for example, string, boolean, flot, decimal, timeInstant, binary, etc. In our example, zipCode is an user-derived datatype.

4.1.3. Structures

Structures provide facilities for constraining the contents of elements and the values of attributes and for augmenting the information set of instances, e.g. with defaulted values and type information (see [Thompson et al., 2000]). They make use of the datatypes for this purpose. An example is the element zip that makes use of the datatype zipCode. Another example is the definition of the element type "name". The value "true" for the "mixed" attribute of the complexType allows to mix strings with (sub-)tags.

Attributes are defined by their *name*, a datatype that constraints their values, default or fixed values, and constraints on their presence (minOccurs and maxOccurs), see for example: <attribute name="key" type="integer" minOccurs="1" maxOccurs="1"/>

Elements can be constrained by reference to a simple datatype. The datatypes can be unconstrained, can be constrained to be empty, or can allow elements in its content (called rich content model).

- In the former case, *element declarations* associate an element name with a type, either by reference (e.g. *zip* in Figure 4.2) or by incorporation (i.e., by defining the datatype within the element declaration).
- In the latter case, the content model consists of a *simple grammar governing the allowed types of child elements and the order in which they must appear*. If the mixed qualifier is present, text or elements may occur. Child elements are defined via an *element reference* (e.g. <element ref="zip"/>) or directly via an *element declaration*. Elements can be combined in *group*s with a specific *order* (*all*, *sequence* or *choice)*. This combination can be recursive, for example, a sequence of some elements can be a selection from a different sequence or a sequence of different elements (i.e., the "()", "," and "| "of a DTD are present). Elements and their groups can be accompanied with *occurrence constraints*, for example, *<element name="street" minOccurs="1" maxOccurs="2" type="string"/>.*

In the previous subsection we already discussed the differences between primitive and generated datatypes, where the latter is defined in terms of other datatypes (see [Biron & Malhotra, 2000]). This is not only possible for simple datatypes like integer, but also for complex types. There are two mechanisms for derived type definitions defined in [Thompson et al., 2000].

```
 <xsd:complexType name="personName">
 <xsd:sequence>
  <xsd:element name="title" minOccurs="0"/>
  <xsd:element name="forename" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element name="surname"/>
 </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="extendedName">
 <xsd:complexContent>
  <xsd:extension base="personName">
    <xsd:sequence>
     <xsd:element name="generation" minOccurs="0"/>
    </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
 </xsd:complexType>
 <xsd:element name="name" type="extendedName"/>

 A snippet of a valid XML-file according to this schema is:
 <name>
 <forename>Albert</forename>
 <forename>Arnold</forename>
 <surname>Gore</surname>
 <generation>Jr</generation>
</name>
```

Figure 4.3 An example for a derived type definitions via extension.

Here the following two cases are distinguished:

- Derivation by *extension*. A new complex type can be defined by adding additional particles at the end of its definition and/or by adding attribute declarations. An example for such an extension is provided in Figure 4.3.

- Derivation by *restriction*. A new type can be defined by decreasing the possibilities made available by an existing type definition: narrowing ranges, removing alternatives, etc.

4.1.4. Namespaces

The facilities in XML Schema to construct schemas from other ones builds on XML namespaces. An important concept in XML Schema is the *target namespace*, which defines the URL that can be used to uniquely identify the definitions in the schema. XML Schema provides two mechanism for assembling a complete component set from separate original schemas (cf. [Thompson et al., 2000]):

- The first is via the *include* element (<include schemaLocation="http..."/>). The effect of this include element is to bring in the definitions and declarations contained in the refered schema and make them available as part of the including schema target namespace. The effect is to compose a final effective schema by merging the declarations and definitions of the including and the included schemas. The one important caveat is that the target namespace of the included components must be the same as the target namespace of the including schema.

The *redefine* mechanism is very much the same as the include mechanism, but also allows to change the included types.

- Second, the *import* element (<import namespace="http..."/>) can be used to import schemas with a different target namespace. It should coincide with a standard namespace declaration. XML Schema in fact permits multiple schema components to be imported, from multiple namespaces, and they can be referred to in both definitions and declarations.

In general, only inclusion is provided as means to combine various schemas and module name prefix is used to realize the non-equality of name assumptions (i.e., identifiers of two different schemas are by definition different).

4.2. The Relation Between OIL and XML Schema

On the one hand, ontologies and XML schemas serve very different purposes. Ontology languages are a means to specify domain theories and XML schemas are a means to provide integrity constraints for information sources (i.e., documents and/or semistructured data). It is therefore not surprising to encounter differences when comparing XML schema with ontology languages like OIL. On the other hand, XML Schema and OIL have one main goal in common: both provide vocabulary and structure for describing information sources that are aimed at exchange. It is therefore legitimate to compare both and investigate their commonalities and differences. In this section, we provide a twofold way to deal with this situation. First we analyze the main differences and second we try to characterize their relation.

4.2.1. Comparing OIL and XML Schema

Modelling primitives. XML Schema’s main modelling primitives are elements. Elements may be simple, composed or mixed. Simple elements have as their contents datatypes, like string or integer. Composed elements have as contents other (child) elements. Also they define a grammar that defines how they are composed from their child elements. Finally, mixed elements can mix strings with child elements. In addition, elements may have attributes. OIL takes a different point of view. The basic modeling primitives are concepts and slots. Concepts can be roughly identified with elements and child elements are roughly equivalent to slots defined for a concept. However, slots defined independently from concepts have no equivalents in XML Schema.

Datatypes. XML Schema provides a large collection of built-in datatypes as, for example, string, boolean, float, decimal, timeInstant, binary, etc. OIL only provides *string* and *integer* as built-in datatypes, because of the difficulty of providing clear semantics and reasoning support for a large collection of complex datatypes.

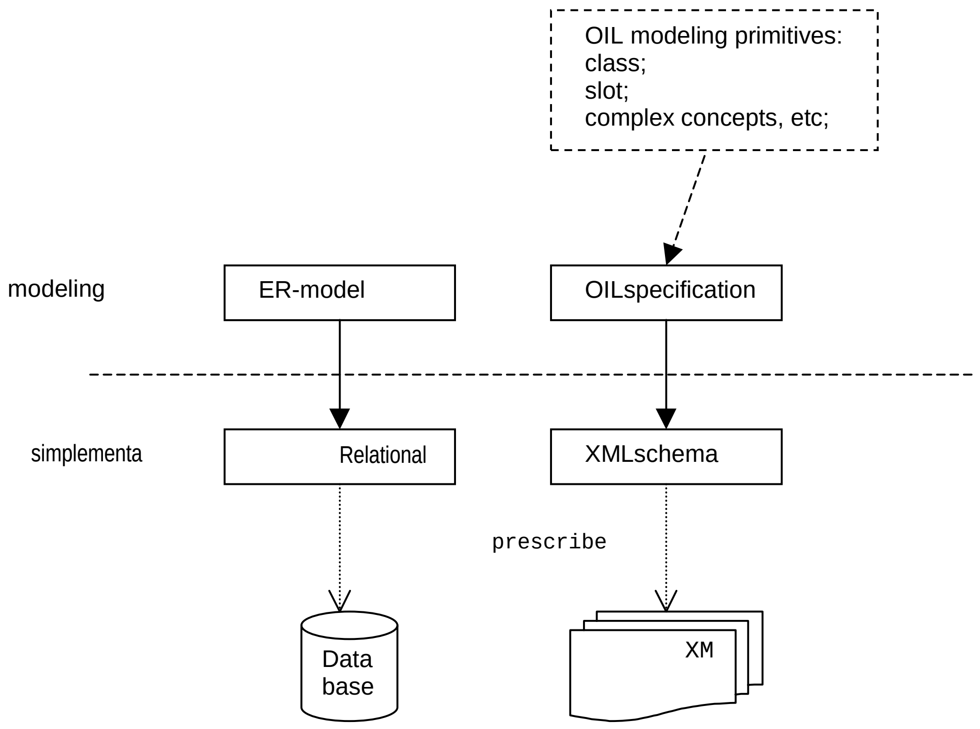


Figure 4.4 The relationship between schemas and ontologies in a nutshell.

In XML Schema all inheritance must be defined explicitly, so reasoning about hierarchical relationships is not an issue. In XML Schema, a datatype is defined by a value space, a lexical space, and a set of facets. Restricting a value space (i.e., the membership of classes) is also present in OIL, however, OIL does not provide a lexical space and facets. These aspects are much more related to the representation of a datatype than to the aspect of modeling a domain. That is, *date* may be an important aspect of a domain, but various different representations of *dates* are not. This is a rather important aspect when talking about how to represent the information. Finally, although XML Schema mentions the possibility of defining types intensionally via axioms, no language, semantics, nor any actual reasoning service is provided for this purpose. OIL is a flexible language for the intensional, i.e. axiomatic, definition of types. It provides facilities to for the intensional definition of types (via *defined concepts*) that is completely lacking in XML Schema [7]

Grammar. OIL does not provide any grammar for the application of slots to concepts, i.e., an instance of a concept comprises of a *set* of slots values. XML Schema allows the definition of stronger requirements via a grammar: *sequence* and *choice* of attributes applied to an instance can be defined.

Inheritance. XML Schema incorporates the notion of type-derivation. However, this can only partially be compared with what is provided with inheritance in ontology languages. First, in XML Schema all inheritance has to be modeled explicitly. In OIL inheritance can be derived from the definitions of the concepts. Second, XML Schema does *not* provide a direct way to inherit from multiple parents. Types can only be derived from one basetype. OIL (like most ontology languages) provides multiple inheritance. Third, and very important, the is-a relationship has a twofold role in conceptual modeling which is not directly covered by XML *Schema*:

- Top-down inheritance of attributes from superclasses to subclasses. Assume *employee* as a subclass of a class *person*.

Then *employee* inherits all attributes that are defined for *person*.

- Bottom-up inheritance of instances from subclasses to superclasses. Assume *employee* as a subclass of a class *person*. Then *person* inherits all instances (i.e., elements) that are an element of *employee*.

In XML Schema, both aspects can only be modeled in an artificial way. The top-down inheritance of attributes is difficult to model, because type derivations in XML Schema can either extend *or* restrict the base type. A "dummy" intermediate type has to be used to model full top-down inheritance of attributes with both extending and restricting derivations. For example, it is not possible to model a student as a *person* with a student-number and age < 28 in only one step. You first have to model a dummy type "young person", which *restricts* the age of persons to less than 28. After that it is possible to model a student as a "young person" *extended* with a student-number.

Also the bottom-up inheritance of instances to superclasses is not automatically available in XML Schema. For example, an instance of a student is not automatically an valid instance of a person, even if the student-type inherits from the person-type. However, using an additional attribute, it is possible to use an instance of a subclass wherever a superclass of it is expected. So, to use a student is as a filler of a "driver" element, which requires type person, we can write:

```
<driver xsi:type="student">
<name>John</name>
<studentnumber>0792098</studentnumber>
</driver>
```

The *type* attribute - which is part of the XML Schema instance namespace - explicitly declares that the current element is a derivation of the expected element.

4.2.2. Characterization of the Relation Between OIL and XML Schema

On the one hand, OIL provides much richer modeling primitives. It distinguish classes and slots, and class (or slot) definitions can be used to derive the hierarchy (and its according inheritance). On the other hand, XML Schema provides richer modeling primitives concerning the variety of built-in datatypes and the grammar for structuring the content of elements. The latter is not of importance when building a domain model but important when defining the structure of documents. Therefore, models in OIL can be viewed as a high level description of a domain that is further refined when aiming at a document structure model.

The relation between ontology languages and XML Schema can be compared to the relation between the *Entity Relationship model (ER model)* and the relational model [8]. We realize that this analogy is only partially correct, because ER is a model for analysis, whereas OIL is a language for design. Nevertheless, the metaphor illustrates the relation nicely. The Entity Relationship model provides a modeling framework for modeling information sources required for an application, and the relational model provides an implementation oriented description of databases. The former provides entities (with attributes) and relationships and the latter only provides relations. In [Elmasri & Navathe, 2000], Elmasri and Navathe provides a procedure that translates models formulated in the Entity Relationship model into the relation model. During system development you start with a high-level ER model. Then you transform this model into a more implementation oriented relational model. Concepts and relationships are both expressed as relations. A similar procedure can be used when transforming OIL specifications into XML Schema definitions. We will illustrate this procedure in Section 1.

Figure 4.4 shows the overall picture of the relation between ER and the relational model on the one hand and OIL and XML Schema on the other.

The benefit of the procedure that we suggest is that a document schema is created that is founded in a domain ontology. This schema in its turn can be used to prescribe or validate document markup. Finally, this gives us a well-founded semantic annotation of actual data. This has two main advantages:

- it provides a way to represent instance data of an ontology in plain XML;
- an ontology can be used as a conceptual layer on top of a set of structured (XML) documents, because the markup of actual XML documents is founded in an ontology.

We want to stress that this relation that we describe here is completely different from expressing an ontology in some kind of XML format itself, although both approaches may use XML Schema documents. In our case, the XML Schema definition is a "simplified" version of the ontology, whereas in the other case, the XML Schema definition prescribes how an ontology should be encoded in XML. The latter approach can be found in [Horrocks et al., 2000], where an XML Schema definition is provided to write down an ontology in a plain XML document. In this chapter, we propose to produce a schema that captures the underlying semantics of an ontology, which can be used for representing instances of an ontology in XML.

5. RDF Schema

We will now discuss the main features RDF Schema (or RDFS for short) and show how RDF Schema relates to ontology languages. We will first describing RDF itself, which is the basis of RDF Schema, and then RDF Schema.

5.1. RDF and RDF Schema

5.1.1. Introduction to RDF

A prerequisite for the Semantic Web is machine-processable semantics of the information. The Resource Description Framework (RDF) [Lassila & Swick, 1999] is a foundation for processing metadata; it provides in-teroperability between applications that exchange machine-understandable information on the Web. Basically, RDF defines a data model for describing machine processable semantics of data. The basic data model consists of three object types:

Resources: A resource may be an entire Web page; a part of a Web page; a whole collection of pages; or an object that is not directly accessible via the Web; e.g. a printed book. Resources are always named by URIs.

- *Properties*: A property is a specific aspect, characteristic, attribute, or relation used to describe a resource.
- *Statements*: A specific resource together with a named property plus the value of that propertyfor that resource is an RDF statement.

These three individual parts of a statement are called, respectively, the *subjec*t, the *predicat*e, and the *objec*t. In a nutshell, RDF defines object-property-value-triples as basic modeling primitives and introduces a standard syntax for them. An RDF document will define properties in terms of the resources to which they apply. For example:

```
<rdf:RDF>
 <rdf:Description about="http://www.w3.org">
  <Publisher>World Wide Web Consortium </Publisher>
</rdf:Description>
</rdf:RDF>
```

states that http://www.w3.org (the subject) has as publisher (the predicate) the W3C (the object). Since both the subject and the object of a statement can be resources, these statements can be linked in a chain:

```
<rdf:RDF>
 <rdf:Description about="http://www.w3.org/Home/Lassila">
  <Creator rdf:resource=http://www.w3.org/staffId/85740"/>
 </rdf:Description>
 <rdf:Description about="http://www.w3.org/staffId/85740">
  <Email>lassila@w3.org</v:Email>
 </rdf:Description>
</rdf:RDF>
```

states that http://www.w3.org/Home/Lassila (the subject) is created by staff member no. 85740 (the object). In the next statement, this same resource (staff member 85740) plays the role of subject to state that his email address is lassila@w3.org. Finally, RDF statements are also resources, so that statements can be applied recursively to statements, allowing their nesting. All this leads to the underlying datamodel being a labelled hyper-graph, with each statement being a predicate-labelled link between object and subject.

The graph is a hyper-graph since each node can itself again contain an entire graph.

5.1.2. Introduction to RDF Schema

The modeling primitives offered by RDF are very basic. Therefore, the RDF Schema specification [Brickley & Guha, 2000] defines further modeling primitives in RDF. That is, RDF Schema extends (or: enriches) RDF by giving an externally specified semantics to specific resources. e.g., to rdfs:subclassOf, to rdfs:Class etc. It is only because of this external semantics that RDF Schema is useful. Moreover, this semantics cannot be captured in RDF - if it could then there would be no need for RDFS.

Figure 4.5 pictures the subclass-of hierarchy of RDFS according to [Brickley and Guha, 2000]. The ‘rdf’ prefix refers to the RDF name space (i.e., primitives with this prefix are already defined in RDF) and ‘rdfs’ refers to new primitives defined by RDFS. Note that RDFS uses a non-standard object-meta model: the properties rdfs:subClassOf, rdf:type, rdfs:domain and rdfs:range are used both as primitive constructs in the definition of the RDF schema specification and as specific instances of RDF properties. This dual role makes it possible to view e.g. rdfs:subClassOf as an RDF property just like other predefined or newly introduced RDF properties, but introduces a self referentiality into the RDF schema definition, which makes it rather unique when compared to conventional model and meta modeling approaches, and makes the RDF schema specification very difficult to read and to formalize, cf. [Nejdl et al., 2000].

5.1.3. The modeling primitives of RDF Schema

In this section, we will discuss the main classes, properties, and constraints in RDFS.

- Core classes are *rdfs:Resource*, *rdf:Property* [3] , and *rdfs:Class*. Everything that is described by RDF expressions is viewed to be an instance of the class *rdfs:Resource*. The class rdf:Property is the class of all properties used to characterize instances of *rdfs:Resource*, i.e., each slot / relation is an instance of *rdf:Property*. Finally, *rdfs:Class* is used to define concepts in RDFS, i.e., each concept must be an instance of *rdfs:Class*.
- Core properties are *rdf:type*, *rdfs:subClassOf*, and *rdfs:subPropertyOf*. The rdf:type relation models instance-of relationships between resources and classes. A resource may be an instance of more than one class. The *rdfs:subClassOf* [4] relation models the subsumption hierarchy between classes and is supposed to be transitive. Again, a class may be subclass of several other classes, however, a class can neither be a subclass of its own nor a subclass of its own sub-classes, i.e., the inheritance graph is cycle-free. The *rdfs:subPropertyOf* relation models the subsumption hierarchy between properties. If some property *P2* is a *rdfs:subPropertyOf* another property *P1* , and if a resource *R*

has a *P2* property with a value *V* , this implies that the resource *R* also has a *P1* property with value *V*. Again, the inheritance graph is supposed to be cycle-free.

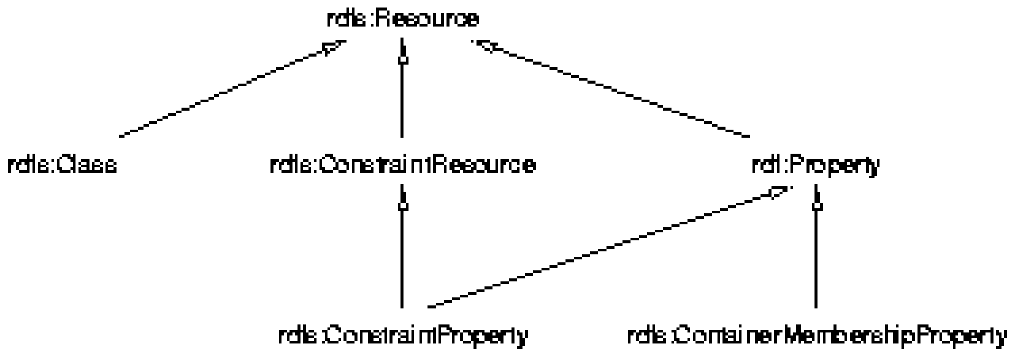


Figure 4.5 The subclass-hierarchy of RDF Schema.

- Core constraints are *rdfs:ConstraintResource*, *rdfs:ConstraintProperty*, *rdfs:range*, and *rdfs:domain*. *rdfs:ConstraintResource* defines the class of all constraints. *rdfs:ConstraintProperty* is a subset of *rdfs:ConstraintResource* and *rdf:Property* covering all properties that are used to define con-straints. At the moment, it has two instances: *rdfs:range* and *rdfs:domain* that are used to restrict range and domain of properties. It is not permitted to express two or more range constraints on a property. For domains this is not enforced and is interpreted as the union of the domains.

5.2. The Relation Between OIL and RDF Schema

RDF Schema and ontology languages aim to a great extent at the same goals. Both provide a kind of domain theory and a vocabulary that can be used in the information exchange. However, there are some differences. We will now investigate those differences, by comparing OIL with RDF Schema. After that, we will further detail on their relation.

5.2.1. Comparing OIL and RDF Schema

The most important similarities and differences between OIL and RDF Schema are the following:

Main modelling primitives. The main modelling primitives of OIL and RDF Schema are the same. OIL uses concepts and slots, RDF Schema calls them classes and properties. In both languages, it is possible to form hierarchies of classes, by the subclass-of relation, and hierarchies of properties, by the subproperty-of relation. However, in contrast to OIL, it is forbidden to form cyclic inheritance relations in RDF Schema.

Local property restrictions. OIL allows the specification of restrictions on properties on a per class basis. We will call this local property restrictions. For example, this allows to define that the fillers of a slot called "has_parent" in a class "dog" should be of type "dog", too, without requiring that the range of "has_parent" is "dog". In RDFS, in contrast, it is only possible to specifiy domain and range restrictions on properties on a global level.

Additional modelling primives. OIL has a number of modeling primitives that does not exist in RDFS: This allows to specify a domain to more detail. Additional constructs are:

Cardinality. To specify the number of fillers of a slot, OIL provides a "min-cardinality", "max-cardinality" and (exact) "cardinality" constructs.

Concrete types. RDFS has only one datatype, which is called "Literal". OIL distinguishes between "string" and "integer", and those datatypes can be used to specify constraints. For example, "young_persons" have an age less than "25".

Intensional class definitions. Class-membership in OIL can be derived from the characteristics of their properties. This enables the intensional definitions of classes. For example, not only do "young_persons" have an age less than "25", but also holds that all persons with an age less than "25" are "young_persons". This can be specified in OIL with the "defined class" construct.

Characteristics of properties. Specific characteristics can be given to properties in OIL. For example, properties can be defined as "transitive" or "functional".

Boolean class expressions. In OIL, one can constructs new anonymous classes as boolean combinations of other classes. OIL provides the operators "and", "or" and "not" to this end.

Axioms. OIL provides axioms, like "disjointness" and "covering". RDF Schema not.

Formal semantics. A final - and important - difference between OIL and RDFS is the formal semantics. The semantics of RDF Schema are only defined in textual descriptions. On the contrary, OIL has an exact mapping to an expressive Description Logic. This gives precise semantics to the constructs, which allows for reasoning on the ontologies. Reasoning on RDF Schemas is very difficult, because of the unclear interpretation of some constructs.

5.2.2. Relating RDF Schema and Ontology Languages

We have seen that the main difference between OIL - or other ontology languages - and RDF Schema is not their purpose, but the expressiveness and formal semantics. Therefore, although the syntaxes differ, their relation can be characterized as an extension relation. OIL adds formal sematics and some additional primitives to that what is provided in RDF Schema.

Now, because the aim of this chapter is to investigate how schema languages can be used to apply ontology-based semantics to online resources, we have to make the additional expressiveness of ontology languages available to RDF Schema. This can be done in the same way as RDF Schema extends RDF! RDF Schema extends (or: enriches) RDF by giving an externally specified semantics to specific resources. e.g., to rdfs:subclassOf, to *rdfs:Class* etc. OIL can be used to further extend (or: enrich) RDF Schema by defining a semantics for specific resources. This allows the extension to capture meaning that cannot be captured in RDF Schema, and this is where the added value is. Furthermore, if such an extension to RDF Schema is carefully designed, a partial interpretation without the additional semantics from the ontology language will still yield a valid RDF Schema interpretation.

```
 Geographical_Location
 Country                  <= Geographical_Location
 City                   <= Geographical_Location
 AreaComparison
KilometerLength-OR-MilesLength
 KilometerLength            <= KilometerLength-OR-MilesLength
 MilesLength              <= KilometerLength-OR-MilesLength
 [,,,]
 coastline
 neighbor_country
```

Figure 4.6 Step 1: materializing the hierarchy.

In Section 2, we will show how such an extension to RDF Schema can be created. We express OIL in RDF Schema, thus enriching it with the required additional expressivity and the semantics of this language. The OIL extension of RDFS has been carefully engineered so that a partial interpretation of OIL meta-data is still correct under the intended semantics of RDFS: simply ignoring the OIL specific portions of an OIL document yields a correct RDF(S) document whose intended RDFS semantics is precisely a subset of the semantics of the full OIL statements. In this way, the approach ensures maximal sharing of meta-data on the Web: even partial interpretation of meta-data by less semantically aware processors will yield a correct partial interpretation of the meta-data.

The result is an RDF Schema definition of OIL primitives, which allows one to express any OIL ontology in RDF Schema syntax. This enables the added benefits of OIL, such as reasoning support and formal semantics, to be used on the Web, while retaining maximal backward compatability with ’pure’ RDF.

6. Applying ontologies to online resources

In the previous sections we have defined the relation between ontologies and XML Schema on the one hand, and ontologies and RDF Schema on the other. We will now show how this relations can be used in practice to apply the benefits of ontologies to online resources.

6.1. Translating an ontology onto an XML Schema prescription

Like (Extended) ER models have to be translated into database schemas to use them in an actual DB system, an OIL ontology can be translated into an XML schema document to use it in an XML data exchange environment. We will provide a translation procedure from an ontology to a specific XMLschema document that is very close in spirit to that provided in [Elmasri & Navathe, 2000] for ER models..ER models provide entities, attributes, and relationships as their primary modeling primitives. This closely corresponds to OIL where we have concepts (i.e., entities), slot definitions for concepts (i.e., attributes), and global slot definitions (i.e., relationships). Extended ER models also incorporate the notion of inheritance, however, require their explicit definition. On the other hand, the relation model only provides relations and the arguments (called attributes) of relations. Therefore, a translation step is required. A similar procedure that translates a high-level conceptual description of a domain into a specific document definition via XML Schema is decribed below.

We assume a definition of an ontology in OIL. An example is provided in Figure 4.1. We will now describe its stepwise translation into an XML schema using the stepwise translation of this example as illustration.

First, materialize the hierarchy. Give all complex class expressions that are used in subclass definitions and slot constraints names. Then, materialize the hierarchy, i.e., make all class- and slot-subsumptions explicit. This is necessary because XML Schema lacks any notion of implicit hierarchy and it is possible because subsumption is decidable in OIL. Actually, the FaCT system can be used for this purpose (via its CORBA interface if desired [Bechhofer et al., 1999])[9]. In this step, make also explicit which slots can be applied to a class, exploiting the domain and range restrictions of the global slots definitions. Figure 4.6 provides the materialized hierarchy of our running example. Note that *KilometerLength-OR-MilesLength* is a new concept, constructed from a complex class expression. In our small example, there are no new class subsumptions derived, because all of them are already stated explicitly. See [Horrocks et al., 2000] or [Fensel et al., 2000] for a more complex example which illustrates the derivation of implicit subsumptions.

```
Part of the result of step 2: type definitions for slots:
<xsd:complexType name="capitalType">
<xsd:sequence>
 <xsd:element ref="City"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="coastlineType">
<xsd:sequence>
 <xsd:element ref="KilometerLength-OR-MilesLength"/>
</xsd:sequence>
</xsd:complexType>
Part of the result of step 3: type definitions for classes:
<xsd:complexType name="CountryType">
<xsd:complexContent>
 <xsd:extension base="GeographicalLocationType">
   <xsd:sequence>
    <xsd:element name="capital" type="capitalType" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="has-boundary" type="has_boundaryType" minOccurs="0" maxOc-
curs="unbounded"/>
    <xsd:element name="relative_area" type="relative_areaType" minOccurs="0" maxOc-
curs="unbounded"/>
   </xsd:sequence>
 </xsd:extension>
</xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="AeraComparisonType">
<xsd:sequence>
 <xsd:element name="proportion" type="string" minOccurs="0" maxOccurs="unbounded"/>
 <xsd:element name="compared_to" minOccurs="0" maxOccurs="unbounded">
   <xsd:complexType>
    <xsd:complexContent>
       <xsd:extension base="compared_toType">
          <xsd:sequence>
             <xsd:element ref="Country"/>
          </xsd:sequence>
       </xsd:extension>
    </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
</xsd:sequence>
</xsd:complexType>
Part of the result of step 4: element definitions for slots and classes:
<xsd:element name="capital" type="capitalType"/>
<xsd:element name="GeographicalLocation" type="GeographicalLocationType"/>
<xsd:element name="Country" type="CountryType"/>
<xsd:element name="City" type="CityType"/>
<xsd:element name="AeraComparison" type="AeraComparisonType"/>
```

Figure 4.7 Examples of the result of step 2 and 3

Second, create a complexType definition for each slot definition in OIL. Add a reference to the (still to be defined) element definition for the *range* component in the OIL slot-definition Figure 4.7 begins with some example slot-definitions. If a slot has more than one range, an intermediate element must be used that is the conjunction of all the range components. The place of this intermediate element in the class hierarchy should be derived in the first step.

Third, also create a complexType definition for each class definition in OIL. Add the names of the slots that can be applied to the classes as elements in the type definition. The facets on the slot-constraints are translated in the following way: *has-value* facets give a *minOccurs="1"* attribute in the *element*-element, *value-type* facets give *minOccurs="0"* and *min-cardinality, max-cardinality,* and *cardinality* gives *minOccurs="value", maxOccurs="value"* or both as attributes respectively. If a slot has the property *functional* it will get the attribute *maxOccurs="1".* See for example the attributes of "capital" in "CountryType" in Figure 4.7: the value of "minOccurs" is 1 because of the has-value constraint, and the value of "maxOccurs" is 1 because of the "functional" property.

For the slots that appear in explict slot-constraints (i.e., those that are actually described in a class-definition, not those that can be applied to a class according to their domain), an anonymous type is defined, which is derived from the appropriate slot-type defined in step two. The extension of the base type consist of the reference to the class which must be the filler of the slot.

```
<?xml version="1.0" encoding="UTF-8"?>
<Country xmlns="worldfactbook.xsd"
     xmlns:xsi="http://www.w3.org/1999/XMLSchema/instance">
<name>The Netherlands</name>
<capital><City><name>Amsterdam</name></City></capital>
<relative_area>
 <AeraComparison>
   <proportion>slightly less than twice the size of</proportion>
   <compared_to><Country><name>New Jersey</name></Country></compared_to>
 </AeraComparison>
</relative_area>
<has_boundary>
 <LandBoundary>
   <neighbor_country><Country><name>Belgium</name></Country></neighbor_country>
   <length xsi:type="KilometerLength"><value>450</value><unit>km</unit></length>
 </LandBoundary>
 <LandBoundary>

<neighbor_country><Country><name>Germany</name></Country></neighbor_country>
   <length xsi:type="KilometerLength"><value>577</value><unit>km</unit></length>
 </LandBoundary>
</has_boundary>
<coastline><KilometerLength><value>451</value><unit>km</unit></KilometerLength></co
astline>
</Country>
```

Figure 4.8 Example of XML document conforming the generated schema.

For slots that can be applied to the classes (according to their domain) but that are not present in explict slot-constraints, the type of the element is directly the slot-type from step 2, with the attributes *minOccurs="0"* and *maxOccurs="unbounded"*. The second part of Figure 4.7 gives an example.

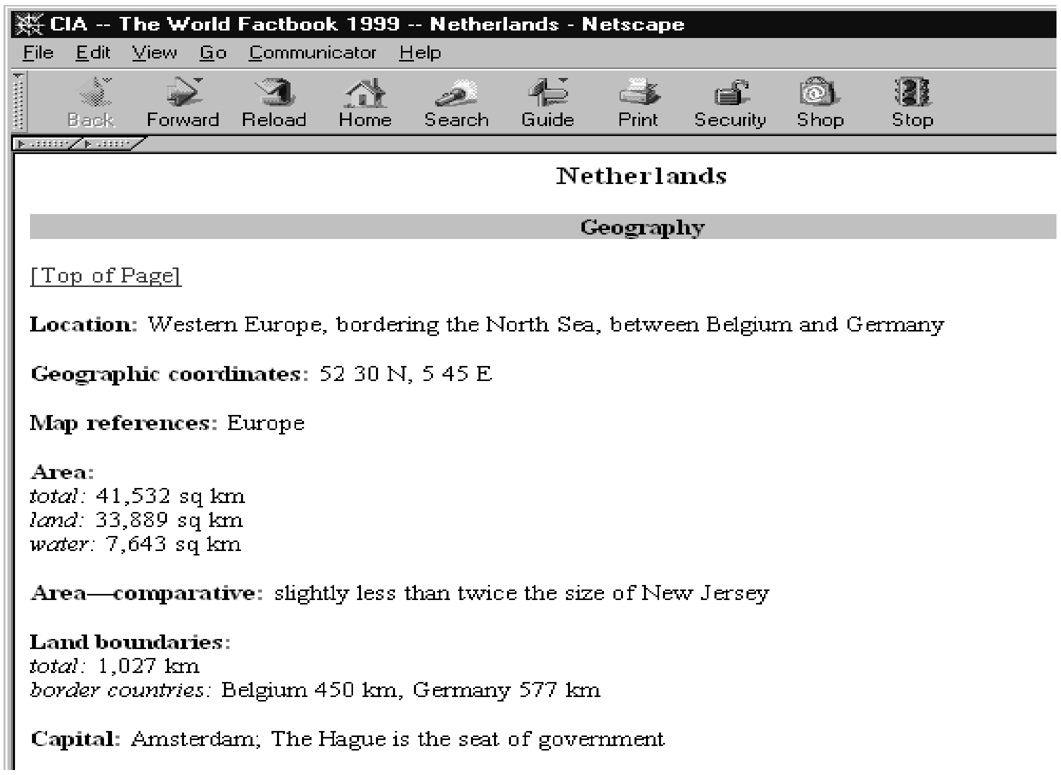
Fourth, create an element definition for each slot and class. Each slot and each class definition is translated into an element definition in the XML schema. The type of the elements will obviously be the complexType definitions which are created in the second and third step. See also Figure 4.7.

Fifth, define a grammar for each entity, associate basic datatypes with built-in datatypes if desired, add lexical constraints on datatypes if desired. This step adds an additional level of expressiveness that is not present in OIL. It is purely concerned with document structure and appearance.

Sixth, replace the module concept of OIL with the namespace and inclusion concept of XML Schema. This step is straightforward because the concepts only differ syntactically.

6.1.1. Using the Schema for Document Markup

The resulting schema can be used to create XML instance documents. The structure of these documents must conform to the schema. As an example, we show in Figure 4.8 an XML document which could constitute a wepage in the World Fact Book. Together with a appropriate stylesheet, this document can be used to produce a page as is shown in Figure 4.9. Note that we now have a webpage which looks similar to the original HTML version, but which has a markup that is well-founded on an ontology.


CIA -- The World Factbook 1999 -- Netherlands - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Shop Stop

Netherlands

Geography

[Top of Page]

Location: Western Europe, bordering the North Sea, between Belgium and Germany

Geographic coordinates: 52 30 N, 5 45 E

Map references: Europe

Area:
total: 41,532 sq km
land: 33,889 sq km
water: 7,643 sq km

Area—comparative: slightly less than twice the size of New Jersey

Land boundaries:
total: 1,027 km
border countries: Belgium 450 km, Germany 577 km

Capital: Amsterdam; The Hague is the seat of government

Figure 4.9 An possible view on the data from Figure 4.8.

6.1.2. Problems In the Translation Procedure

In the following, we will discuss some of the points that arose when investigating the relation between ontologies and XML schemas.

First, multiple inheritance forms a problem in the translation procedure. As we already discussed in Section 4.2.1, in XML Schema there is no explicit way to define multiple inheritance. It is not possible to simulate multiple inheritance by inheriting from different superclasses in several type definitions, because conform the XML Schema specification it is not allowed to have more than one definition for a type. Because of this lack of multiple inheritance, XML Schema is not well suited as an ontology language. This is not meant as a criticism, because XML Schema is not designed for ontological modeling, it is designed for describing valid structures of documents.

Second, the question may arise as to whether the translation process can be automated and whether it is reversible. Concerning the first question, we can state that most of the steps can be completely automatic. The fifth step can be partially automated, for example by using sequence as standard grammar for applying slots to classes. Final tuning via human interference may be necessary. The reverse direction is possible but more difficult, a high degree of automatization should be achievable, however.

6.2. Defining an Ontology Language as Extension to RDF Schema

Now we have shown a mechanims to use XML Schema for applying ontologies to online resources, we will now discuss a second method. We will show how OIL ontologies can be written down in RDF, while trying to make use of existing RDFS constructs as much as possible, but where necessary extending RDFS with additional constructs.

6.2.1. Class Definitions

In RDFS, classes are simply declared by giving them a name (with the ID attribute). In OIL, a class definition links a class with a name, a documentation, a type, its superclasses, and the attributes defined for it. We need to extend the schema language to capture all the knowledge of an OIL ontology. To illustrate the use of these extensions, we will walk through them by means of some example OIL class definitions that need to be represented in RDFS syntax:

class-def defined herbivore

subclass-of animal

slot-constraint eats

value-type (plant or (slot-constraint *is-part-of plant*))

class-def elephant

subclass-of herbivore mammal

slot-constraint eats

value-type plan

slot-constraint colour

has-filler "grey"

The first defines a class "herbivore", a subclass of animal, whose instances eat plants or parts of plants. The second defines a class "elephant", which is a subclass of both herbivore and mammal.

OIL distinguises between primitive classes and defined classes. To express the type of a class in the RDFS syntax, we chose to introduce two extra classes as extension to RDFS, named "oil:PrimitiveClass" and "oil:DefinedClass". In a particular class definition, we can use one of these meta-classes to express the type, e.g.:

```
<oil:DefinedClass rdf:ID="herbivore"> </oil:DefinedClass>
```

This way of making an actual class an instance of either DefinedClass or PrimitiveClass introduces a nice object-meta distinction between the OIL RDFS schema and the actual ontology: using rdf:type you can consider the class "herbivore" to be an instance of DefinedClass. In OIL in general, if it is not explicitly stated that a class is defined, the class is assumed to be primitive.

Next, we have to translate the subclass-of statement to RDFS. This also can be done in a straightforward manner, simply re-using existing RDFS expressiveness:

```
<oil:DefinedClass rdf:ID="herbivore">
 <rdfs:subClassOf rdf:resource="#animal"/>
</oil:DefinedClass>
```

To define a class as a subclass of a class expression (which is not possible in RDFS), we introduced the additional "oil:subClassOf property" in the RDFS extension.

We still need to serialize the slot constraint on the class "herbivore". In RDFS, there is no mechanism for restricting the attributes of a class on a local level. This is due to the property-centric nature of the RDF data model: properties are defined globally, with their domain description coupling them to the relevant classes. To overcome this problem, we introduce the property "*oil:hasPropertyRestriction*" in the RDFS extension, which is an *rdf:type* of rdfs:ConstraintProperty (analogous to *rdfs:domain* and *rdfs:range*). Here we take full advantage of the intended extensibility of RDFS. We also introduce oil:PropertyRestriction as a place-holder class for specific classes of slot constraints, such as has-value, value-type, cardinality and so on. For the three cardinality constraints, an extra property "number" is introduced, which is used to assign a concrete value to the cardinality constraints. In our example ontology, the first part of the slot constraint would be serialized using the primitives introduced above as follows:

```
<oil:DefinedClass rdf:ID="herbivore">
 <rdfs:subClassOf rdf:resource="#animal"/>
 <oil:hasPropertyRestriction>
  <oil:ValueType>
   <oil:onProperty rdf:resource="#eats"/>
   <oil:toClass> </oil:toClass>
  </oil:ValueType>
 </oil:hasPropertyRestriction>
</oil:DefinedClass>
```

The slot constraint has not been completely translated yet: the toClass element is not yet filled. Here we come across another feature of OIL that is not available in RDFS: the class expression. A class expression is an expression that evaluates to a class definition. Such an expression can be a simple class name, or it can be a boolean expression of classes and/or slot constraints. In the example, we have a boolean 'or' expression that evaluates to the class of all things that are plants or that are parts of plant. We introduce *oil:ClassExpression* and *oil:BooleanExpression* to hold the operators "and", "or" and "not" as subclasses.

Also, since a single class is a essentially a simple kind of class-expression, *rdfs:Class* itself should be a subclass of *oil:ClassExpression*. The "and", "or" and "not" operators are connected to operands using the newly introduced "*oil:hasOperand*" property. This property again has no direct equivalent in OIL primitive terms, but is a helper to connect two class-expressions, because in the RDF data model one can only relate two classes by means of a Property.

We can now fill the toClass part of our example as follows:

```
<oil:toClass>
 <oil:Or>
  <oil:hasOperand rdf:resource="#plant"/>
  <oil:hasOperand>
   <HasValue>
    <oil:onProperty rdf:resource="#is-part-of"/>
    <oil:toClass rdf:resource="#plant"/>
   </HasValue>
  </oil:hasOperand>
 </oil:Or>
</oil:toClass>
```

6.2.2. Slot Definitions

Both OIL and RDFS allow slots as first-class citizens of an ontology. Therefore, slot definitions in OIL map nicely onto property definitions in RDFS. Also the "subslot-of", "domain", and "range" properties have almost direct equivalents in RDFS. However, there are a few subtle differences between domain and range restrictions in OIL and their equivalents in RDFS. First, the specification of OIL is very clear on multiple domain and range restriction: these are allowed, and the semantic is the intersection of the individual statements (conjunctive semantics). In RDFS, multiple domain statements are allowed, but their interpretation is the union of the classes in the statements (disjunctive semantics).

Secondly, in contrast to RDFS, OIL not only allows classes as range and domain of properties, but also class-expressions, and - for range - concrete-type expressions. It is not possible to reuse rdfs:range and rdfs:domain for these sophisticated expressions, because of the conjunctive semantics of multiple range statements: we cannot extend the range of rdfs:range or rdfs:domain, we can only restrict it. Therefore, we introduced two new ConstraintProperties *oil:domain* and *oil:range*. They have the same domain as their RDFS equivalent (i.e., *rdf:Property*), but have a broader range. For domain, class expressions are valid fillers, for range both class expressions and concrete type expressions may be used.

When translating a slot definition, rdfs:domain and rdfs:range should be used for simple (one class) domain and range restrictions, while for more complicated statements the oil:range or oil:domain properties should be used. For example:

```
slot-def age
  domain animal
  range   (range 0 120)
```

is in the RDFS representation:

```
<rdf:Property rdf:ID="age">
<rdfs:domain rdf:resource="#elephant"/>
<oil:range>
  <oil:Range>
   <oil:integerValue>0</oil:integerValue>
   <oil:integerValue>120</oil:integerValue>
  </oil:Range>
 </oil:range>
</rdf:Property>
```

However, global slot-definitions in OIL allow specification of more aspects of a slot than property definitions in RDFS do. Besides the domain and range restrictions, OIL slots can also have an "inverse" attribute and qualities like "transitive" and "symmetric". We therefore add a property "inverseRelationOf" with "rdf:Property" as domain and range. We also added the classes "TransitiveProperty", "FunctionalProperty" and "SymmetricProperty" to reflect the different qualities of a slot. In the RDFS-serialization of OIL, the rdf:type property can be used to add a quality to a property. For example, the OIL definition of:

```
slot-def has-part
  domain is-part-of
  range   (range 0 120)
```

is in RDFS:

```
<oil:TransitiveProperty rdf:ID="has-part">
<oil:inverseRelationOf rdf:resource="#is-part-of"/>
</oil:TransitiveProperty>
```

This way of translating the qualitities of properties features the same nice object-meta distinction between the OIL RDFS schema and the actual ontology as the translation of the "type" of a class (see section 4.2). In an actual ontology, the property "has-part" can be considered as an instance of a TransitiveProperty. Note that it is allowed to make a property an instance of more than one class, and thus giving it multiple qualities.

6.2.3. Representing Axioms

Axioms in OIL are factual statements about the classes in the ontology. They correspond to n-ary relations between class expressions, where n is 2 or greater. RDF only knows binary relations (properties). Therefore, we cannot simply map OIL axioms to RDF properties. Instead, we chose to model axioms as classes, with helper properties connecting them to the class expressions involved in the relation. Since axioms can be considered objects, this is a very natural approach towards modeling them in RDF (see also [Staab & Madche, 2000],[Staab et al., 2000]). Observe also that binary relations (properties) are modeled as objects in RDFS as well (i.e., any property is an instance of the class rdf:Property). We simply introduce a new primitive alongside rdf:Property for relations with higher arity.

We introduce a placeholder class "oil:Axiom", and model specific types of axioms as subclasses and likewise for Equivalent. We also introduce a property to connect the axiom object with the class expressions it relates to each other: oil:hasObject is a property connecting an axiom with an object class expression. For example, to serialize the axiom that herbivores, omnivores and carnivores are (pairwise) disjoint:

```
<oil:Disjoint>
<oil:hasObject rdf:resource="#herbivore"/>
<oil:hasObject rdf:resource="#carnivore"/>
<oil:hasObject rdf:resource="#omnivore"/>
</oil:Disjoint>
```

For modeling covering axioms, we introduce a seperate placeholder class, "oil:Covering", which is a subclass of "oil:Axiom". The specific types of coverings available are modeled as subclasses of "oil:Covering" again. Furthermore, two additional properties are introduced: "oil:hasSubject", to connect a covering axiom with its subject, and "oil:isCoveredBy", which is a subproperty of "oil:hasObject", to connect a covering axiom with the classes that cover the subject. For example, we serialize the axiom that the class animal is covered by carnivore, herbivore, omnivore, and mammal (i.e. every instance of animal is also an instance of at least one of the other classes) as follows:

```
<oil:Cover>
<oil:hasSubject rdf:resource="#animal"/>
<oil:isCoveredBy rdf:resource="#carnivore"/>
<oil:isCoveredBy rdf:resource="#herbivore"/>
<oil:isCoveredBy rdf:resource="#omnivore"/>
<oil:isCoveredBy rdf:resource="#mammal"/>
</oil:Cover>
```

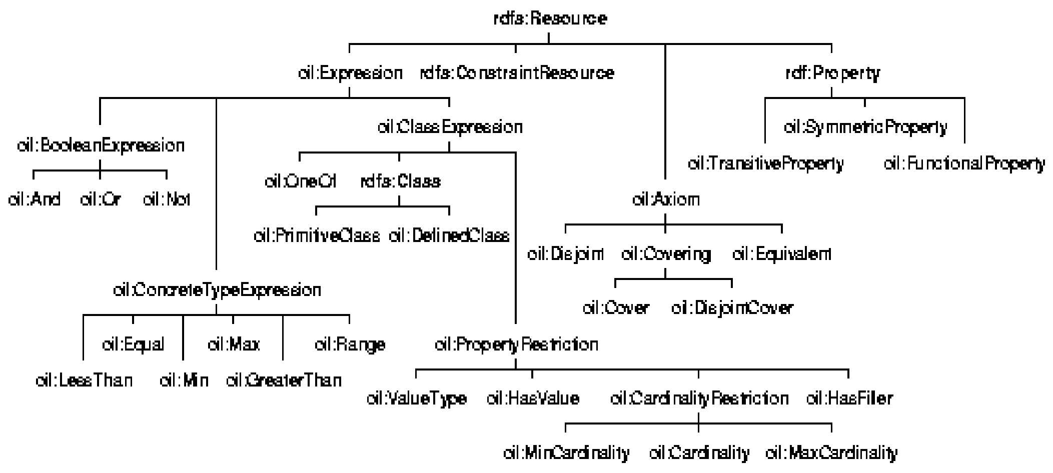


Figure 4.10 The is-a hierarchy of RDFS primitives (in bold) with the OIL extenstions.

6.2.4. Compatability With RDF Schema

Figure 4.10 shows the resulting is-a hierarchy of the OIL constructs that we introduced to define OIL ontologies in RDFS representation. The hierarchy nicely illustrates that the new OIL langauge representation includes RDF Schema as a sublanguage. As a result, all RDF Schema expressions are actually also valid and meaningful OIL expression. This is called backward compatibility. Of course, since OIL is an extension of RDF Schema, not all OIL definitions are interpretable as RDF Schema definitions. However, the way in which this extension is defined ensures that all OIL definitions are still valid RDF Schema expressions, although only partly meaningful. The partial interpretation can be achieved by simply ignoring any statement not from the rdf or rdfs namespaces (in our example those from the oil namespace). For example, in the above definition of "herbivore", an RDF Schema processor will interpret this statement simply as stating that herbivores are a subclass of animals, and that they some other property that it cannot interpret. This is a correct albeit partial interpretation of the definition. Furthermore, because OIL is built on top of RDF Schema, it shares with it the mechanism to represent instance information.

Together, this gives another important compatibility result besides the *backward compatibility*: even if an ontology is written in the richer modeling language (OIL), a processor for the simpler ontology language (RDF Schema) can still: fully interpret all the instance information of the ontology, and partially interpret the class-structure of the ontology.

This can be called *partial forward compatibility*. Such partial interpretability of semantically rich meta-data by semantically poor is a crucial step towards the sharing of meta-data on the Semantic Web. We cannot realistically hope that all of the Semantic Web will be build on a single standard for semantically rich meta-data. The above shows that multiple semantic modeling languages do not have to lead to meta-data that are totally uninterpretable by others. Instead, simpler processors can still pick up as much of the meta-data from rich processors as they can "understand", and safely ignore the rest in the knowledge that their partial interpretation is still a correct with respect to the original intention of the meta-data.

Conclusion

In this chapter, we discussed two important schema languages for the web. We compared these languages with full-fledged ontology languages, and described the relation between them. Moreover, we have shown in two procedures how these schema languages can be used to apply ontologies to online resources.

When comparing the Ontology Inference Layer OIL with the proposed XML Schema, our main conclusion is that an ontology language and XML Schema refer to a different level of abstraction. Therefore, OIL and XML Schema each play a role in a different phase in describing the semantics of on-line information sources. OIL is suitable for domain modeling, XML Schema can be used to prescribe how the information in a domain is structured. The comparison also revealed that the XML Schema type hierarchy can be used to express some conceptual knowledge. Although there are still some questions about the best way to create this type hierarchy, with the help of some artifices it is possible to capture the central is-a relationship of an ontology in an XML Schema definition.

We exploited this possibility in a translation procedure from an OIL ontology to an XML structure prescription, which we provided as an illustration of their relation. As a result of the translation procedure, a document schema is created, which is founded in a domain ontology. The main advantage of this translation is that an ontology can be used as a conceptual layer on top of a set of structured (XML) documents, because the markup of actual XML documents is founded in an ontology. Another benefit of this procedure is that it yields an XML Schema that can be used to encode instance data of an ontology in plain XML.

We have also compared RDF Schema with OIL. We have observed that there is a big difference in the level of expressiveness. However, we have shown how full-fledged ontology languages can be applied in the Resource Description

Framework. We did this by representing the modeling primitives as defined by OIL in RDF Schema. The OIL extension of RDFS has been carefully engineered so that a partial interpretation of OIL meta-data is still correct under the intended semantics of RDFS: simply ignoring the OIL specific portions of an OIL document yields a correct RDF(S) document whose intended RDFS semantics is precisely a subset of the semantics of the full OIL statements. In this way, the approach ensures maximal sharing of meta-data on the Web: even partial interpretation of meta-data by less semantically aware processors will yield a correct partial interpretation of the meta-data.

We can conclude that, despite the similarity in their names, RDF Schema and XML Schema fulfil a different role. First, XML Schemas, and also DTDs, prescribe the order and combination of tags in an XML document. In contrast, RDF Schemas provides information about the interpretation of the statements given in RDF data (i.e., RDF Schema is a simple type system for RDF), but it does not constrain the syntactical appearance of an RDF description. From another point of view, RDF Schema and XML Schema can be considered as orthogonal to each other: RDF is mainly intended for describing explicit metadata about webresources as a whole; XML Schema gives semantics to the actual markup of a document (i.e. the tags and their stucture) and answers the question how the structure of documents is related to conceptual terms.

This chapter provided an illustration of how these two ortogonal mechanisms can be used to apply ontologies to online resources. XML Schema and RDF Schema each play a different role in this process. The semantically grounded document markup that is provided by XML Schema complements the kind of semantic description that is provided by RDF based approaches. We think that both mechanisms are important building stones for the semantic web. Finally, we believe that this way of translating and extending is generally applicable across other ontology languages then OIL.

References

[Bechhofer et al., 1999] S. Bechhofer, I. Horrocks, P. F. Patel-Schneider, and S. Tessaris: A proposal for a description logic interface. In P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, and P. Patel-Schneider, editors, Proceedings of the International Workshop on Description Logics (DL'99), pages 33-36, 1999.

[Biron & Malhotra, 2000] P. V. Biron and A. Malhotra: XML Schema Part 2: Datatypes, Candidate Recommendation, 24 October 2000. http://www.w3.org/TR/2000/CR-xmlschema-2-20001024/

[Bray, 1998] Extensible markup language (XML) 1.0 W3C recommendation, 1998. http://www.w3.org/TR/REC-xml

[Brickley & Guha, 2000] D. Brickley and R.V. Guha: Resource Description Framework (RDF) Schema Specification 1.0, W3C Candid1ate Recommendation 27 March 2000. http://www.w3.org/TR/2000/CR-rdf-schema-20000327.

[Broekstra et al., 2000] J. Broekstra, M. Klein, D. Fensel, S. Decker, F. van Harmelen and I. Horrocks: Enabling knowledge representation on the Web by extending RDF Schema, Proceedings of the 10th World Wide Web conference, May 1-5, 2001, Hong Kong. http://www.cs.vu.nl/~mcaklein/papers/www10/

[Elmasri & Navathe, 2000] R. Elmasri and S. B. Navathe: Fundamentals of Database Systems, 3rd ed., Addison Wesley, 2000.

[Fallside, 2000] David C. Fallside: XML-Schema Part 0: Primer, Candidate Recommendation, 24 October 2000. http://www.w3.org/TR/2000/CR-xmlschema-0-20001024/

[Fensel, 2000] D. Fensel: Relating Ontology Languages and Web Standards. In J. Ebert et al. (eds.), Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik, Modellierung 2000, St. Goar, April 5-7, 2000, Foelbach Verlag, Koblenz, 2000.

[Fensel, 2001] D. Fensel: Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce, Springer-Verlag, 2001.

[Fensel et al., 2000] D. Fensel, I. Horrocks, F. Van Harmelen, S. Decker, M. Erdmann, and M. Klein: OIL in a Nutshell. In, Dieng, R., and Corby, O., eds, Proceedings of EKAW-2000: The 12th International Conference on Knowledge Engineering and Knowledge Management. Lecture Notes in Computer Science, Springer Verlag, October 2000.

[Guarino & Welty, 2000] N. Guarino and C. Welty: A Formal Ontology of Properties, In, Dieng, R., and Corby, O., eds, Proceedings of EKAW-2000: The 12th International Conference on Knowledge Engineering and Knowledge Management. Lecture Notes in Computer Science, Springer Verlag, October 2000.

[Horrocks et al., 2000] I. Horrocks, D. Fensel, J. Broekstra, S. Decker, M. Erdmann, C. Goble, F. Van Harmelen, M. Klein, S. Staab, and R. Studer: OIL: The Ontology Inference Layer, Technical Report IR-479, Vrije Universiteit Amsterdam, September 2000. http://www.ontoknowledge.com/oil.

[Lassila & Swick, 1999] O. Lassila and R. R. Swick: Resource Description Framework (RDF): Model and Syntax Specification, W3C Recommendation, 22 February 1999. http://www.w3.org/TR/REC-rdf-syntax/

[Nejdl et al., 2000] W. Nejdl, M. Wolpers, and C. Capella: The RDF Schema Revisited. In Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik, Modellierung 2000, St. Goar. Foelbach Verlag, Koblenz, 2000.

[Staab et al., 2000] S. Staab, M. Erdmann, A Madche, and S. Decker: An extensible approach for modeling ontologies in RDF(S). In First Workshop on the Semantic Web at the Fourth European Conference on Digital Libraries, Lisbon, Portugal, 2000.

[Staab & Madche, 2000] S. Staab and A. Madche: Axioms are objects, too - ontology engineering beyond the modeling of concepts and relations. In Benjamins, V., Gomez-Perez, A., and Guarino, N., editors, Proceedings of the Workshop on Applications of Ontologies and Problem-solving Methods, 14th European Conference on Arti_cial Intelligence ECAI 2000, Berlin, Germany, 2000.

[Thompson et al., 2000] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn: XML Schema Part 1: Structures, W3C Candidate Recommendation, 24 October 2000. http://www.w3.org/TR/2000/CR-xmlschema-1-20001024/.

[Walsh, 1999] N. Walsh: Schemas for XML, July 1, 1999. http://www.xml.com/pub/1999/07/schemas/ index.html.

Footnotes

[1]Department of Computer Science, Vrije Universiteit Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, the Netherlands, {mcaklein| dieter|frankh}@cs.vu.nl

[2]Aidministrator bv, Amersfoort, the Netherlands, jeen.broekstra@aidministrator.nl

[3]Department of Computer Science, University of Manchester, Oxford Road, Manchester, M13 9PL, UK, horrocks@cs.man.ac.uk

[4]http://www.odci.gov/cia/publications/factbook/

[5]http://purl.org/DC/

[6] However, XML Schema does not provide any formal language for these intensional definitions. Actually primitive datatypes are defined in prose or by reference to another standard. Derived datatypes can be constrained along their facets (such as maxInclusive, maxExclusive etc.).

[8]A general comparison of type systems and description logics can be found in [Borgida, 2000]

[9] OilEd is an integrated tool for building OIL ontologies and exploiting the FaCT reasoner; it can found at: http://img.cs.man.ac.uk/oil/

5. UPML: The Language and Tool Support for Making the Semantic Web Alive

B. Omelayenko[1], M. Crubézy[2], D. Fensel[1], R. Benjamins[3], B. Wielinga[4], E. Motta[5], M. Musen[2], Y. Ding[1]

1. Introduction

Originally, the Web grew mainly around representing static information using the HTML language, which provided a standard for document layout and was interpreted by browsers in a canonical way to render documents. On the one hand, it was the simplicity of HTML that enabled the fast growth of the WWW. On the other hand, its simplicity seriously hampered more advanced Web application in many domains and for many tasks. *The Semantic Web* [Berners-Lee & Fischetti, 1999] will transform the current World-Wide Web into a network of resources structured with the annotations defining their meaning and relationships. In this context, computers not only provide more efficient access to Web resources, but are also able to perform intelligent tasks with those resources. The explicit representation of the semantics of data, accompanied with domain theories (i.e. *ontologies*), will enable a Web that provides a qualitatively new level of service. It will weave an incredibly large network of human knowledge and will complement it with machine processability. Various automated services will help users to achieve their goals by accessing and providing necessary information in a machine-understandable form. This process might ultimately create an extremely knowledgeable system with various specialized reasoning services – the systems which can support us in many aspects of life.

Many steps need to be taken to make this vision become true. Languages and tools for enriching information sources with machine-processable semantics must be developed (cf. [Fensel et al., 2001], [Fensel et al., 2000(b)]). Web-based reasoning services need to be developed, services which employ these semantic-enriched information sources to provide intelligent support to human users in task achievement. Specifically, such services will need to support users not only in finding and accessing information but also in achieving their goals based on this information.

The objective of the IBROW[6] project is to develop an intelligent broker able to configure knowledge systems from reusable components on the World Wide Web ([Fensel & Benjamins, 1998], [Benjamins et al., 1999]). IBROW brokers will handle Web requests for some classes of knowledge systems (e.g., diagnostic systems) by accessing libraries of reusable reasoning components on the Web, and selecting, adapting, and configuring them in accordance with the domain in question. This project integrates research on heterogeneous databases, interoperability, and Web technology with knowledge-system technologies, namely ontologies and problem-solving methods.

Nowadays, ontologies [Gruber, 1993] represent mainly *static* and declarative knowledge about a domain of expertise. The way to apply domain knowledge to achieve user tasks, or *dynamic* knowledge, is usually encoded in inference algorithms, which reason on the contents of the domain ontologies. Making this dynamic knowledge explicit and generic, and regarding it as an important part of the entire knowledge contained in a knowledge-based system (KBS) is the rationale which underlies *problem-solving methods* (cf. [Stefik, 1995], [Benjamins & Fensel, 1998], [Benjamins & Shadbolt, 1998], [Motta, 1999], [Fensel, 2000]).

Problem-solving methods (PSMs) provide reusable components for implementing the reasoning part of the KBSs.[7] PSMs refine inference engines to allow a more direct control of the reasoning process of a system to achieve a task. PSMs encode control knowledge independently of an application domain: PSMs can therefore be reused for different domains and applications. PSMs decompose the reasoning task of a knowledge-based system in a number of subtasks and inference actions, which are connected by knowledge roles representing the ‘role’ that knowledge plays in the reasoning process. Several libraries of problem-solving methods have been developed (cf. [Marcus, 1988], [Chandrasekaran et al., 1992], [Puppe, 1993], [Breuker & van de Velde, 1994], [Benjamins, 1995], [Musen 1998], and [Motta, 1999]) and a number of problem-solving method specification languages have been proposed, ranging from informal notations (e.g. CML [Schreiber et al., 1994]) to formal modeling languages (see [Fensel & van Harmelen, 1994], [Fensel, 1995], [Gomez Perez & Benjamins, 1999] for summaries).

Ontologies and PSMs provide the components which need to be combined by the IBROW broker to configure a particular knowledge system on the Web. As a central requirement, both types of components, ontologies and PSMs, need to be properly marked-up to be localized on the Web and assembled into a working system by the broker. In the IBROW project, we have developed *the Unified Problem-solving Method Development Language* UPML[8] to specify problem-solving components and their software architectures to facilitate their semi-automatic reuse and adaptation (see [Fensel & Benjamins, 1998], [Fensel et al., 1999(a)], [Fensel et al., 1999(b)], [Fensel et al., to appear]). Concisely, UPML is a framework for specifying knowledge-intensive reasoning systems based on libraries of generic problem-solving components. Since its first definition, UPML has been adopted by the members of the IBROW consortium and has been used to specify a design library[9] [Motta et al., 1998] at the Open University, a library for classification problem-solving [Motta & Lu, 2000], and parts of the PSM library at Stanford University (cf. [Musen 1998]).

The goal of this chapter is to provide an overview of the UPML framework and tools and to demonstrate how this language provides the enabling technology for the next level of service of the World-Wide Web: to provide users of the Web with online, ready-to-use problem-solving resources. In Section 2, we discuss the mission of IBROW in the context of the future Web and the central role of a specification language such as UPML. In Section 3, we introduce the UPML language for component markup. Tool support for UPML is provided by the Protégé environment, as presented in Section 4. Concluding remarks are presented in Section Conclusions.

2. Brokering Reasoning Components on the Web

The mission of the IBROW project is to develop an intelligent brokering service capable of retrieving a set of knowledge components from the Web which, combined together, can solve the users' problem, according to stated requirements [Benjamins et al., 1999]. As illustrated in Figure 5.1, the main goal of the IBROW broker is to identify the components needed to solve the problem, and to adapt and configure them into a running reasoning service. In the context of the World-Wide Web, the task of the broker includes reusing third-party components available on the Web, and operating problem solvers and knowledge bases together in a distributed, 'plug-and-play' fashion.

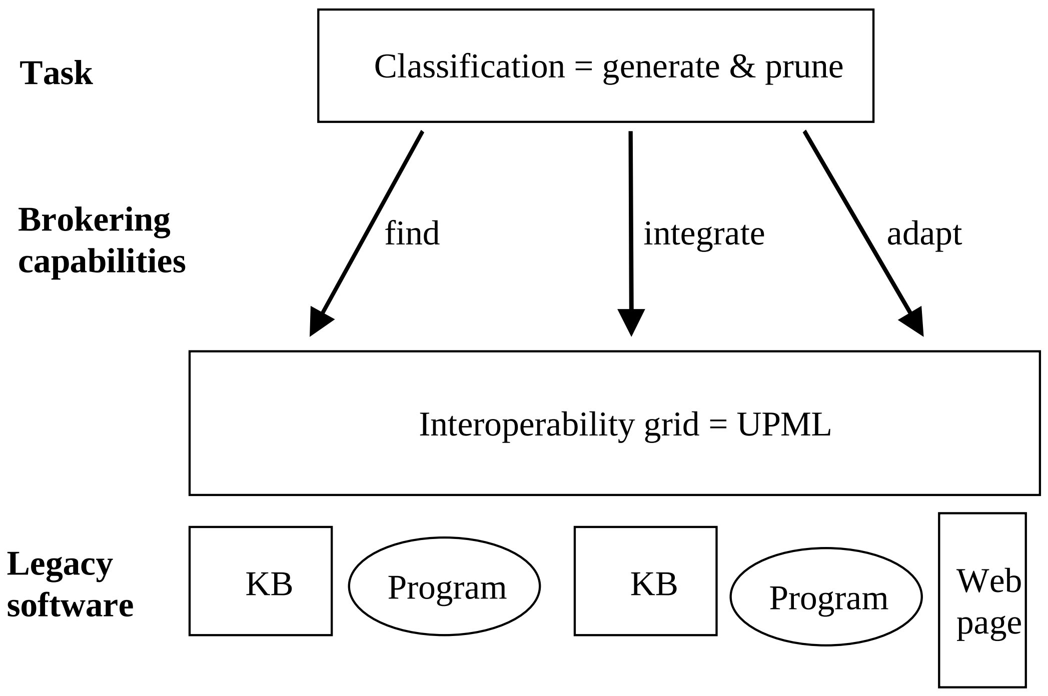


Figure 5.1 The IBROW approach. Based on user requirements about a task to be solved, IBROW broker needs to find, integrate and adapt a set of suitable knowledge components (i.e., knowledge bases and problem-solving methods) on the Web to solve a user-specified task. UPML is a central interoperability element of the IBROW approach.

The IBROW approach offers a number of innovative aspects. Most of today’s Web brokers (e.g. Metacrawler[10], Ariadne[11], Ontobroker[12]) handle only static information, whereas the IBROW broker is capable of managing dynamic information[13]. As a result, our approach offers an opportunity for a new type of electronic marketplaces, where reasoning services can be configured dynamically out of independent components to solve a specific task. In particular, IBROW will enable the construction of ‘throw-away’ applications for Web users.

A key issue in the IBROW approach is that reasoning components themselves need to be available on the Web. Moreover, components must be marked-up with machine-processable data, so that an IBROW broker can identify their capabilities and reason about them. This requirement calls for the development of a language for component markup, which must support the specification of the capabilities and assumptions of available problem solvers, the goal and assumptions of tasks, and the properties of domains. In addition, this markup language must hold non-functional, pragmatics descriptions of available reasoning components. The language must be formal enough to express characteristics of reasoning components at the knowledge level, and it should also have a Web-compatible syntax, to allow the components to be distributed as Web resources. As explained in Section 3, the first achievement of the IBROW project was to develop the UPML language, which matches the requirements for brokering reasoning components on the Web.

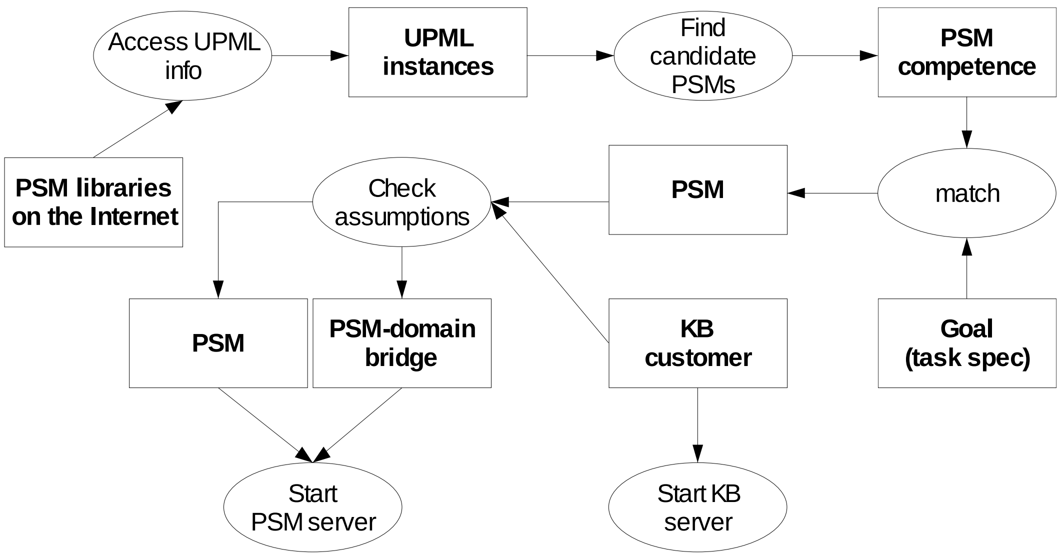


Figure 5.2 The brokering process in IBROW, in which the UPML markup language plays a central role.

Based on UPML, the IBROW broker can reason about component annotations to determine the localization of the components and select appropriate components by matching them to the users' requirements. Reasoning capabilities are required from the broker to recognize and analyze the users' problem, to find relevant problem solvers for each (sub)task of the problem and to check the applicability of problems solvers according to the knowledge bases available. Once the broker has selected a suitable set of knowledge components, it needs to adapt them and enable their integration into a single service. Finally, the broker needs to execute the configured running system to solve the users' task. Figure 5.2 shows a view of the brokering process: libraries of problem-solving methods marked-up with UPML and correspond to UPML instances, which are selected by the broker on the basis of their competence and the users’ task specification. Given the users’ domain model (shown as a ‘Customer KB’ in the figure), the broker selects the PSMs, which make proper assumptions about the domain model via special adaptation elements in UPML – PSM-Domain bridges. Finally, the broker passes the PSMs on a PSM server able to execute them.

Again, the UPML language for component markup, discussed in the next section, plays a kernel role at each stage of the brokering process.

3. UPML: The Language for Knowledge Component MarkUp

UPML is a software architecture specially designed to describe knowledge systems. The UPML architecture presented in Figure 5.3 consists of six different kinds of elements. A task defines the problem to be solved by the knowledge system. A problem-solving method defines the reasoning process used to solve the problem. A domain model defines the domain knowledge available to solve the problem. Each of these elements is described independently to enable reuse of: task descriptions in different domains, problem-solving methods for different tasks and domains, and domain knowledge for different tasks and problem-solving methods. Ontologies provide the terminology used in the tasks, problem-solving methods, and domain definitions. Again this separation enables knowledge sharing and reuse. For example, different tasks or problem-solving methods can share parts of the same vocabulary and definitions. Further elements of the specification are adapters, which are necessary to adjust other (reusable) elements to each other and to the specific application problem. UPML provides two types of adapters: bridges and refiners. Bridges explicitly model the relationships between two specific parts of the architecture, e.g. between a domain and a task or a task and a problem-solving method. Refiners can be used to express the stepwise adaptation of other elements of the specification, e.g. generic problem-solving methods and tasks can be refined to more specific ones by applying to them a sequence of refiners ([Fensel, 1997], [Fensel & Motta, to appear]). Again, separating the generic and specific parts of a reasoning process enhances reusability. The main distinction between bridges and refiners is that bridges change the input and output of the components to make them fit together. Refiners, by contrast, may change only the internal details, e.g. the subtasks of a problem-solving method. This provides UPML with a structured and principled approach for developing and refining heuristic reasoning components. This approach provides a three-dimensional design space where different types of adapters correspond to different types of moves in that space [Fensel, 2000] and [Fensel & Motta, to appear].

3.1. Overview of the UPML Framework

UPML relies on a meta-ontology that defines the concepts and relations which are then instantiated for specific knowledge components. As shown in Figure 5.4, this ontology is based on a root class *Entity*, which defines that each UPML *concept* and relation is represented with a list of attributes of a certain type. The root ontology of UPML defines two basic classes: *concept* and *binary relation*, both of which are subclasses of the *Entity class*. All UPML concepts and relations are subclasses of these two root classes. The hierarchy of classes used to define UPML is also presented in Figure 5.4. The main classes of this hierarchy are discussed in the remainder of this section. The concepts of UPML define parts of a problem-solving system, as described in Section 3.2. *Binary Relations* specify the interactions between the concepts as described in Section 3.3. *Architectural constraints and design guidelines* for UPML, which make it a full-fledged software architecture, are described in [Fensel & Groenboom, 1999] and [Fensel et al., to appear]. The Web syntaxes for UPML is discussed in Section 3.4.

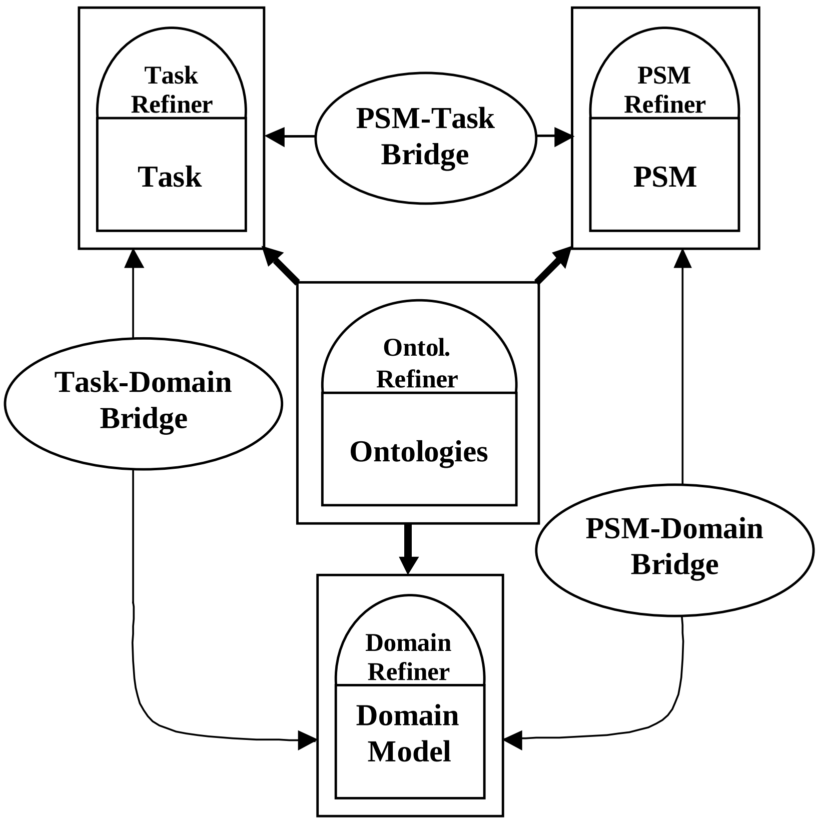


Figure 5.3 The UPML architecture.

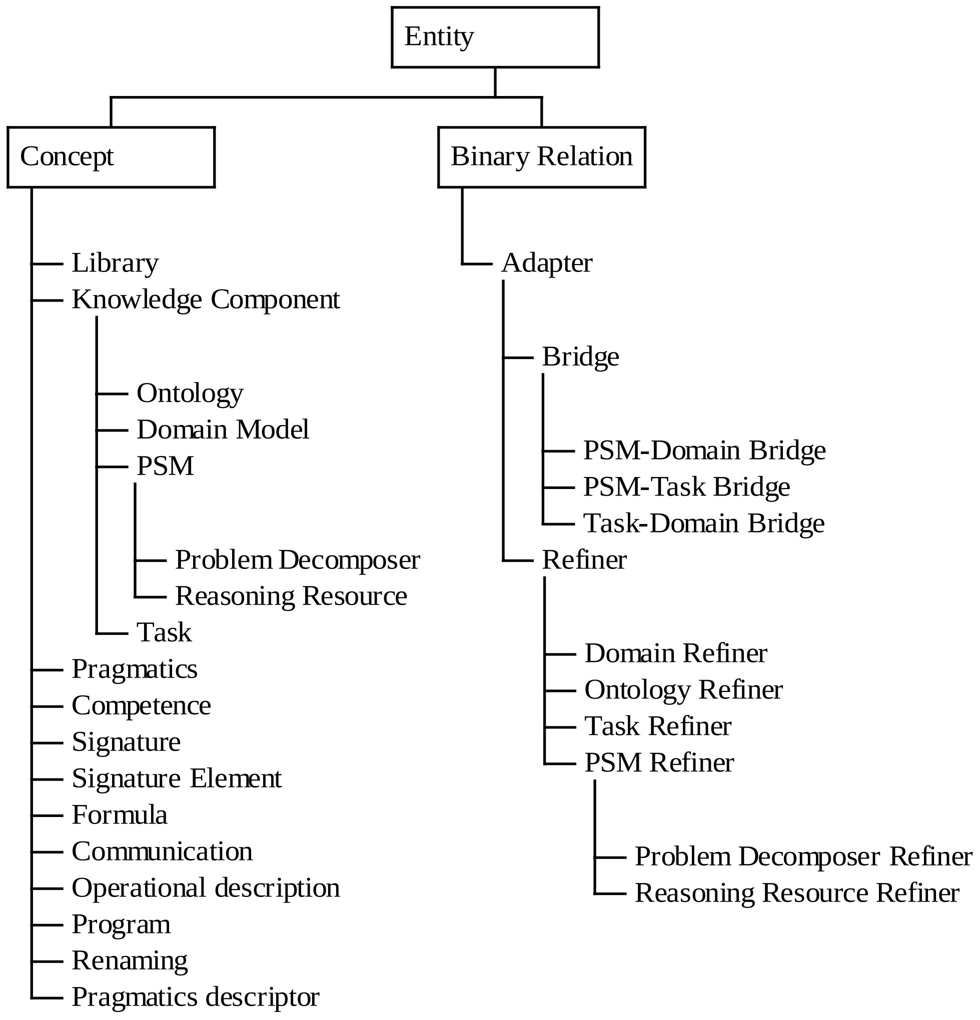


Figure 5.4 Class hierarchy of UPML. Each concept or relation in the UPML ontology is derived from the root ontology entities represented by the boxes.

3.2. Concepts

The *Library* concept is the overarching concept of UPML architecture: it contains a pointer to each component of a UPML specification (Figure 5.5). The subclass-of relationship between two entities is denoted by the symbol <. Each concept or relation is represented as a list of attribute-type pairs (attribute : type), where the type is either a primitive type (STRING), or a class of the hierarchy. The brackets around the types denote that the corresponding attribute can have multiple values.

Library < Concept
pragmatics : (Pragmatics)
tasks : (Task)
domain models : (Domain Model)
problem decomposers : (Problem Decomposer)
reasoning resources : (Reasoning Resource)
ontologies : (Ontology)
ontology refiners : (Ontology Refiner)
problem decomposer refiners : (Problem Decomposer Refiner)
reasoning resource refiners : (Reasoning Resource Refiner)
task refiners : (Task Refiner)
domain refiners : (Domain Refiner)
psm-domain bridges : (PSM-Domain Bridge)
psm-task bridges : (PSM-Task Bridge)
task-domain bridges : (Task-Domain Bridge)

Figure 5.5 The *Library* concept.

There are four main types of knowledge components in UPML: *Ontology, Domain Model, Task,* and *PSM* (see Figure 5.6). All types of components are defined as subclasses of the root concept *Knowledge Component*, also presented in Figure 5.6. Each component has a pragmatics description and relies on one or more ontologies, which define its universe of discourse. All attributes represented in the figure model part-of relationship besides uses.[14]

An *Ontology* (cf. [Fensel, 2001]) is used in the definition of *tasks*, *PSMs*, and *domain models*. An ontology provides an explicit specification of a conceptualization, which can be reused and shared by multiple reasoning components. It enables the definition of reusable terminology used by all other components through a signature, theorems and axioms. Ontologies are the key instrument of interchange among knowledge components. The core of an ontology specification in UPML is its *signature* definition, which captures the ontological elements used by a component, or *signature elements*. *Signature elements* are expressed in a certain modeling language. An *ontology* also provides axioms which characterize logical properties of the *signature elements*. Additional *theorems* may list useful statements, which are implied by the axioms.

The *Task* concept specifies the task to be achieved by the PSMs of the library. The *input roles* and *output roles* together with the *competence* property define the input/output specification of the task. The *input roles* specify the input of case data and the *output roles* specify the output of case data. The *Competence* concept represents the functional input/output specification of the *Task* component. *Competence* includes *preconditions* restricting valid inputs and *postconditions* which describe the output of a method or task. The *assumptions* property of *Task* contains requirements regarding the knowledge used to define the goal. A *Task* can import and refine other tasks via its *uses* attribute.

The *Domain Model* concept introduces domain knowledge, merely the formulas used by the problem-solving methods and tasks. The *Domain Model* consists of three elements:

properties, *meta-knowledge*, and the domain *knowledge* itself. *Meta-knowledge* captures the implicit and explicit assumptions made in the domain model of the real world. Meta-knowledge is assumed to be true. In other words, it has not been proven or cannot be proven, and corresponds to our assumptions about the domain. Domain *knowledge* is the knowledge base of the domain necessary to define the task in a given application domain and to carry out the inference steps of the selected problem-solving method. *Knowledge* is specified under the assumption that the *meta-knowledge* is true. *Properties* (a synonym for theorems) can be derived from *domain knowledge*

Knowledge Component < Concept
Pragmatics : Pragmatics
Ontologies : (Ontology)
Ontology < Knowledge Component
Signature : Signature
Theorems : (Formula)
Axioms : (Formula)
Domain Model < Knowledge Component
uses : (Domain Model)
properties : (Formula)
metaknowledge : (Formula)
knowledge : (Formula)

Task < Knowledge Component
Uses : (Task)
Input roles : (Signature Element)
Output roles : (Signature Element)
Competence : Competence
Assumptions : (Formula)
PSM < Knowledge Component
Communication : Communication
Input roles : (Signature Element)
Output roles : (Signature Element)
Competence : Competence

Figure 5.6 Knowledge Components: Ontology, Domain Model, Task, and PSM.

The *PSM* component represents a problem-solving method, defined by its *competence* and *communication* properties. The *input roles* and *output roles* of the PSM specify its inputs and outputs, similarly to their function in the *Task* component. The PSM’s *communication* property describes its interaction protocol with its environment in particular with other (PSM) components.

The *PSM* concept has the following two subclasses: *Problem Decomposer* and *Reasoning Resource* presented in Figure 5.7. A *Problem Decomposer* decomposes a task to be solved into a set of *subtasks*. Its *Operational Description* specifies the control structure over the subtasks and internal data flow among the subtasks. A *Reasoning Resource* solves a primitive step (or subtask) of a problem provided by the *problem decomposer*. It specifies *assumptions* regarding the domain knowledge, which must be fulfilled in order to perform a primitive reasoning step. Its internal structure is usually not specified, as it is considered as an implementational aspect of no interest to the architectural specification of a problem-solving system. The *knowledge roles* attribute specifies the input of the (domain) knowledge to the reasoning resource.

Problem Decomposer < PSM
Subtasks : (Task)
operational description : Operational Description
Reasoning Resource < PSM
knowledge roles : (Signature Element)
assumptions : (Formula)

Figure 5.7 Problem Decomposer and Reasoning Resource.

Considering knowledge components as Web resources themselves, UPML defines the concept of *Pragmatics*, which holds attributes which describe practical and reference information about a component. *Pragmatics* attributes are derived mainly from the Dublin Core[15] metadata recommendation for annotating Web resources.

UPML does not commit to any logical language to express formulas about the knowledge components, or to any procedural language to describe the operational control of a problem decomposer. In several places, developers can extend UPML with additional concepts to hold the primitives of their languages of choice.

3.3. Binary Relations

Binary Relations specify the interactions between the UPML knowledge components. The root binary relation of UPML is *Adapter* (Figure 5.8). The *Adapter* connects two components, called *arguments*, through a set of *renaming* correspondences between the terms of both arguments. Similar to knowledge components, an *Adapter* holds *pragmatics* information and refers to specific *ontologies*. UPML introduces two subclasses of *Adapter*: a *Bridge* and a *Refiner* shown in Figure 5.9.

Adapter_< Binary Relation
argument1 : Knowledge Component
argument2 : Knowledge Component
pragmatics : Pragmatics
ontologies : (Ontology)
renamings : (Renaming)

Figure 5.8 Adapter.

The *Bridge* relation connects two *Knowledge Components* of different kinds. It defines *mapping axioms* and additional *assumptions* about the components it relates. The *Bridge* relation has three subrelations: a *PSM-Domain Bridge* connects a *PSM* with a *Domain Model*; a *PSM-Task Bridge* connects a *PSM* and a *Task*; and a *Task-Domain Bridge* connects a *Task* and a *Domain Model.*

Bridge < Adapter
uses : (Bridge)
mapping axioms : (Formula)
assumptions : (Formula)
Refiner < Adapter
In : Knowledge Component
Out : Knowledge Component

Figure 5.9 Bridge and Refiner

The *Refiner* relation connects two knowledge components of the same type and so expresses the stepwise adaptation of one component into the other. Very generic problem-solving methods and tasks can be refined to more specific ones by applying a sequence of refiners to them. A *Refiner* assumes that its two attributes *in* and *out* have the same type. This serves to ensure that the refiner modifies a given component, as opposed to mapping it to a different kind of component via the *Bridge* relation.

Each main UPML component has its own associated type of refiner. Consequently, the *Refiner* relation has four component-specific subrelations: a *Domain Refiner*, an *Ontology Refiner*, a *Task Refiner*, and a *PSM Refiner*. The definition of a refiner includes the attributes specific to each kind of component. Each refiner has its own restrictions on *in*-put and *out*-put: the *Domain Refiner* contains redefined *properties*, *metaknowledge* and *knowledge* in the refined component.

Similarly, the *Ontology Refiner* contains refined *signature*, *theorems*, and *axioms*. The *Task Refiner* refines *competence* and *assumptions*, and the *PSM Refiner* refines *communication* and *competence*.

The separation of generic and specific parts of a reasoning process maximizes reusability. UPML offers two ways of combining components of the same type. Both serve a similar purpose, however they provide complementary means. First, a component can import another component via the *uses* attribute. Hence, the component can make use of definitions imported from the other component, monotonically refine them, and extend them. In this case, the *uses* relationship is not modeled by an explicit entity in the UPML specification but rather via an attribute of an existing component (the one that imports another component). This first approach corresponds to a monotonic extension of a component. Second, a component can be defined as a refinement of another component via the *Refiner* relation. In this case, the former component can rewrite the aspects of the latter component via the *renamings* attribute. Also, in this case we model the *uses*-relationship by an explicit entity of the UPML specification (i.e., a *Refiner*). This second approach enables non-monotonic modification of a component via an explicit element of the architecture. As mentioned earlier, this provides UPML *with a structured and principled approach for developing and configuring heuristic* *reasoning components* by adapting and refining generic components (cf. [Fensel, 2000], [Fensel & Motta, to appear]).

3.4. Web Syntaxes

In this section we describe the Web syntaxes for UPML meta-ontology based on XML and RDF. A Web syntax is crucially important for UPML because UPML is posed as a standard for knowledge component markup on the Web. The syntax consists of three documents: an XML DTD definition for UPML, an XML Schema definition for UPML, and an RDF Schema definition for UPML as described in the next section.

3.4.1. XML Syntax

XML[16] is one of the Web standards which can be used to describe UPML. XML is a widely supported, domain-independent language for representing, storing, sharing, and exchanging data. It provides the means to mark-up the semantics of data, as well as to validate, and exchange data structures. XML documents consist of XML tags, which have their names and associated values. The tags can be nested into one another to represent the hierarchical structure of a document. This structure provides a mechanism to impose constraints on the storage layout and logical structure.

XML Schema[17] is a W3C standard aimed to specify the structure of XML documents. Parts of a document are specified with a set of data types, either primitive or complex, which can be inherited from one another. This inheritance of data structures allows explicit encoding of UPML structures as XML Schema structures.

Figure 5.10 shows a fragment of XML Schema syntax for the *Knowledge Component* concept.

```
 <xsd:complexType name="upml:Knowledge_component" base="upml:Concept">
 <xsd:element name="upml:pragmatics" type="upml:Pragmatics" minOccurs="1"
maxOccurs="1"/>
 <xsd:element name="upml:ontology" type="upml:Ontology" minOccurs="1"
maxOccurs="unbounded"/>
 </xsd:complexType>
```

Figure 5.10 Part of the XML Schema defining the XML representation of the *Knowledge Component* concept.

A relatively new standard, XML Schema is still not widely supported by industry. Hence, we also provide a *DTD* specification for UPML, part of which is presented in Figure 5.11. However, DTDs are only capable of representing the structure of document instances; they cannot capture the hierarchy of the UPML structures.

```
 <!ELEMENT upml:library (upml:pragmatics, upml:ontology+, upml:domain_models+,
upml:complex_psms+, upml:primitive_psms+, upml:tasks+, upml:ontology_refiners+,
upml:problem_decomposer_refiners+,
```

Figure 5.11 Part of the DTD for UPML defining the *Library* concept.

Both the full XML Schema and the full DTD for UPML are available from the UPML website[18].

3.4.2. RDF Syntax

RDF[19] is an upcoming standard for representing machine-processable semantics of on-line information resources. Unlike

XML, which enables serialization of trees, RDF provides extensive representation of options to perform knowledge-level markup. The foundation of RDF is a model for representing named properties and property values. RDF properties may be thought of as attributes of resources. In this respect, they correspond to traditional attribute-value pairs used in UPML. RDF properties also represent relationships between the resources. The RDF model can therefore resemble an entity-relation diagram.

The structure of RDF documents is specified with *RDF Schema* (RDFS) [Brickley & Guha, 2000]. RDF resources represent some components and correspond to the *Concept* class in the UPML hierarchy. Hence, in the RDF Schema of UPML we define *Entity* as a subclass of *rdfs:resource*, and *Concept* as a subclass of *Entity*. We define all other concepts of UPML as direct or indirect subclasses of the RDF class *Concept*.

RDF properties are conceptually equivalent to the UPML *Binary Relations*. Hence, the latter can be represented as properties. However, RDF Schema does not allow defining properties of properties. Accordingly, if we were to define *Binary Relation* as a subclass of *rdfs:property*, then we would have no way of describing the attributes of binary relations. Consequently, in the RDF Schema of UPML we defined the *Binary Relation* as a subclass of *Entity*.

As a result, we used RDF classes to define both the *Concepts* and the *Binary Relations*. Each attribute of *Concepts* and *Binary Relations* is defined as a property of the corresponding class in RDFS. Each property has the corresponding class as its domain, and the type of this attribute, as defined in the UPML specification, as its range.

The RDF syntax of UPML is generated from the Protégé-2000[20] knowledge-acquisition tool, which supports import and export of ontologies from and to RDF. For example, a sample of the RDFS specification for the *Knowledge Component* concept is shown in Figure 5.12, while the whole specification is available from the UPML website.

```
<s:Class rdf:about="&a;Knowledge_Component">
 <a:ROLE>Abstract</a:ROLE>
 <s:subClassOf rdf:resource="&a;Concept"/>
</s:Class>
<rdf:Property rdf:about="&a;pragmatics">
 <a:SLOT-MAXIMUM-CARDINALITY>1</a:SLOT-MAXIMUM-CARDINALITY>
 <s:domain rdf:resource="&a;Adapter"/>
 <s:domain rdf:resource="&a;Knowledge_Component"/>
 <s:range rdf:resource="&a;Pragmatics"/>
</rdf:Property>
<rdf:Property rdf:about="&a;ontologies">
 <s:domain rdf:resource="&a;Adapter"/>
 <s:domain rdf:resource="&a;Knowledge_Component"/>
 <s:domain rdf:resource="&a;Library"/>
 <s:range rdf:resource="&a;Ontology"/>
</rdf:Property>
```

Figure 5.12 A fragment of the RDF Schema for UPML.

The tool for specifying UPML components is discussed in the next section.

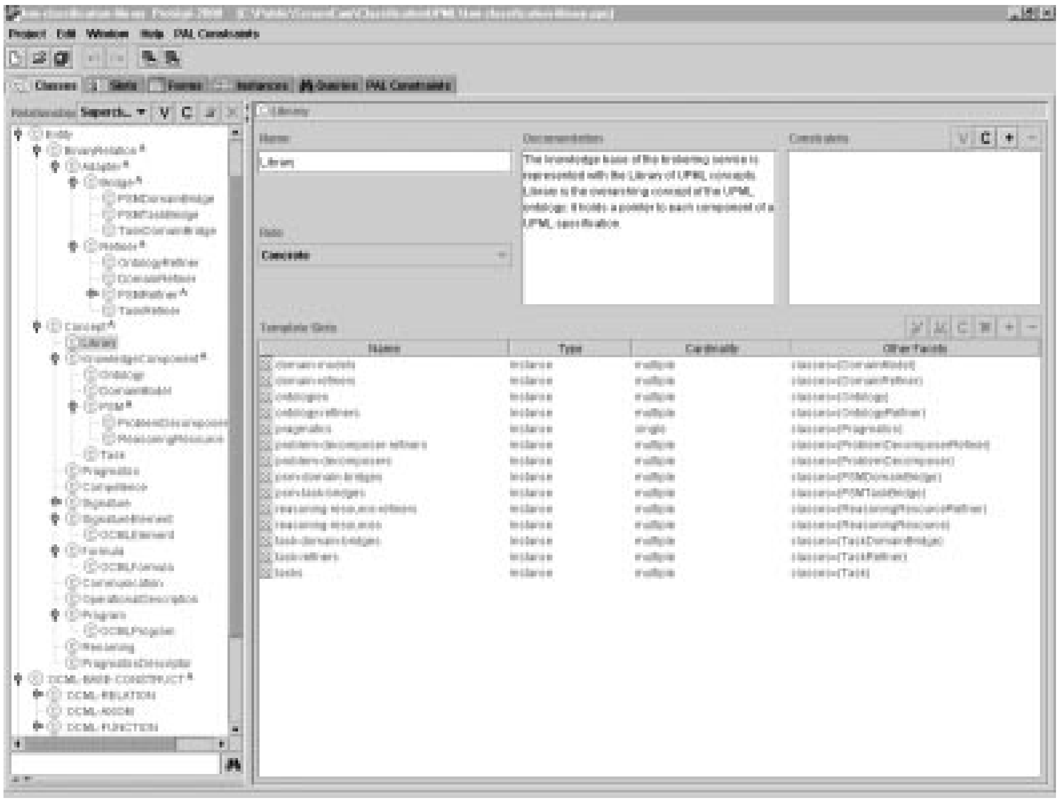
4. An Editor for UPML Specifications based on Protégé-2000

As described in the previous section, the UPML framework can be seen as an ontology composed of classes and relations describing reusable knowledge components. Instances of these classes and relations are particular *knowledge components* and *adapters* (e.g., a classification task and a heuristic classifier problem decomposer). To enable developers to specify and annotate libraries of knowledge components, we created an editor for UPML using Protégé-2000[21]. Protégé is an extensible ontology-editing and knowledge-acquisition environment assisting users in the construction of large electronic knowledge bases [Grosso et al., 1999]. Protégé-2000 allows users to create, browse, and edit domain ontologies using a frame-based representation, compliant with the OKBC knowledge model [Chaudhri et al., 1998].

In Protégé-2000, an ontology is represented with a multiple-inheritance hierarchy of the classes of concepts which are important in a domain. Slots are attached to these classes and define their attributes. Facets restrict the type of value that a slot can take. Protégé automatically generates a graphical knowledge-acquisition tool from the ontology, which enables application specialists to enter the detailed content knowledge required to define specific applications [Puerta et al., 1992]. Protégé allows developers to customtailor this knowledge-acquisition tool directly by configuring graphical entities on forms, which are attached to each class in the ontology for the acquisition of instances (particular exemplars of the classes). Consequently, the application specialists can enter domain information by filling in the blanks of intuitive forms and by drawing diagrams composed of selectable icons and connectors. Protégé-2000 is able to store the knowledge bases in several formats, including RDF [Noy et al., to appear].

We modeled the set of concepts and relationships of UPML as a hierarchy of classes in Protégé-2000, with slots and facets attached to them. Both concepts and binary relations in UPML are reified as classes in Protégé, so they can have attributes, and be subclassed.

Figure 5.13 shows most of the hierarchy of the classes we used to model UPML and the definition of the class Library. The UPML ontology in Protégé reflects the fact that UPML does not commit to any logical or procedural language to express the formulas and programs that define knowledge components and adapters. By means of the ontology-extension and ontology inclusion mechanisms of Protégé, developers can extend the UPML ontology with the primitives necessary to write expressions in their object language of choice. For example, we recently used the UPML editor to specify a library of classification problem solving components [Motta & Lu, 2000]. The components of the library are coded using the logical and operational OCML language [Motta, 1999]. First, we modeled the OCML set of basic primitives (such as classes, relations, axioms and functions) themselves as a meta-ontology in Protégé-2000. We then included this ontology in the UPML editor (as partially shown in the lower left part of Figure 5.13). From there, we were able to extend (subclass) the UPML concepts *Signature*, *Signature Element*, *Formula* and *Program* with the definitions which refer to the OCML primitives, as presented in Figure 5.13.



Name: Library

Documentation: The knowledge base of the brokering service is represented with the Library of UPML concepts. Library is the overarching concept of the UPML ontology. It holds a pointer to each component of a UPML specification.

Role: Concrete

Template Slots

Name	Type	Cardinality	Other Facets
domain-models	Instance	multiple	classes={DomainModel}
domain-refiners	Instance	multiple	classes={DomainRefiner}
ontologies	Instance	multiple	classes={Ontology}
ontology-refiners	Instance	multiple	classes={OntologyRefiner}
pragmatics	Instance	single	classes={Pragmatics}
problem-decomposer-refiners	Instance	multiple	classes={ProblemDecomposerRefiner}
problem-decomposers	Instance	multiple	classes={ProblemDecomposer}
psm-domain-bridges	Instance	multiple	classes={PSMDomainBridge}
psm-task-bridges	Instance	multiple	classes={PSMTaskBridge}
reasoning-resource-refiners	Instance	multiple	classes={ReasoningResourceRefiner}
reasoning-resources	Instance	multiple	classes={ReasoningResource}
task-domain-bridges	Instance	multiple	classes={TaskDomainBridge}
task-refiners	Instance	multiple	classes={TaskRefiner}
tasks	Instance	multiple	classes={Task}

Figure 5.13 A snapshot of the Protégé-based UPML editor: The ontology of UPML modelled in Protégé as a hierarchy of classes, extended with OCML-specific classes (left panel) and the definition of the Library concept, which holds a pointer to every component in the architecture (right panel).

Given this model of UPML, Protégé-2000 automatically generated an RDF Schema representation holding the UPML classes and properties for annotating reasoning resources. Based on the UPML ontology, Protégé-2000 also generated a graphical editor for instantiating specific UPML specifications, e.g., the components of the library for classification problem-solving. We then custom-tailored this editor to center the knowledge-acquisition process on the use of diagrammatic metaphors. In particular, we defined specific kinds of diagrams to enter the task decomposition (*inference structure*) of a problem decomposer and the control regime of its corresponding operational description (*control structure*).

As shown in Figure 5.14, the UPML editor makes it possible to browse the list of instances of a selected class and to view and edit the knowledge-acquisition form associated to the selected instance. In this figure, the editor displays the ‘heuristic optimal solution classifier’ instance of the *Problem Decomposer* class. The knowledge-acquisition form for a *problem decomposer* PSM contains a number of user interface components to enter the values of the slots defined for the *Problem Decomposer* class. For example, this form contains a sub-form specifying the *operational description* slot of the *PSM* (on the right). This sub-form includes an inference structure diagram, which helps users to specify the competence slots of the *PSM* (*input roles*, *output roles* and *subtasks*) by directly drawing nodes and links in the diagram, which, in turn, automatically creates and fills in the corresponding instances of the *Signature Element* and the *Task* classes.

Once created, the specification of a set of UPML knowledge components (instances) can be exported as a corresponding set of RDF statements, which refer to the RDF Schema of UPML. When Protégé-2000 generates an RDF specification from an ontology, it resolves the differences between the knowledge models of Protégé and RDF Schema by adding specific RDF statements to express facets such as cardinality constraints on slots, or multiple domains or ranges for a given slot. Consequently, the resulting RDF code for UPML contains the complete translation of the UPML ontology; however, some parts of the ontology are only understandable by Protégé-2000 (see [Noy et al., to appear] for a discussion).

As a result, the UPML editor in Protégé-2000 provides a guided framework, which helps developers to define the UPML specifications and to annotate the knowledge components, which they want to publish on the Web. Protégé-2000 is extensible through its API (application programming interface) [Musen et al., 2000]. We envisage enhancing the UPML editor with services which will help the users to configure the reasoning resources and problem decomposers for different domains and tasks in connection to the IBROW broker. The resulting configurations of problem solvers will help augment the UPML specification of the libraries available with the appropriate bridges and refiner components.

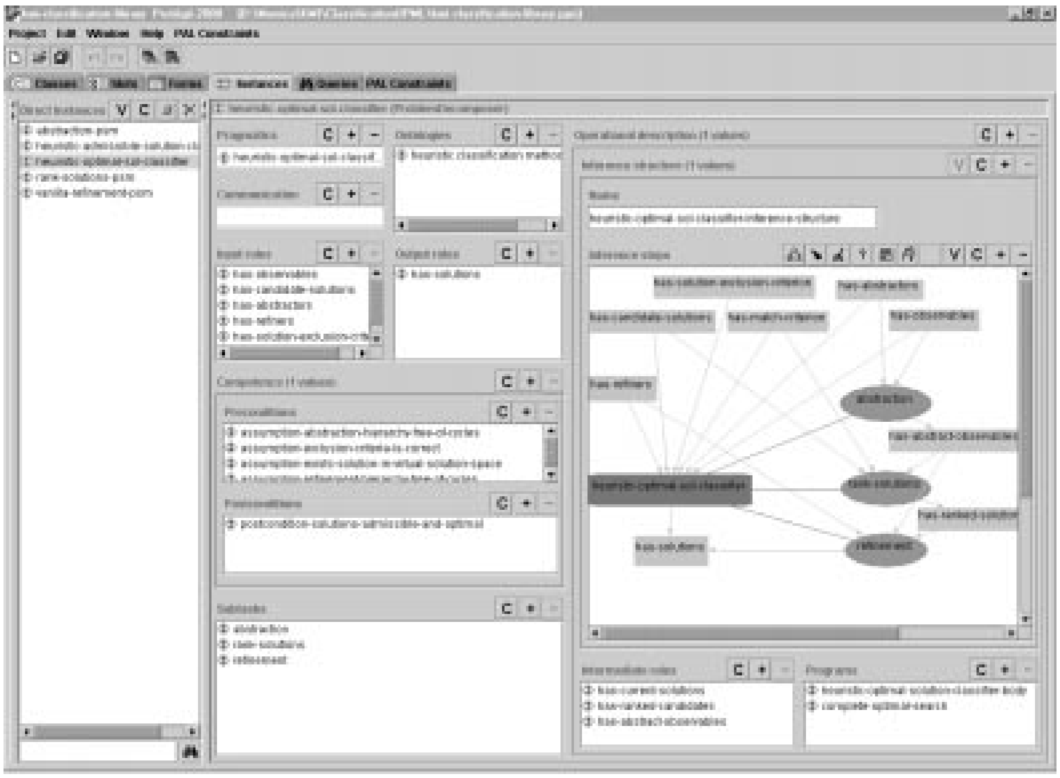


Figure 5.14 A snapshot of the Protégé-based UPML editor: the 'heuristic optimal solution classifier' instance of the Problem-Decomposer class, with its inference structure diagram on the right.

Conclusions

UPML defines an *architecture* for describing heuristic reasoning components (cf. [Berners-Lee & Fischetti, 1999] for the vision of the latter) for the Semantic Web. UPML is language-neutral in the sense that different formal languages can be plugged in to describe the elementary slots defined by UPML. We already used order-sorted logic [Fensel et al., 1998], frame logic [Kifer et al., 1995], and OCML [Motta, 1999] successfully for adding formal semantics to UPML specifications (cf. [Fensel et al., to appear]). We also tried to use the OIL language [Fensel et al., 2001] which has been developed within the Ontoknowledge[22] project (cf. [Fensel et al., 2000(b)]) as a Web-based ontology language. OIL was a significant source of inspiration for the ontology language called DAML+OIL[23] developed in a joined EU/US working group on language standardization, which is currently present as a W3C working group on the Semantic Web. However in the case of OIL, the results achieved were somewhat disappointing (cf. [Fensel et al., 2000(a)]). OIL does not provide an adequate expressive power for many of the axiomatic parts of UPML specifications. Extending the expressive power of OIL seems to be absolutely necessary for making it usable in this context.

In general, UPML is concerned with describing the dynamic reasoning aspect of the Semantic Web. Therefore, it defines a new layer on top of the currently developed language standards for the Semantic Web. This layer provides machine-processable semantics for the dynamic information sources of the Semantic Web (see Chapter 1 for relevant discussion).

UPML is a description language and does not require operational semantics. However, the IBROW broker must be able to reason about the expressions in the description language. In that sense, one can view the broker as a special-purpose ‘interpreter’ of the language. The current priority in the IBROW project is to use UPML for annotating large libraries of problems-solving methods and implementing UPML-based reasoning services to enable their intelligent brokering on the Web. Here we can employ many concepts of the component retrieval area developed for software engineering. The use of formal techniques for software component retrieval is discussed in [Jeng & Cheng, 1992], [Jeng & Cheng, 1995], [Penix & Alexander, 1995], [Chen & Cheng, 1997], [Jiliani et al., 1997], [Mili, 1997], [Mili et al., 1997], [Schuman & Fischer, 1997], [Penix et al., 1997], and [Zaremski & Wing, 1997].

In many aspects the RETSINA[24] project (cf. [Sycara et al., 2001]) is similar to IBROW. Although it is focused on multi-agent systems, this project deals with aspects similar to those we encounter in IBROW. Heterogeneous agents must be able to communicate with each other by means of a common language. The language needs to be coordinated effectively across distributed networks of information. An agent capability description language called LARKS (*Language for Advertisement and Request for Knowledge Sharing*) has been developed (cf. [Sycara et al., 2001]) addressing the problem of agent interoperability. This common language is used by middle or matchmaking agents to pair service-requesting agents with service-providing agents, which meet the requesting agents' requirements. The matching engine of the matchmaker agent contains five different filters for context matching, word frequency profile comparison, similarity matching, signature matching, and constraint matching. The user configures these filters to achieve the desired tradeoff between performance and matching quality. The main differences between UPML and LARKS are:

- UPML defines a richer architecture for describing the reasoning components compared to LARKS.
- UPML is a full-fledged methodological framework for developing Web-enabled libraries of such components.
- LARKS fixes the language used to describe the competence of these components, whereas UPML ‘merely’ provides an architecture in which several languages can be plugged into.

Finally, one has to admit that the actual retrieving component in RETSINA is far more developed than the current brokering support in IBROW. This also indicates the main direction for further development in IBROW. In particular, we foresee that the scope of the knowledge components to be brokered in IBROW will need to be broadened from traditional problem-solving methods to agent-based components capable of performing reasoning steps autonomously. Both UPML and the broker will need to take into account these new kinds of components [Abasolo et al., 2001].

During the early phase of the IBROW project, we demonstrated how our brokering approach based on UPML could be used for the toy problem of classifying apples for users[25]. The broker used an approach based on Prolog and CORBA to localize, compose, and integrate the heterogeneous components of the configured system, such as knowledge bases of the apple domain and classification problem-solving methods [Benjamins et al., 1999]. The results of this experiment appeared very promising and form the foundations for the work that we have planned for the next phases of the IBROW project. Consequently, we are expecting even more promising results from this stage, results that will help to make the vision of the Semantic Web a reality.

References

[Abasolo et al., 2001] C. Abasolo, J.-L. Arcos, E. Armengol, M. Gómez, J.-M. López-Cobo, M. López-Sánchez, R. López de Màntaras, E. Plaza, C. van Aart, and B. Wielinga: Libraries for Information Agents, In: IBROW Project IST-1999-19005: An Intelligent Brokering Service for Knowledge-Component Reuse on the World-Wide Web, Deliverable 4, 2001.

[Benjamins & Fensel, 1998] V. R. Benjamins and D. Fensel: Special issue on problem-solving methods, International Journal of Human-Computer Studies, 49(4), 1998.

[Benjamins & Shadbolt, 1998] V. R. Benjamins and N. Shadbolt: Special Issue on Knowledge Acquisition and Planning, International Journal of Human-Computer Studies, 48(4), 1998.

[Benjamins et al., 1999] V. R. Benjamins, B. Wielinga, J. Wielemaker, and D. Fensel: Brokering Problem-Solving Knowledge at the Internet. In: D. Fensel et al. (eds.), Knowledge Acquisition, Modeling, and Management, Proceedings of the European Knowledge Acquisition Workshop (EKAW-99), LNAI 1621, Springer-Verlag, May, 1999.

[Benjamins, 1995] V. R. Benjamins: Problem Solving Methods for Diagnosis And Their Role in Knowledge Acquisition, International Journal of Expert Systems: Research and Application, 8(2):93-120, 1995.

[Berners-Lee & Fischetti, 1999] T. Berners-Lee and M. Fischetti: Weaving the Web, Harper, San Francisco, 1999.

[Breuker & van de Velde, 1994] J. Breuker and W. van de Velde (eds.): The CommonKADS Library for Expertise Modeling, IOS Press, Amsterdam, The Netherlands, 1994.

[Brickley & Guha, 2000] D. Brickley and R. Guha: Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation, March, 2000; available online at http://www.w3.org/TR/rdf-schema

[Chandrasekaran et al., 1992] B. Chandrasekaran, T. Johnson, and J. Smith: Task Structure Analysis for Knowledge Modeling, Communications of the ACM, 35(9):124-137, 1992.

[Chaudhri et al., 1998] V. Chaudhri, A. Farquhar, R. Fikes, P. Karp, and J. Rice: OKBC: A programmatic foundation for knowledge base interoperability. In: Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and of the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98), AAAI Press, 1998, p. 600–607.

[Chen & Cheng, 1997] Y. Chen and B. Cheng: Facilitating an Automated Approach to Architecture-based Software Reuse. In Proceedings of the 12th IEEE International Conference on Automated Software Engineering (ASEC-97), Incline Village, Nevada, November 3-5, 1997.

[Fensel, 1995] D. Fensel: Formal Specification Languages in Knowledge and Software Engineering, The Knowledge Engineering Review, 10(4), 1995.

[Fensel, 1997] D. Fensel: The Tower-of-Adapter Method for Developing and Reusing Problem-Solving Methods. In: E. Plaza et al. (eds.), Knowledge Acquisition, Modeling and Management, LNAI 1319, Springer-Verlag, 1997.

[Fensel, 2000] D. Fensel: Problem-Solving Methods: Understanding, Description, Development, and Reuse, LNAI 1791, Springer-Verlag, 2000.

[Fensel, 2001] D. Fensel: Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce, Springer-Verlag, 2001.

[Fensel & Benjamins, 1998] D. Fensel and V. Benjamins: Key Issues for Problem-Solving Methods Reuse. In: Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98), Brighton, UK, August 23-28, 1998, p. 63-67.

[Fensel & Groenboom, 1999] D. Fensel and R. Groenboom: An Architecture for Knowledge-Based Systems, The Knowledge Engineering Review, 14(3):153-173, 1999.

[Fensel & Motta, to appear] D. Fensel and E. Motta: Structured Development of Problem Solving Methods, IEEE Transactions on Knowledge and Data Engineering, to appear; available online at http://www.cs.vu.nl/~dieter/pub.html

[Fensel & van Harmelen, 1994] D. Fensel and F. van Harmelen: A Comparison of Languages which Operationalize and Formalize KADS Models of Expertise, The Knowledge Engineering Review, 9(2), 1994.

[Fensel et al., 1998] D. Fensel, R. Groenboom, and G. Renardel de Lavalette: Modal Change Logic (MCL): Specifying the Reasoning of Knowledge-based Systems, Data and Knowledge Engineering, 26(3):243-269, 1998.

[Fensel et al., 1999(a)] D. Fensel, V. R. Benjamins, S. Decker, M. Gaspari, R. Groenboom, W. Grosso, M. Musen, E. Plaza, G. Schreiber, R. Studer, and Bob Wielinga: The Unified Problem-Solving Method Development Language UPML. In: IBROW3 ESPRIT Project 27169: An Intelligent Brokering Service for Knowledge-Component Reuse on the World-Wide Web, Deliverable 1.1, 1999.

[Fensel et al., 1999(b)] D. Fensel, V. Benjamins, E. Motta, and B. Wielinga: UPML: A Framework for knowledge system reuse. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99), Stockholm, Sweden, July 31 - August 5, 1999.

[Fensel et al., 2000(a)] D. Fensel, M. Crubézy, F. van Harmelen, and I. Horrocks: OIL & UPML: A Unifying Framework for the Knowledge Web. In Proceedings of the Workshop on Applications of Ontologies and Problem-solving Methods, 14th European Conference on Artificial Intelligence ECAI'00, Berlin, Germany August 20-25, 2000.

[Fensel et al., 2000(b)] D. Fensel, F. van Harmelen, H. Akkermans, M. Klein, J. Broekstra, C. Fluyt, J. van der Meer, H.-P. Schnurr, R. Studer, J. Davies, J. Hughes, U. Krohn, R. Engels, B. Bremdahl, F. Ygge, U. Reimer, and I. Horrocks: OnToKnowledge: Ontology-based Tools for Knowledge Management. In: Proceedings of the eBusiness and eWork 2000 Conference (EMMSEC 2000), Madrid, Spain, October 18-20, 2000.

[Fensel et al., 2001] D. Fensel, I. Horrocks, F. van Harmelen, D. McGuiness, and P. Patel-Schneider: OIL: Ontology Infrastructure to Enable the Semantic Web, IEEE Intelligent Systems, March/April, 2001.

[Fensel et al., to appear] D. Fensel, E. Motta, V. R. Benjamins, M. Crubézy, S. Decker, M. Gaspari, R. Groenboom, W. Grosso, F. van Harmelen, M. Musen, E. Plaza, G. Schreiber, R. Studer, and B. Wielinga: The Unified Problem-solving Method Development Language UPML, to appear in Knowledge and Informational Systems (KAIS); available online at http://www.cs.vu.nl/~dieter/pub.html

[Gomez Perez & Benjamins, 1999] A. Gomez Perez and V. R. Benjamins: Applications of ontologies and problem-solving methods. AI Magazine 20(1):119– 122, 1999.

[Grosso et al., 1999] W. Grosso, H. Eriksson, R. Fergerson, J. Gennari, S. Tu, and M. Musen: Knowledge Modeling at the Millennium (The Design and Evolution of Protégé-2000). In: Proceedings of the Twelfth Workshop on Knowledge Acquisition, Modeling and Management (KAW99), Banff, Alberta, Canada, October 16-21, 1999.

[Gruber, 1993] T. Gruber: Towards Principles for the Design of Ontologies Used for Knowledge Sharing, In: N. Guarino and R. Poli (eds.), Formal Ontology in Conceptual Analysis and Knowledge Representation, Kluwer Academic Publishers, Deventer, The Netherlands, 1993.

[Jeng & Cheng, 1992] J.-J. Jeng and B. H. Cheng: Using Automated Reasoning Techniques to Determine Software Reuse, International Journal of Software Engineering and Knowledge Engineering, 2(4):523-546, 1992.

[Jeng & Cheng, 1995] J.-J. Jeng and B. H. Cheng: Specification Matching for Software Reuse: A Foundation. In: Proceedings of the ACM Symposium on Software Reuse, Seattle, Washington, April, 1995, pp. 97-105.

[Jiliani et al., 1997] L. Jilani, J. Desharnais, M. Frappier, R. Mili, and A. Mili: Retrieving Software Components That Minimize Adaptation Effort. In: Proceedings of the 12th IEEE International Conference on Automated Software Engineering (ASEC-97), Incline Village, Nevada, November 3-5, 1997.

[Kifer et al., 1995] M. Kifer, G. Lausen, and J. Wu: Logical Foundations of Object-Oriented and Frame-Based Languages, Journal of the ACM, 42(4):741-843, 1995.

[Marcus, 1988] S. Marcus (ed.): Automating Knowledge Acquisition for Experts Systems, Kluwer Academic Publisher, Boston, 1988.

[Mili et al., 1997] R Mili, A. Mili, and R. Mittermeir: Storing and Retrieving Software Components: A Refinement Based System, IEEE Transactions on Software Engineering, 23(7):445-460, 1997.

[Mili, 1997] F. Mili: Transformational Based Problem Solving Reuse. In Proceedings of the 9th International Conference on Software Engineering & Knowledge Engineering (SEKE-97), Madrid, Spain, June 18-20, 1997.

[Motta & Lu, 2000] E. Motta and W. Lu: A Library of Components for Classification Problem Solving. In: Proceedings of the 2000 Pacific Rim Knowledge Acquisition Workshop, Sydney, Australia, December 11-13, 2000.

[Motta et al., 1998] E. Motta, M. Gaspari, and D. Fensel: UPML Specification of a Parametric Design Library, In: IBROW3 ESPRIT Project 27169: An Intelligent Brokering Service for Knowledge-Component Reuse on the World-Wide Web, Deliverable D4.1, 1998.

[Motta, 1999] E. Motta: Reusable Components for Knowledge Modeling, IOS Press, Amsterdam, 1999.

[Musen 1998] M. Musen: Modern Architectures for Intelligent Systems: Reusable Ontologies and Problem-Solving Methods. In: C.G. Chute(ed.), 1998 AMIA Annual Symposium, Orlando, FL, 1998, p. 46-52.

[Musen et al., 2000] M. Musen, R. Fergerson, W. Grosso, N. Noy, M. Crubezy, and J. Gennari. Component-Based Support for Building Knowledge-Acquisition Systems. In Proceedings of the Conference on Intelligent Information Processing (IIP 2000) of the International Federation for Information Processing World Computer Congress (WCC 2000), Beijing, 2000.

[Noy et al., to appear] N. Noy, M. Sintek, S. Decker, M. Crubézy, R. Fergerson, M. Musen: One Size Does Fit All: Acquiring Semantic Web Contents with Protégé-2000. IEEE Intelligent Systems, Special issue on Semantic Web Technology, to appear.

[Omelayenko et al., 2000] B. Omelayenko, M. Crubézy, D. Fensel, Y. Ding, E. Motta, and M. Musen: Meta Data and UPML, In: IBROW Project IST-1999-19005: An Intelligent Brokering Service for Knowledge-Component Reuse on the World-Wide Web, Deliverable 5; available online at: http://www.cs.vu.nl/~upml/

[Penix & Alexander, 1995] J. Penix and P. Alexander: Design Representation for Automating Software Component Reuse. In Proceedings of the First International Workshop on Knowledge-Based Systems for the (Re)use of Program Libraries, Sophia Antipolis, France, November 23-24, 1995.

[Penix et al., 1997] J. Penix, P. Alexander, and K. Havelund: Declarative Specification of Software Architectures. In Proceedings of the 12th IEEE International Conference on Automated Software Engineering (ASEC-97), Incline Village, Nevada, November 3-5, 1997.

[Puerta et al., 1992] A. Puerta, J. Egar, S. Tu, and M. Musen: A Multiple-method Knowledge-Acquisition Shell for the Automatic Generation of Knowledge-acquisition Tools, Knowledge Acquisition, 4(2):171-196, 1992.

[Puppe, 1993] F. Puppe: Systematic Introduction to Expert Systems: Knowledge Representation and Problem-Solving Methods, Springer-Verlag, Berlin, 1993.

[Schreiber et al., 1994] A. Schreiber, B. Wielinga, J. Akkermans, W. van de Velde, and R. de Hoog: CommonKADS. A Comprehensive Methodology for KBS Development, IEEE Expert, 9(6):28-37, 1994.

[Schuman & Fischer, 1997] J. Schuman and B. Fischer: NORA/HAMMER: Making Deduction-Based Software Component Retrieval Practical. In: Proceedings of the 12th IEEE International Conference on Automated Software Engineering (ASEC-97), Incline Village, Nevada, November 3-5, 1997.

[Shaw & Garlan, 1996] M. Shaw and D. Garlan: Software Architectures. Perspectives on an Emerging Discipline, Prentice-Hall, 1996.

[Stefik, 1995] M. Stefik: Introduction to Knowledge Systems, Morgan Kaufman Publ., San Francisco, 1995.

[Sycara et al., 2001] K. Sycara, M. Paolucci, M. van Velsen, and J. Giampapa: The RETSINA MAS Infrastructure. In: Robotics Institute Technical Report #CMU-RI-TR-01-05, 2001; available online at http://www.cs.cmu.edu/~softagents/publications.html

[Zaremski & Wing, 1997] A. Zaremski and J. Wing: Specification Matching of Software Components, ACM Transactions on Software Engineering and Methodology, 6(4):335-369, 1997.

Footnotes

[1]Vrije Universiteit Amsterdam, Division of Mathematics and Computer Science, De Boelelaan 1081a, 1081 HV Amsterdam, the Netherlands, {borys, dieter, ying}@cs.vu.nl

[2]Stanford University, Stanford Medical Informatics, 251 Campus Dr., Suite 215, Stanford, CA 94305-5479, USA, {crubezy, musen}@smi.stanford.edu

[3]Intelligent Software Components, S.A., iSOCO Madrid, C. Hernandez de Tejada 7, 1st floor, 28027 Madrid, Spain, richard@isoco.com

[4]University of Amsterdam, Department of Social Science Informatics (SWI), Roetersstraat 15, 1018 WB Amsterdam, The Netherlands, bob@swi.psy.uva.nl

[5]The Open University, Knowledge Media Institute, Walton Hal, lMK7 6AA, Milton Keynes, United Kingdom, e.motta@open.ac.uk

[6]http://www.swi.psy.uva.nl/projects/ibrow/home.html

[7]As such, PSMs are a special type of software architectures ([Shaw & Garlan, 1996]).

[8]www.cs.vu.nl/~upml/. The recent version is described in [Omelayenko et al., 2000]

[9]http://webonto.open.ac.uk

[10]www.metacrawler.com

[11]www.isi.edu/ariadne

[12]www.aifb.uni-karlsruhe.de/Projekte/ontobroker/inhalt_en.html

[13]Note that we distinguish this from automatically generated Web pages, which are called dynamic opposite to static HTML pages.

[14] Uses is a very important attribute that will be explained later on.

[15]http://dublincore.org/

[16]www.w3c.org/xml

[17]www.w3.org/XML/Schema

[18]www.cs.vu.nl/~upml

[19]www.w3c.org/rdf

[20]See the next section for a detail description

[21]http://protege.stanford.edu/

[22]www.ontoknowledge.org

[23]http://www.cs.man.ac.uk/~horrocks/DAML-OIL/

[24]http://www.cs.cmu.edu/~softagents/retsina.html

[25]http://www.swi.psy.uva.nl/projects/ibrow/

6. Ontologies Come of Age

Deborah L. McGuinness[1]

1. Introduction: The web's growing needs

We may be poised for the next major evolution of online environments. In the early days of the web, HTML pages were generated by hand. The pages contained information about how to present information on a page. Early adopters took to the web quickly since it provided a convenient method for information sharing. Arguably, the generation of tools for machine generation and management of web pages allowed the web to really take off. Tool platforms allowed non-technical people to generate and publish web pages quickly and easily. The resulting pages typically included content and display information and targeted human readers (rather than targeting programs or automatic readers).

The web continues to grow at an astounding rate with web pages ubiquitously integrated into many aspects of business and personal life. However, web pages still preserve much of their character of being aimed at human consumption. Thus, applications such as search still require humans to review results pages in order to find the right answer to their queries. While search engine advances such as Google [Google, 2000] improve the situation, most people agree that finding the exact information one is seeking on the web today is not as easy as one would hope. One reason for this is that answers to search queries typically are a rank ordered list of pages that may contain the answer to the query. The answers rarely are just the portion of the page that the search engine "thought" contained the answer to the query. Additionally, web pages typically do not contain markup information about the contents of the page. If pages were marked up with information concerning what information or services could be obtained (and how that information or service could be obtained), then a page could be used more effectively by programs to return the portion of the page (or the answer from a service) that contains a specific answer to a question. Once web pages are aimed for machine or program consumption, instead of human consumption, the next generation of the web can be realized. The proliferation of markup languages aimed at marking up content and services instead of just presentation information can be viewed as support for this position. Markup languages such as XML [XML 2000], RDF [Lassila 1998, Lassila-Swick, 1999], RDFS [Brickley-Guha, 2000], DAML [Hendler-McGuinness, 2000], etc are becoming more accepted as users and application developers see the need for more understanding of what is available from web pages.

The view presented in this paper is consistent with the vision being put forward by Tim Berners-Lee of the W3C consortium. In a widely cited presentation [Berners-Lee, 2000] at XML 2000 conference, Berners-Lee presented his vision of the semantic web as being machine processable. We support this view as well. We believe that the next web evolution requires machines to understand the content of pages – both what can be obtained from pages and what that information means. Markup languages allow specification of this information. Berners-Lee offered an architecture diagram[2] in his presentation that provides a nice foundation. We include it here in Figure 6.1.

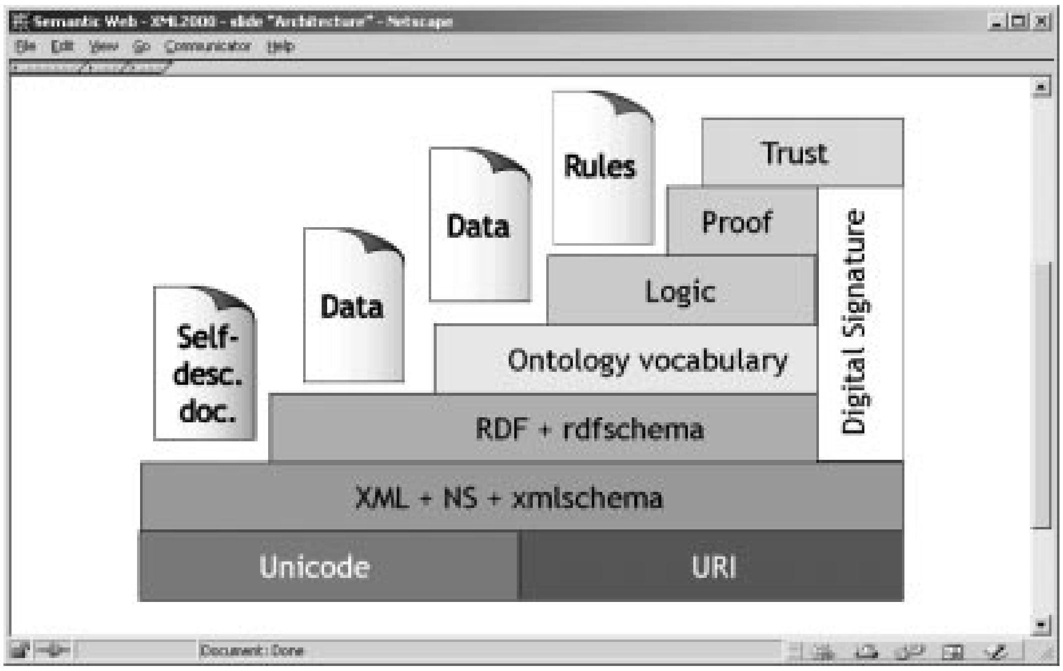


Figure 6.1 Berners-Lee’s Architecture

He shows the markup languages at the base (just above Unicode) for use in term specification (or in web speak, "resource" definition). The next layer and the one we will consider here, is the ontology layer. In this layer, we can define terms and their relationships to other terms. The next layer is the logic layer. In this layer, we can deduce information, thereby allowing us to deduce implications of the term definitions and relationships. In the rest of this paper, we will discuss the ontology and logic layers, what they have come to mean on the web, and how one might generate ontologies and use them in applications.

2. Ontologies

The term ontology has been in use for many years. Merriam Webster (*http://www.m-w.com/home.htm*), for example, dates ontology circa 1721 and provides two definitions (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. Mathematical or formal ontologies have also been written about for many years. Smith [Smith, 1998] points out that at least as early as 1900, the notion of a formal ontology has been distinguished from formal logic by the philosopher Husserl. While ontologies (even formal ontologies) have had a long history, they remained largely the topic of academic interest among philosophers, linguists, librarians, and knowledge representation researchers until somewhat recently.

Ontologies have been gaining interest and acceptance in computational audiences (in addition to philosophical audiences). Guarino [Guarino, 1998] provides a nice collection of fields that embrace ontologies including knowledge engineering, knowledge representation, qualitative modeling, language engineering, database design, information retrieval and extraction, and knowledge management and organization. That collection put together in early 1998 did not include nearly the web emphasis that is seen today. We would also include areas of library science [Dublin Core, 1999], ontology-enhanced search (e.g., eCyc (http://www.e-Cyc.com/) and FindUR [McGuinness, 1998]), possibly the largest one, e-commerce (e.g., Amazon.com, Yahoo Shopping, etc.), and configuration.

In this paper, we will be restricting our sense of ontologies to those we see emerging on the web. Today's use of ontology on the web has a different slant from the previous philosophical notions. One widely cited definition of an ontology is Gruber's [Gruber, 1993] "A specification of a conceptualization". We will use this notion and expand upon it in our use of the term.

People (and computational agents) typically have some notion or conceptualization of the meaning of terms. Software programs sometimes provide a specification of the inputs and outputs of a program, which could be used as a specification of the program. Similarly ontologies can be used to provide a concrete specification of term names and term meanings. Within the line of thought where an ontology is a specification of the conceptualization of a term, there are still a number of potential interpretations. Web ontologies may be viewed as a spectrum of detail in their specification. One might visualize a simple (linear) spectrum of definitions in Figure 6.2[3] below.

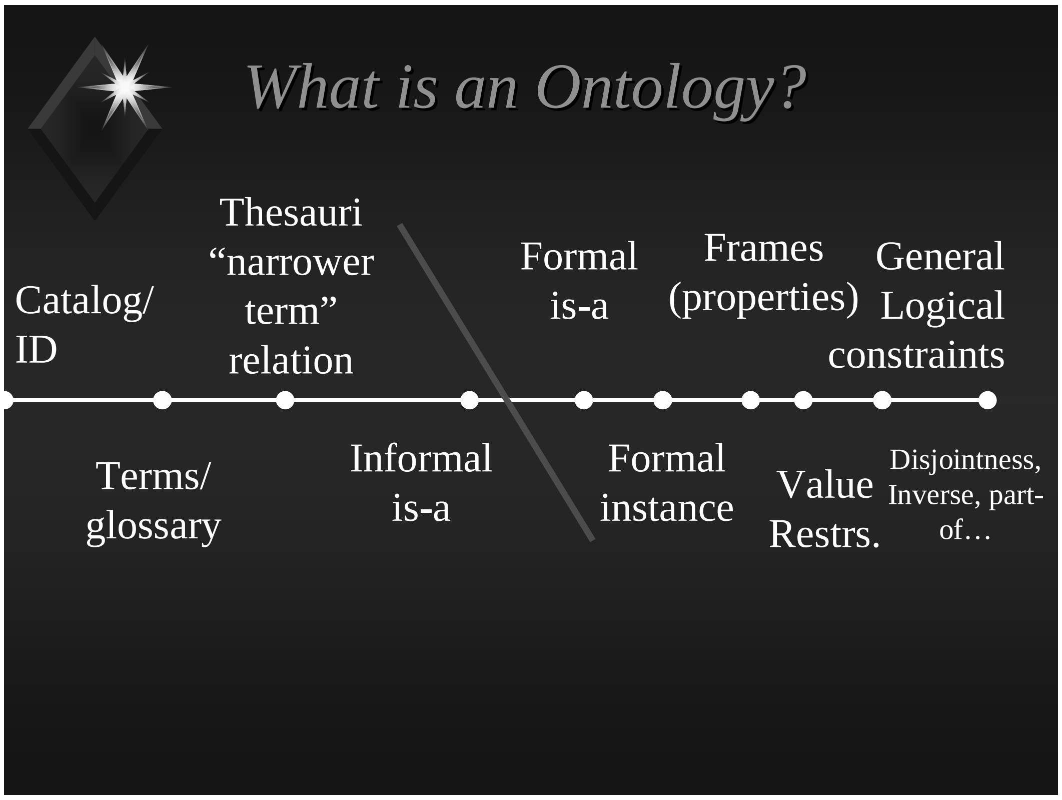


Figure 6.2 An Ontology Spectrum.

One of the simplest notions of a possible ontology may be a controlled vocabulary – i.e., a finite list of terms. Catalogs are an example of this category. Catalogs can provide an unambiguous interpretation of terms – for example, every use of a term, say car – will denote exactly the same identifier – say 25.

Another potential ontology specification is a glossary (a list of terms and meanings). The meanings are specified typically as natural language statements. This provides a kind of semantics or meaning since humans can read the natural language statements and interpret them. Typically, interpretations are not unambiguous and thus these specifications are not adequate for computer agents, thus this would not meet the criteria of being machine processable.

Thesauri provide some additional semantics in their relations between terms. They provide information such as synonym relationships. In many cases their relationships may be interpreted unambiguously by agents. Typically thesauri do not provide an explicit hierarchy (although with narrower and broader term specifications, one could deduce a simple hierarchy).

Early web specifications of term hierarchies, such as Yahoo’s, provide a basic notion of generalization and specialization. Yahoo, for example, provides a small number of top-level categories such as apparel and the category dresses as a kind of (women’s) apparel. Some people consider the previous categories (of catalogues, glossaries, and thesauri) to be ontologies but many prefer to have an explicit hierarchy included before something is considered an ontology. Yahoo, for example, does provide an explicit hierarchy. Its hierarchy is not a strict subclass or "isa" [Brachman, 1983] hierarchy however. This point was distinguished on the spectrum slide since it seems to capture many of the naturally occurring taxonomies on the web. In these organization schemes, it is typically the case that an instance of a more specific class is also an instance of the more general class but that is not enforced 100% of the time. For example, the general category apparel includes a subcategory women (which should more accurately be titled women’s apparel) which then includes subcategories accessories and dresses. While it is the case that every instance of a dress is an instance of apparel (and probably an instance of women’s dress), it is not the case that a dress is a woman and it is also not the case that a fragrance (an instance of a women’s accessory) is an instance of apparel. This mixing of categories such as accessories in web classification schemes is not unique to Yahoo – it appears in many web classification schemes[4]. Without true subclass (or true "isa") relationships, we will see that certain kinds of deductive uses of ontologies become problematic.

The next point on the figure includes strict subclass hierarchies. In these systems if A is a superclass of B, then if an object is an instance of B it necessarily follows that the object is an instance of A. For example, if "Dress" is a subclass of "Apparel" and "MyFavoriteDress" is an instance of "Dress", then it follows that "MyFavoriteDress" is an instance of "Apparel". Strict subclass hierarchies are necessary for exploitation of inheritance. The next point on the ontology spectrum includes formal instance relationships. Some classification schemes only include class names while others include ground individual content. This point includes instances as well.

The next point includes frames[5]. Here classes include property information. For example, the "Apparel" class may include properties of "price" and "isMadeFrom". My specific dress may have a price of $100 and may be made from cotton. Properties become more useful when they are specified at a general class level and then inherited consistently by subclasses and instances. In a consumer hierarchy, a general category like consumer product might have a "price" property associated with it. Possibly apparel would be the general category to which the property "isMadeFrom" is associated. This would mean that the domain of "isMadeFrom" is apparel. All subclasses of these categories would inherit these properties.

A more expressive point in the ontology spectrum includes value restrictions. Here we may place restrictions on what can fill a property. For example, a "price" property might be restricted to have a filler that is a number (or a number in a certain range) and "isMadeFrom" may be restricted have fillers that are a kind of material. One can now see a possible problem with a classification scheme that does not support a strict "isa" or subclass relationship. For example, if "Fragrance" were a subclass of "Apparel", it would inherit the property "isMadeFrom" and the value restriction of material that was stated.

As ontologies need to express more information, their expressive requirements grow. For example, we may want to fill in the value of one property based on a mathematical equation using values from other properties. Some languages allow ontologists to state arbitrary logical statements. Very expressive ontology languages such as that seen in Ontolingua [Farquhar et al, 1997] or CycL allow ontologists to specify first order logic constraints between terms and more detailed relationships such as disjoint classes, disjoint coverings, inverse relationships, part-whole relationships, etc.

In this paper, we will require the following properties to hold in order to consider something an ontology. Specifications meeting these properties will be referred to as simple ontologies.

- Finite controlled (extensible) vocabulary
- Unambiguous interpretation of classes and term relationships
- Strict hierarchical subclass relationships between classes

We consider the following properties typical but not mandatory:

- Property specification on a per-class basis
- Individual inclusion in the ontology
- Value restriction specification on a per-class basis

Finally, the following properties may be desirable but not mandatory nor typical:

- Specification of disjoint classes
- Specification of arbitrary logical relationships between terms

- Distinguished relationships such as inverse and part-whole

The line in our chart is drawn such that everything to the right of it will be called an ontology and meet at least the first three conditions stated above. Additionally, everything to the right of it can be used as a basis for inference.

3. Simple Ontologies and Their Uses

We will now consider ontologies and their impact on applications. We break this section into two parts: uses of simple ontologies and uses for more sophisticated ontologies. We do this because we acknowledge that building the more complicated ontologies may be cost prohibitive for certain applications.

Simple ontologies are not as costly to build and potentially more importantly, many are available. Simple ontologies are available in many forms – many exist as freeware on the web today and also many exist as internal information organization structures within companies, universities, etc. Some collaborative efforts exist such as DMOZ (*www.dmoz.com*) that are generating large simple ontologies. DMOZ, for example, leverages over 35,000 volunteer editors and at submission time, had over 360,000 classes in its taxonomy. Additionally, some more sophisticated ontologies are available today. For example, the unified medical language system (UMLS - *http://www.nlm.nih.gov/research/umls/* and [Humphreys &

Lindberg, 1993]) developed by the national library of medicine is a large sophisticated ontology about medical terminology. Some companies such as Cycorp (*www.cyc.com*) are making available portions of large, detailed ontologies. We will further address the issue of ontology acquisition and maintenance later, but for now, we just wanted to make the point that many simple and some sophisticated ontologies are easily available today.

Now lets consider some of the ways that simple ontologies may be used in practice.

First, they provide a *controlled vocabulary*. This by itself can provide great leverage since users, authors, and databases can all use terms from the same vocabulary. In addition programs can generate interfaces that encourage usage of the controlled terms. The result is that people use the same set of terms. Of course, some of the terms may still be used with different senses, but common term usage is a start for interoperability.

Second, a simple taxonomy may be used for *site organization and navigation support*. Many web sites today expose the top levels of a generalization hierarchy of terms as a kind of browsing structure. The categories are typically hot and a user may click on them to expand the subcategories.

Third, taxonomies may be used to support *expectation setting*. It is an important user interface feature that users be able to have realistic expectations of a site. If they may explore even the top level categories of the hierarchy, they can quickly determine if the site might have content (and/or services) of interest to them.

Fourth, taxonomies may be used as *"umbrella" structures from which to extend content*. Some freely available ontologies are attempting to provide the high level taxonomic organization from which many efforts may inherit terms. The UNSPSC (Universal Standard Products and Services Classification *www.unspsc.org*) is one such categorization scheme. It was jointly done by the United Nations Development Program and Dun & Bradstreet and was aimed at providing the infrastructure for interoperability of terms in the domains of products and services. It provides a classification scheme (with associated numbers). For example, Category 50 – Food, Beverage, and Tobacco Products has a subclass family 5010 called "Fruits and vegetables and nuts and seeds[6]" which in turn contains a subclass 501015 called Vegetables, which in turn has a subclass commodity 50101538 called fresh vegetables. The numbers provide a unique identification for each term and also encode the hierarchy. A number of e-commerce applications today are looking for such umbrella organization structures and in fact many have chosen to be compliant with the UNSPSC. Most applications will need to extend these ontologies, but if applications need to communicate between a number of content providers, it is convenient to use a shared umbrella or upper level ontology.

Fifth, taxonomies may provide *browsing support*. Content on a site may be tagged with terms from the taxonomy. This may be done manually in the style of Yahoo or it may be done automatically (possibly using a clustering approach). Once a page (or service) is meta-tagged with a term chosen from a controlled vocabulary, then search engines may exploit the tagging and provide enhanced search capabilities.

Sixth, taxonomies may be used to provide *search support*. A query expansion method may be used in order to expand a user query with terms from more specific categories in the hierarchy. We exploited this approach in our work on FindUR [McGuinness,1998] and found that under certain conditions (such as short document length and limited content areas), query expansion can radically improve search.

Seventh, taxonomies may be used to *sense disambiguation support*. If the same term appears in multiple places in a taxonomy, an application may move to a more general level in the taxonomy in order to find the sense of the word. For example, if an ontology contains the information that Jordan is an instance of a BasketballPlayer and also an instance of a country, an application may choose to query a user searching for Jordan if she is interested in basketball players or countries. Sense disambiguation using ontologies may be seen in the work of eCyc along with Hotbot and Lycos.

4. Structured Ontologies and Their Uses

Up to this point, we have focused on simple taxonomies for usage in applications. Once ontologies begin to have more structure however, they can provide more power in applications. Once the ontologies have more structure than simple generalization links, property information can be used in many forms.

First, they can be used for simple kinds of *consistency checking*. If ontologies contain information about properties and value restrictions on the properties, then type checking can be done within applications. For example, if a class called "Goods" has a property called "price" that has a value restriction of number, then something that is known to be of type "Goods" that has its "price" property filled in with a value that is not a number can be caught as an error. This just exploits simple value restrictions that designate the type of a value. A value restriction might include a range, for example, a number between 10 and 100. Then if the "price" is 10,000, it is out of the range and can be determined to be an error.

Second ontologies may be used to provide *completion*. An application may obtain a small amount of information from a user, such as the fact that she is looking for a high-resolution screen on a pc, and then have the ontology expand the exact pixel range that is to be expected. This can be done simply by defining what the term "HighResolutionPc" is with respect to a particular pixel range on two roles – "verticalResolution" and "horizontalResolution". Similarly, information may interact. For example, a medical system may obtain information from an ontology that if a patient is stated to be a man, then the gender of the patient is "male" and that information may be used to determine that a question concerning whether or not the patient is pregnant should not be asked since there could be information in the system that things whose gender is male are disjoint from things that are pregnant.

Third, ontologies may be able to provide *interoperability support*. In the simple case of considering controlled vocabularies, there is enhanced interoperability support since different users and applications are using the same set of terms. In simple taxonomies, we can recognize when one application is using a term that is more general or more specific than another term and facilitate interoperability. In more expressive ontologies, we may have a complete operational definition for how one term relates to another term and thus, we can use equality axioms or mappings to express one term precisely in terms of another and thereby support more "intelligent" interoperability. For example, an ontology may include a definition that a "StanfordEmployee" is equal to a "Person" whose "employer" property is filled with the individual "StanfordUniversity". This definition may be used to expand the term "StanfordEmployee" in an application that does not understand either "StanfordEmployee" or "Employee" but does understand the terms "Person", "employer", and "StanfordUniversity".

Fourth, ontologies may be used to *support validation and verification testing* of data (and schemas). If an ontology contains class descriptions, such as "StanfordEmployee", these definitions may be used as queries to databases to discover what kind of coverage currently exists in datasets. For example, if one was going to expose the class "StanfordEmployee" on an interface to some application, it would be useful to know first if the dataset contained any instances of "Person" whose "employer" property was filled with the value "Stanford". Additionally, if in a simple data model, we stated that a "Person" had at most one "employer", then we could use that information to check to see if any current information on "Person"s in the dataset contained more than one "employer" value. Similarly, checks could be done to see if there were currently "Person"s in the dataset that were known to be "Employee"s yet did not have a value for the "employer" property (thereby showing that the dataset is not complete). Chimaera [McGuinness-et-al, 2000] is an example ontology evolution environment that provides a set of diagnostics tests for checking ontologies for both problems in the ontology definitions as well as problems with the instance data. It looks for provable inconsistencies as well as conditions that "typically" reflect situations where an ontology or the data may need to be fixed.

Fifth, ontologies containing markup information may *encode entire test suites*. An ontology may contain a number of definitions of terms, some instance definitions, and then include a term definition that is considered to be a query – find all terms that meet the following conditions. Markup information could be encoded with this query to include what the answer should be, thus providing enough information to encode regression testing data. We provide one such example ontology in *http://ksl.stanford.edu/projects/DAML/chimaera-jtp-cardinality-test1.daml*. The ontology contains a regression test suite for checking cardinality inferences (such as persons having two employers yet being stated to have at most one employer) in a Stanford Theorem prover (*http://www.ksl.Stanford.EDU/software/jtp/*).

Sixth, ontologies can provide the foundation for *configuration support*. Class terms may be defined so that they contain descriptions of what kinds of parts may be in a system. Additionally interactions between properties can be defined so that filling in a value for one property can cause another value to be filled in for another slot. For example, one may generate an ontology of information about home theatre products as is done in a small configurator example using a simple description logic-based system [McGuinness-et-al, 1995]. Terms such as "Television", "Amplifier", "Tuner", etc are defined. Additionally, information connecting the terms together is included. A class of "HighQualityTelevision"s is defined so that users may choose from this class and the configurator will automatically fill in limited sets of manufacturers to choose from, minimum diagonal values, minimum price ranges etc. Also, information is encoded that propagates restrictions from one component to another. For example, some of the components in this system were meant to be sold in pairs. If one buys one particular kind of speaker (which is only sold in pairs, thus two speakers are added to the parts list), then restrictions on particular speaker stands appear in the configuration specification. There are many such configuration examples using ontologies, a few of which are described in a special configuration issue of Artificial Intelligence for Engineering Design, Analysis, and Manufacturing Journal [Darr et al., 1998].

Seventh, ontologies can *support structured, comparative, and customized search*. For example, if one is looking for televisions, a class description for television may be obtained from an ontology, its properties may be obtained (such as diagonal, price, manufacturer, etc), and then a comparative presentation may be made of televisions by presenting the values of each of the properties. Those properties can also be used to provide a form for users to fill in so that they may provide a detailed set of specifications about the items they are looking to find. This also provides the foundation for providing a number of different search interfaces – the simple text box where the user is expected to type a textual query along with search interfaces exposing important properties of products that can provide a structured search query. More sophisticated ontologies may be generated that mark which properties are most useful to present in comparative analyses so that users may have concise descriptions of the products instead of comparisons in complete detail. Thus, ontologies with markup information may also be used to prune comparative searches.

Eighth, ontologies may be used to *exploit generalization/ specialization information*. If a search application finds that a user’s query generates too many answers, one may dissect the query to see if any terms in it appear in an ontology, and if so, then the search application may suggest specializing that term. For example, if one did a search for concerts in the San Francisco Bay area and obtained too many answers, a search engine might look up concert in an ontology and discover that there are subclasses of concert (and it may also discover that there are specific concert locations in the Bay area).

The search engine could then choose to present the user with the option of looking for a particular kind of concert (say rock concert) which would restrict the search, thereby returning fewer answers. Further the search engine could proactively run queries in the background while waiting for user input or also cache information from popular queries. Then the search engine could also present a list of subclasses of concerts and provide the user with the approximate number of retrievals the user would get if they specialised their query in the different manners. These are just some of the ways in which ontologies may be used to refine search queries. We could also look at the ontology to provide alternative values (by looking at siblings in the ontology) for terms specified in the search query.

We have not claimed to present an exhaustive list of the ways in which ontologies may be used in applications. The above lists are illustrative of some ways that ontologies have been used to support intelligent applications.

5. Ontology Acquisition

After having presented some of the ways ontologies are useful components in applications, we will now look at some sources of ontologies. As mentioned previously, many ontologies exist in the public domain. It may be possible to start with an existing industry standard and use that as the ontology starting point. Most likely application developers will need to modify and/or extend ontologies that are available and were developed for other uses. Still, one methodology for obtaining ontologies is to begin with an industry standard ontology and then modify or extend it.

Another methodology is to semi-automatically generate a starting point for an ontology. Many taxonomic structures exist on the web or in the table of contents of documents. One might crawl certain sites to obtain a starting taxonomic structure and then analyze, modify, and extend that.

One question is where to look for existing ontologies or sources of information to be crawled. Many controlled vocabularies are being made available today. Sometimes standards organizations, such as NIST (the National Institute of Standards and Technology - *http://www.nist.gov/*), support efforts in producing controlled vocabularies and ontologies. Some consortiums are forming to generate ontologies. See, for example, RosettaNet (*http://www.rosettanet.org*) in the area of information technology, electronic technology, electronic components, and semiconductor manufacturing. They are creating industry-wide open e-business standards and providing a language for business processes. Sometimes trade organizations provide class hierarchies on their sites that can also be used as a standard structured controlled vocabulary. There are also broad sources of class structures. Essentially every e-commerce site today encodes at least a taxonomic organization of terms. Sites like Amazon in organizing their book and music information provides a very broad organization of information. Also, some government programs are generating large ontologies that are being put into the public domain for reuse, such as many ontologies generated in the Darpa High Performance Knowledge Base Program, the Rapid Knowledge Formation Program (*http://reliant.teknowledge.com/RKF/*).

Another emerging trend is the use of markup languages. Some pages are being annotated using markup languages such as XML, RDF, DAML, etc. The pages including the annotations may be using markup terms from controlled vocabularies. Some libraries are emerging of ontologies potentially of use for markup. For example, the DAML program maintains a library of DAML ontologies in *http://www.daml.org/ontologies/*.

Much of this section has introduced the idea of obtaining either a simple or complex ontology as a starting point and then analyzing, modifying, and maintaining it over time. In the next section, we will address the issue of implications and needs from ontology-based applications.

6. Ontology-related Implications and Needs

When starting an ontology-based application, the two major concerns will be language and environment.

Language: When considering ontology-related applications, inevitably the issue of *ontology language* will arise. An ontology must be encoded in some language. If one is using a simple ontology, few issues arise. However, if one is considering a more complex ontology, expressive power of a representation and reasoning needs to be considered. As with any problem where a language is being chosen, it must be epistemologically adequate -- the language must be able to express the concepts in the domain. For example, if one wants to do range checking in an e-commerce application, then it would be unwise to choose a simple language that only contains subclass and instance relationships and does not include property specification with value restrictions. There are a number of candidate ontology languages – in fact there are so many that some research efforts arose in the last decade in order to produce standard specification languages (such as the KRSS effort – the Knowledge Representation System Specification effort [Patel-Schneider-Swartout, 1992]), interchange formats (such as KIF -the Knowledge Interchange Format which is now a proposed ANSI standard [KIF]), and common application programming interface standards (such as OKBC – Open Knowledge Base Connectivity [Chaudhri et al., 1997]).

One does not just want to consider representational constructs in a language; one also wants to consider the reasoning that may be supported in the language. Some fields, such as description logics (*www.dl.kr.org*), make this a central focus in language design. They look for tradeoffs that maintain expressive power needed by applications and also consider what it takes to provide inference engines that can provide deductions based on the constructs represented in the language. For example, if a language supports the notion of stating that two classes are disjoint, then a reasoning engine should be able to be built that enforces the constraint that the classes are disjoint. Thus, an inference engine should be able to warn a user if she is creating an instance or subclass of two disjoint classes.

Also, a language should be usable with existing platforms and should be something that non-experts can use to do their conceptual modeling. The web is clearly the most important platform with which to be compatible today, thus any language choice should be able to leverage the web. Additionally, frame-based systems have had a long history of being thought of as conceptually easy to use, thus a frame paradigm may be worth considering.

Language efforts seen today attempt to take the best of the research on expressive power along with reasoning power and provide representationally powerful languages that have known reasoning properties, and of course, are efficient in their implementations. The DARPA Agent Markup Language program, for example, attempted to take the emerging web languages of today such as XML and RDF and create a language that is web compatible, incorporates the ease of use of frame-based systems, and draws on the 20 year history of description logics in choosing language constructs along with reasoning paradigms. The resulting language –DAML+OIL – attempts to merge the best of existing web languages, description logics, and frame-based systems. OIL, a modern description logic aimed also at web compatibility, [Bechhofer et al., 2000] attempts to provide a layered approach to language design.

Environment: Another consideration is how to generate, analyze, modify, and maintain an ontology over time. If the ontology is to be generated and maintained by subject matter experts (and not require knowledge experts), some ontology support tools will be needed. There are a number of simple ontology tools available commercially. Some information retrieval companies such as Verity (*www.verity.com*) have provided simple editors for generating and browsing simple generalization hierarchies. Verity, for example, has provided a "topic editor" for years which will support users in generating taxonomies and utilizing them in search queries. Research efforts have existed for many years in producing ontology toolkits. Stanford University’s previously mentioned tools of Ontolingua [Farquhar et al, 1997] and Chimaera [McGuinness et al., 2000] are just two examples, however examples abound including OilEd (*http://img.cs.man.ac.uk/oil/*) from Manchester University and Protégé [Protégé 2000] from Stanford Medical Informatics, just to name a few. Application developers may choose commercial vendors as their toolkit provider, sophisticated research applications as the base, or somewhere in between. Some companies with extensive ontology needs such as VerticalNet (*http://www.verticalnet.com/*) have or are developing their own ontology tools in order to build ontologies that meet the needs of a sophisticated commercial ontologist. Their tools were built after analyzing existing research prototypes and were then designed to meet the commercial standards required in diverse, collaborative, e-commerce applications of today.

When choosing to use or build an ontology environment, there are a number of issues that should be considered including the following:

- *Collaboration and distributed workforce support*. Some ontology environments allow users to share a common workspace –i.e., see each other's work environments. This can be particularly useful for debugging. Ontolingua, for example, supports this notion through its use of sessions. Additionally, when workers are distributed in location, it becomes important to have an environment that allows access from multiple places. This is becoming much more typical today with server/client architectures. Finally, collaboration may require concurrency control, locking, and a kind of versioning and permission system.
- *Platform interconnectivity*. As applications become embedded in more complex platforms, it becomes important for environments to be able to read and write compatible formats, be able to be integrated with multiple hardware/software environments, etc. Java-based applications provide a convenient approach to this problem but other systems that support multiple input and output formats, understand common standards, and provide translation and mapping services may help.

- *Scale*. any ontology applications today may need to scale a few orders of magnitude larger than past applications. It is important to look at scaling in terms of size of ontologies as well as numbers of simultaneous users.
- *Versioning*. As applications become long-lived and also are deployed in different environments possibly internationally, it becomes important to be able to support many versions of ontologies. In typical software engineering environments, there are source code control systems and versioning that address these issues and special ontology-oriented change management systems are evolving.
- *Security*. Some applications will have needs for differing access to portions of the ontology. Thus, it is important to have an environment that can expose portions of the ontology based on a security model. The security model may need to support both read and write access.
- *Analysis*. Environments are expected to support acquisition, evolution, and maintenance of ontologies. Thus, it would be common to expect ontologies to have periods when they are incomplete and incorrect. Analysis support that can focus the user’s attention in areas that are likely to need modification can be quite useful. The Chimaera ontology environment, for example, supports a number of diagnostic tests aimed at helping users identify provably incorrect ontologies as well as possible problems.
- *Lifecycle issues*. As ontologies become larger and longer lived, it would be expected that application developers might be maintaining ontologies over many years. Additionally, they may be constantly merging new ontologies into their system as their applications interconnect with more diverse systems. Thus, it becomes important to consider support for ontology evolution issues such as merging terms, breaking apart terms, multiple name spaces,

source code control systems, truth maintenance systems, regression testing systems, etc.

- *Ease of use*. Even if an environment has everything an application developer may need, if it is difficult for the user to decide how to use parts of the environment, they may not get used. Thus training materials, tutorials, conceptual modeling support, graphical browsing tools, etc. all may be important. We have written separately on some of the issues required to make description logic-based systems usable in mainstream use. [McGuinness-Patel-Schneider, 1998, Brachman et al., 1999].
- *Diverse user support*. Some environments are made for power users, some for naïve users, and some have settings that allow users to customize environments as appropriate to the type of user. It is important to determine if the environment can support all of the types of users anticipated. The support should be in all areas along the spectrum of initially generating the ontology in the planning and conceptual modeling state, to evolving it, diagnosing it, maintaining it, etc.
- *Presentation Style*. Possibly closely related to user type is presentation style. Some users need to see extensive detail, some need pruned information, and some need abstractions. Presentation of information may be textual, graphical, or other. While no one environment needs to support all presentation styles, it is important that the environment is extensible enough to have new presentation methods added when needed.
- *Extensibility*. It will be impossible to anticipate all of the needs an application will have. Thus, it is important to use an environment that can adapt along with the needs of the users and the projects.

Conclusions

In this paper, we have noted the emergence of ontologies from academic obscurity into mainstream business and practice on the web. We have introduced the term ontology along with a spectrum of properties that ontologies may exhibit. We have provided criteria necessary, prototypical, and desirable for simple and complex ontologies. We have also identified ways that ontologies (both simple and complex) are being and may be used to provide value in many types of applications. We have addressed the issue of acquiring ontologies and then addressed the issues of maintenance and evolution. Finally, we have identified a number of ontology-related issues that arise from the emergence of ontologies focusing on ontology language and environment. Finally, we concluded with issues that are gaining importance as ontologies grow in their importance and centrality in diverse applications.

Acknowledgements

This paper has evolved as a result of a talk initially presented at the Semantics for the Web Seminar of the Dagstuhl Seminar series in March, 2000 (*http://www.dagstuhl.de/DATA/Reports/00121/* and *http://www.semanticweb.org/events/dagstuhl2000/*). The ideas have been enhanced by many people including many of my collaborators on past ontology environments, in particular collaborators on the CLASSIC (particularly Patel-Schneider, FindUR, and Chimaera environments. It also benefited from joint work with Ora Lassila using it as a foundation for a portion of work on the role of frame-based representation on the semantic web. Additionally, support from the DARPA HPKB, RKF, and DAML programs has helped finance it, motivate, and guide it. Finally, it is the result of a longstanding interest in ontologies and the meanings of terms, first inspired by Richard McGuinness, my father. He generated the first spark many years ago and then helped inspire me that the time had come to take the work to the masses. His influence was still felt in the initial Dagstuhl talk but he has since passed and thus my continuing work is dedicated to him.

References

[Bechhofer et al., 2000] Sean Bechhofer, Jeen Broekstra, Stefan Decker, Michael Erdmann, Dieter Fensel, Carole Goble, Frank van Harmelen, Ian Horrocks, Michel Klein, Deborah McGuinness, Enrico Motta, Peter Patel-Schneider, Steffen Staab, and Rudi Studer, "An informal description of Standard Oil and Instance OIL", available on-line as http://www.ontoknowledge.org/oil/downl/oil-whitepaper.pdf

[Berners-Lee, 2000] Tim Berners-Lee, "Semantic Web on XML", Keynote presentation for XML 2000. Slides available at: http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide1-0.html. Reporting available at: http://www.xml.com/pub/a/2000/12/xml2000/timbl.html

[Berners-Lee, 1999] Tim Berners-Lee, "Weaving the Web", Harper, San Francisco, 1999. http://www.harpercollins.com/hc/bookpage/index.asp?isbn=0062515861

[Berners-Lee et al, 1998] Tim Berners-Lee, Roy Fielding, and Larry Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", Internet Draft Standard RFC 2396, August 1998; available on-line as http://www.isi.edu/in-notes/rfc2396.txt.

[Brachman, 1983] Ronald J. Brachman, R. J. What ISA Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks. IEEE Computer, 16 (10), 30--6. 1983.

[Brachman et al., 1999] Ronald J. Brachman, Alex Borgida, Deborah L. McGuinness, and Peter F. Patel-Schneider. "Reducing" CLASSIC to Practice: Knowledge Representation Theory Meets Reality. In Artificial Intelligence 114(1-2) pages 203-237, October, 1999.

[Brickley & Guha, 2000] Dan Brickley & R.V.Guha, "Resource Description Framework (RDF) Schema Specification 1.0", W3C Candidate Recommendation 27 March 2000, World Wide Web Consortium, Cambridge (MA); available on-line as http://www.w3.org/TR/rdf-schema/.

[Broekstra et. al., 2001] J. Broekstra, M. Klein, S. Decker, D. Fensel, F. van Harmelen, and I. Horrocks. "Enabling knowledge representation on the Web by Extending RDF Schema", Proceedings of the International Conference on the World Wide Web (WWW10), May 2001.

[Chaudhri et. al., 1998] Vinay Chaudhri, Adam Farquhar, Richard Fikes, Peter Karp, and James Rice; "OKBC: A Programmatic Foundation for Knowledge Base Interoperability", AAAI 1998.

[Darr et. al., 1998] Tim Darr, Mark Fox, and Deborah L. McGuinness, editors. Special Configuration Issue of the Artificial Intelligence for Engineering Design, Analysis, and Manufacturing Journal 1998.

[Dublin Core, 1999] "Dublin Core Metadata Element Set, Version 1.1: Reference Description", Dublin Core Metadata Initiative, 1999; available on-line as http://purl.org/dc/documents/rec-dces-19990702.htm

[Farquhar et al, 1997] Adam Farquhar, Richard Fikes, and James Rice; "The Ontolingua Server: a Tool for Collaborative Ontology Construction", Intl. Journal of Human-Computer Studies **46**, 1997

[Fikes & Kehler, 1985] Richard Fikes & Tom Kehler, "The Role of Frame-Based Representation in Reasoning", CACM 28(9): 904-920, 1985.

[Fikes & McGuinness, 2001] Richard Fikes & Deborah L. McGuinness, "An Axiomatic Semantics for RDF, RDF Schema, and DAML+OIL", KSL Technical Report KSL-01-01, Stanford University, 2001; available on-line as http://www.ksl.stanford.edu/people/dlm/daml-semantics/abstract-axiomatic-semantics.html.

[Google, 2000] Connie Guglielmo and Charles Babcock, "Gaga over Google" , Interactive Week, Nov 6, 2000. http://www.zdnet.com/intweek/stories/news/0,4164,2651081,00.html, and http://www.google.com/about.html

[Guarino, 1998] Nicola Guarino, "Formal Ontology and Information Systems". In the Proceedings of Formal Ontology in Information Systems, June 1998. Also in Frontiers in Artificial Intelligence and Applications, IOS-Press, Washington, DC, 1998.

[Gruber, 1993] Tom R. Gruber, "A translation approach to portable ontologies". Knowledge Acquisition, 5(2):199-220, 1993.

[Hendler & McGuinness, 2000] James Hendler and Deborah McGuinness. ``The DARPA Agent Markup Language". In IEEE Intelligent Systems Trends and Controversies, November/December 2000. Available from http://www.ksl.stanford.edu/people/dlm/papers/ieee-daml01-abstract.html .

[Humphreys & Lindberg, 1993] B. L. Humphreys and D. A. B. Lindberg. "The UMLS project: making the conceptual connection between users and the information they need. Bulletin of the Medical Library Association 81(2): 170.

[Husserl, 1900] Edmund Husserl, Logische Untersuchungen, First edition Halle: Niemeyer, 1900/01.

[KIF] Knowledge Interchange Format, Language Description, draft proposed national standard. NCITS.T2/98-004. http://logic.stanford.edu/kif/kif.html.

[Karp, 1992] Peter D. Karp, "The design space of frame knowledge representation systems", Technical Report 520, SRI International AI Center; available on line as ftp://www.ai.sri.com/pub/papers/karp-freview.ps.Z

[Lassila, 1998] Ora Lassila, "Web Metadata: A Matter of Semantics", IEEE Internet Computing 2(4): 30-37 (1998).

[Lassila & Swick, 1999] Ora Lassila & Ralph Swick, "Resource Description Framework (RDF) Model and Syntax Specification", W3C Recommendation 22 February 1999, World Wide Web Consortium, Cambridge (MA); available on-line as http://www.w3.org/TR/REC-rdf-syntax/.

[McGuinness et al., 1995] Deborah L. McGuinness, Lori Alperin Resnick, and Charles Isbell. ``Description Logic in Practice: A CLASSIC: Application." In Proceedings of the 14th International Joint Conference on Artificial Intelligence, Montreal, Canada, August, 1995.

[McGuinness, 1998] Deborah L. McGuinness "Ontological Issues for Knowledge-Enhanced Search". In the Proceedings of Formal Ontology in Information Systems, June 1998. Also in Frontiers in Artificial Intelligence and Applications, IOS-Press, Washington, DC, 1998.

[McGuinness et al., 2000] Deborah L. McGuinness, Richard Fikes, James Rice, and Steve Wilder. An Environment for Merging and Testing Large Ontologies. In the Proceedings of the Seventh International

Conference on Principles of Knowledge Representation and Reasoning (KR2000), Breckenridge, Colorado, USA. April 12-15, 2000.

[McGuinness-Patel-Schneider] Deborah L. McGuinness and Peter F. Patel-Schneider. "Usability Issues in Knowledge Representation Systems". In Proceedings of the Fifteenth National Conference on Artificial Intelligence, Madison, Wisconsin, July, 1998. This is an updated version of ``Usability Issues in Description Logic Systems" published in Proceedings of International Workshop on Description Logics, Gif sur Yvette, (Paris), France, September, 1997.

[Minsky, 1975] Marvin Minsky, "A Framework for Representing Knowledge", in Patrick Henry Winston (ed.), The Psychology of Computer Vision, McGraw-Hill, New York, 1975.

[Patel-Schneider-Swartout] Peter F. Patel-Schneider and Bill Swartout; "Description-Logic Knowledge Representation System Specification"; KRSS Group of the ARPA Knowledge Sharing Effort. http://www-db.research.bell-labs.com/user/pfps/papers/krss-spec.ps

[Protégé 2000] The Protege Project. http://protege.stanford.edu

[Smith, 1998] Barry Smith, "Basic Concepts of Formal Ontologies", in N. Guarino (Ed.) Formal Ontology in Information Systems, IOS Press, 1998.

[Woods. 1975] William A. Woods, "What's in a Link: Foundations for Semantic Networks", in D.G.Bobrow & A.M.Collins (eds.), Representation and Understanding: Studies in Cognitive Science, 35-82, Academic Press, New York, 1975.

[XML 2000] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler, editors. Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation 6 October 2000. http://www.w3.org/TR/2000/REC-xml-20001006

Footnotes

[1]Associate Director and Senior Research Scientist Knowledge Systems Laboratory Stanford University Stanford, CA 94305, dlm@ksl.stanford.edu

[2]http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html

[3]This spectrum arose out of a conversation in preparation for an ontology panel at AAAI ’99. The panelists (Lehman, McGuinness, Ushold, and Welty), chosen because of their years of experience in ontologies

found that they encountered many forms of specifications that different people termed ontologies. McGuinness refined the picture to the one included here.

[4]Some prominent hierarchies such as Yahoo have renamed their classes to broad disjunctive categories such as "Apparel, Accessories, and Shoes" presumably in order to provide for more strict subclass relationships. Disjunctive categories make inheritance more problematic however with class-specific properties.

[5]Frames were introduced by Minsky [Minsky 1975] and have been widely adopted, see for example [Fikes & Kehler 1985, Karp 1992, and Chaudhri-et-al 1998].

[6]Note, if one is using the common logical meanings of connectives, this class should really be named "Fruits or vegetables or nuts or seeds".

II Knowledge support

7. Sesame: An Architecture for Storing and Querying RDF Data and Schema Information

Jeen Broekstra[1], Arjohn Kampman[2], Frank van Harmelen[3]

1. Introduction

The Resource Description Framework (RDF) [Lassila and Swick, 1999] is a W3C Recommendation for the notation of meta-data on the World Wide Web. RDF Schema [Brickley and Guha, 2000] extends this standard by providing developers with the means to specify vocabulary and to model object structures.

These techniques will enable the enrichment of the Web with machine-processable semantics, thus giving rise to what has been dubbed the Semantic Web. However, simply having this data available is not enough. Tooling is needed to process the information, to transform it, to reason with it. As a basis for this, we have developed Sesame, an architecture for efficient storage and expressive querying of large quantities of RDF meta-data. Sesame is being developed by Aidministrator Nederland b.v.[4] as part of the European IST project On-To-Knowledge[5] [Fensel et al., 2000].

This paper is organized as follows: in Section 2 we give a short introduction to RDF and RDF Schema. This section is only to make the paper self-contained, and can be skipped by readers already familiar with these languages.

In Section 3 we discuss why a query language specifically tailored to RDF and RDF Schema is needed, over and above existing query languages such as XQuery. In section 4 and 5 we look in detail at Sesame’s architecture. Section 6 discusses our experiences with Sesame until now, and section 7 looks into possible future developments. Finally we provide our conclusions in Section Conclusion.

2. RDF and RDF Schema

The Resource Description Framework (RDF) [Lassila and Swick, 1999] is a W3C recommendation that was originally designed to standardize the definition and use of metadata-descriptions of Web-based resources. However, RDF is equally well suited to representing arbitrary data, be they meta-data or not.

2.1. RDF

The basic building block in RDF is an object-attribute-value triple, commonly written as $A(O; V)$. That is, an object O has an attribute A with value V. Another way to think of this relationship is as a labeled edge between two nodes: *[O] _ A ! [V].* This notation is useful because RDF allows objects and values to be interchanged. Thus, any object from one triple can play the role of a value in another triple, which amounts to chaining two labeled edges in a graphic representation. The graph in Figure 7.1 for example, expresses the following relationships:

HasName (’http://www.famouswriters.org/twain/mark’, "Mark Twain")

hasWritten (’http://www.famouswriters.org/twain/mark’, ’http://www.books.org/ISBN0001047582’)

title (’http://www.books.org/ISBN0001047582’, "The Adventures of Tom Sawyer")

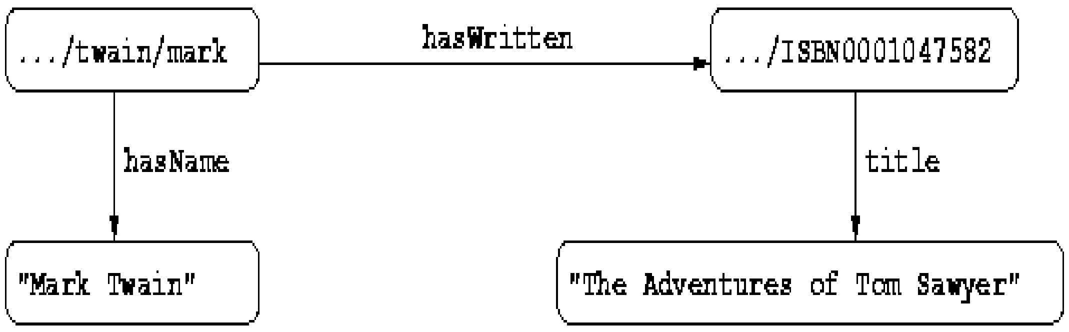


Figure 7.1 An example RDF data graph, capturing three statements

RDF also allows a form of reification[6] in which any RDF statement itself can be the object or value of a triple. This means graphs can be nested as well as chained. On the Web this allows us, for example,to express doubt or support for statements created by other people. Finally, it is possible to indicate that a given object is of a certain type, such as stating that “ISBN0001047582” is of the type *Book*, by creating a type edge referring to the *Book* definition in an RDF schema:

type(’http://www.books.org/ISBN0001047582’,’http://www.description.org/schema#Book’)

The RDF Model and Syntax specification also proposes an XML syntax for RDF data models. One possible serialisation of the above relations in this syntax, would look like this:

```
     <rdf:Description
rdf:about="http://www.famouswriters.org/twain/mark">
     <s:hasName>Mark Twain</s:hasName>
     <s:hasWritten
rdf:resource="http://www.books.org/ISBN0001047582"/>
     </rdf:Description>
     <rdf:Description
rdf:about="http://www.books.org/ISBN0001047582">
     <s:title>The Adventures of Tom Sawyer</s:title>
     <rdf:type
rdf:resource="http://www.description.org/schema#Book"/>
     </rdf:Description>
```

Since the proposed XML syntax allows many alternative ways of writing down information (and indeed still other syntaxes may be introduced), the above XML syntax is just one of many possibilities of writing down an RDF model in XML. It is important to note that RDF is designed to provide a basic object-attribute-value model for Webdata.

Other than this intentional semantics – described only informally in the standard – RDF makes no data modeling commitments. In particular, no reserved terms are defined for further data modeling. As with XML, the RDF data model provides no mechanisms for declaring property names that are to be used.

2.2. RDF Schema

RDF Schema [Brickley and Guha, 2000] is a mechanism that lets developers define a particular vocabulary for RDF data (such as *hasWritten*) and specify the kinds of objects to which these attributes can be applied (such as *Writer*). RDF Schema does this by pre-specifying some terminology, such as *Class, subClassOf* and *Property,* which can then be used in application-specific schemata. RDF Schema expressions are also valid RDF expressions – in fact, the only difference with ‘normal’ RDF expressions is that in

RDF Schema an agreement is made on the *semantics* of certain terms and thus on the *interpretation* of certain statements.

For example, the *subClassOf* property allows the developer to specify the hierarchical organization of classes. Objects can be declared to be instances of these classes using the *type* property. Constraints on the use of properties can be specified using *domain* and *range* constructs.

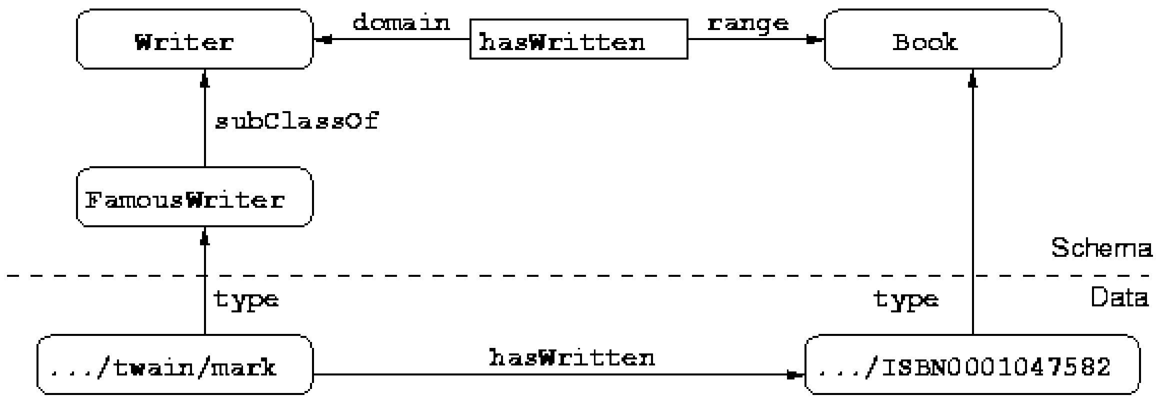


Figure 7.2 An example RDF Schema, defining vocabulary and a class hierarchy

Above the dotted line in Figure 7.2, we see an example RDF schema that defines vocabulary for the RDF example we saw earlier: *Book*, *Writer* and *FamousWriter* are introduced as classes, and *hasWritten* is introduced as a property. A specific instance is described in terms of this vocabulary below the dotted line.

3. The need for an RDF/S Query Language

RDF documents and RDF schemata can be considered at three different levels of abstraction:

1. at the *syntactic level* they are XML documents
2. at the *structure level* they consist of a set of triples
3. at the *semantic level* they constitute one or more graphs with partially predefined semantics.

We can query these documents at each of these three levels. We will briefly consider the pros and cons of doing so for each level in the next sections. This will lead us to conclude that RDF(S) documents should really be queried at the semantic level. We will briefly discuss RQL, a language for querying RDF(S) documents at the semantic level.

3.1. Querying at the syntactic level

As we have seen in Section 2, any RDF model (and therefore any RDF schema) can be written down in XML notation. It would therefore seem reasonable to assume that we can query RDF using an XML query language (for example, XQuery [Chamberlin et al., 2001]).

However, this approach disregards the fact that RDF is not just an XML dialect, but has its own data model that is very different from the XML tree structure. Relationships in the RDF data model that are not apparent from the XML tree structure become very hard to query.

As an example, let us look again at the XML description of the RDF model in Figure 7.1.

```
     <rdf:Description
rdf:about="http://www.famouswriters.org/twain/mark">
     <s:hasName>Mark Twain</s:hasName>
     <s:hasWritten
rdf:resource="http://www.books.org/ISBN0001047582"/>
     </rdf:Description>
     <rdf:Description
rdf:about="http://www.books.org/ISBN0001047582">
     <s:title>The Adventures of Tom Sawyer</s:title>
     <rdf:type
rdf:resource="http://www.description.org/schema#Book"/>
     </rdf:Description>
```

In an XML query language such as XQuery [Chamberlin et al., 2001], expressions to traverse the data structure are tailored towards traversing a node-labeled tree. However, the RDF data model in this instance is a graph, not a tree, and moreover, both its edges (properties) and its nodes (subjects/objects) are labeled.

In querying at the syntax level, this is literally left as an excercise for the query builder: one cannot query the relation between the resource signifying ‘Mark Twain’ and the resource signifying ‘The Adventures of Tom Sawyer’ without knowledge of the syntax that was used to encode the RDF data in XML. Ideally, we would want to formulate a query like “Give me all the relationships that exist between Mark Twain and The Adventures of Tom Sawyer”. However, using only the XML syntax, we are stuck with formulating an awkward query like “Give me all the elements nested in a *Description* element with an *about* attribute with value ’http://www.famouswriters.org/twain/mark’, of which the value of its *resource* attribute occurs elsewhere as the *about* attribute value of a *Description* element which has a nested element *title* with the value ‘The Adventures of Tom Sawyer’.”

Not only is this approach inconvenient, it also disregards the fact that the XML syntax for RDF is not unique: different ways of encoding the same information in XML are possible and in use currently. This means that one query will never be guaranteed to retrieve all the answers from an RDF model.

3.2. Querying at the structure level

When we abstract from the XML linearisation syntax, any RDF document represents a set of triples, each triple representing a statement of the form Object-Attribute-Value. A number of query languages have been proposed and implemented that regard RDF documents as such a set of triples, and that allow to query such a triple set in various ways. See http://perso.enst.fr/˜ta/web/rdf/rdf-query.html for a recent overview.

The RDF/RDF Schema example from Figure 7.2 corresponds to the following set of triples:

(type Book Class)

(type Writer Class)

(type FamousWriter Class)

(subClassOf FamousWriter Writer)

(type hasWritten Property)

(domain hasWritten Writer)

(range hasWritten Writer)

(type twain/mark FamousWriter)

(type ISBN0001047582 Book)

(hasWritten twain/mark ISBN0001047582)

An RDF query language would allow us to query which resources are known to be of type *FamousWriter:*

select ?x from ... where (type ?x FamousWriter)

The clear advantage of such a query is that it directly addresses the RDF data model, and that it is therefore independent of the specific XML syntax that has been chosen to represent the data.

However, a major shortcoming of any query-language at this level is that it interprets *any* RDF only as a set of triples, including those elements which have been given a special semantics in RDF Schema.

For example, since *http://www.famouswriters.org/twain/mark* is of type *FamousWriter*, and since *FamousWriter* is a subclass of *Writer*, *http://www.famouswriters.org/twain/mark* is also of type *Writer*,by virtue of the intended RDF Schema semantics of *type* and *subClassOf*. However, there is no triple that explicitly asserts this fact. As a result, the query

SELECT ?x FROM (type ?x Writer)

will fail because the query only looks for explicit triples in the store, whereas the triple *(type /twain/mark Writer)* is not explicitly present in the store, but is implied by the semantics of RDF Schema. Notice that simply expanding the query into something like

SELECT ?x ?c1 ?c2 ?c3

FROM (type ?x ?c1),

```
(subClassOf ?c2 ?c3)
WHERE ?c1 = ?c2
```

will solve the problem in this specific example, but does not cater for a chain of subClassOf triples, etc.

3.3. Querying at the semantic level: RQL

What is clearly required is a query language that is sensitive to the semantics of the RDF Schema primitives.

RQL [Karvounarakis et al., 2000, Alexaki et al., 2000] is the first (and to the best of our knowledge currently the only) proposal for a declarative query language for RDF and RDF Schema. It is being developed within the European IST project C-Web and its followup project MESMUSES by the Institute of Computer Science at FORTH, in Greece[7].

RQL adopts the syntax of OQL [Cattel et al., 2000]. As OQL, RQL is a functional language: the output of RDF Schema queries is again legal RDF Schema code, which allows the output of queries to function as input for subsequent queries.

RQL is defined by means of a set of core queries, a set of basic filters, and a way to build new queries through functional composition and iterators.

The core queries are the basic building blocks of RQL, which give access to the RDF Schema specific contents of an

RDF triple store, with queries such as *Class* (retrieving all classes), *Property* (retrieving all properties) or *Writer* (returning all instances of the class with name *Writer*). This last query returns of course also all instances of subclasses of *Writer*, since these are also instances of the class *Writer*, by virtue of the semantics of RDF Schema. We can ask for all *direct* instances of *Writer* (i.e. ignoring all instances of subclasses) through the query *Writer*.

RQL can also query the structure of the subclass hierarchy. In our example, the query *subClassOf(Writer)* would return the class *FamousWriter* as its only result. In general, this would return all direct and indirect subclasses of *Writer*, since RQL is aware of the transitivity of the subclass relation. The query *subClassOfˆ(Writer)* would return only the immediate subclasses.

Of course, being based on OQL, RQL also allows a select-from-where construct. A final crucial feature of RQL are the path-expressions. These allow us to match patterns along entire paths in RDF/RDF Schema graphs, such as the one depicted in Figure 7.2. For example, the query

SELECT Y FROM FamousWriter *fXg*. hasWritten *fYg* returns all books written by famous writers, effectively doing pattern-matching along a path in the graph of Figure 7.2.

3.4. Conclusion

The previous subsections have argued that RDF data should not be queried at the level of their (rather incidental) XML encoding, and that RDF Schema data should not be regarded as simply a set of RDF triples, since all intended semantics of the RDF Schema primitives are then lost. Consequently, we should be using a query language that is sensitive to this RDF Schema semantics. RQL is a powerful (and currently the only) candidate for such a language.

In the next few sections, we will discuss the architecture we have designed for a query engine for RQL.

4. Sesame's Architecture

The Sesame system is a Web-based architecture that allows persistent storage of RDF data and schema information and subsequent online querying of that information. In section 4.1, we present an overview of Sesame's architecture. In the sections following that, we look in more detail at several components.

4.1. Overview

An overview of Sesame’s architecture is shown in Figure 7.3. In this section we will give a brief overview of the main components.

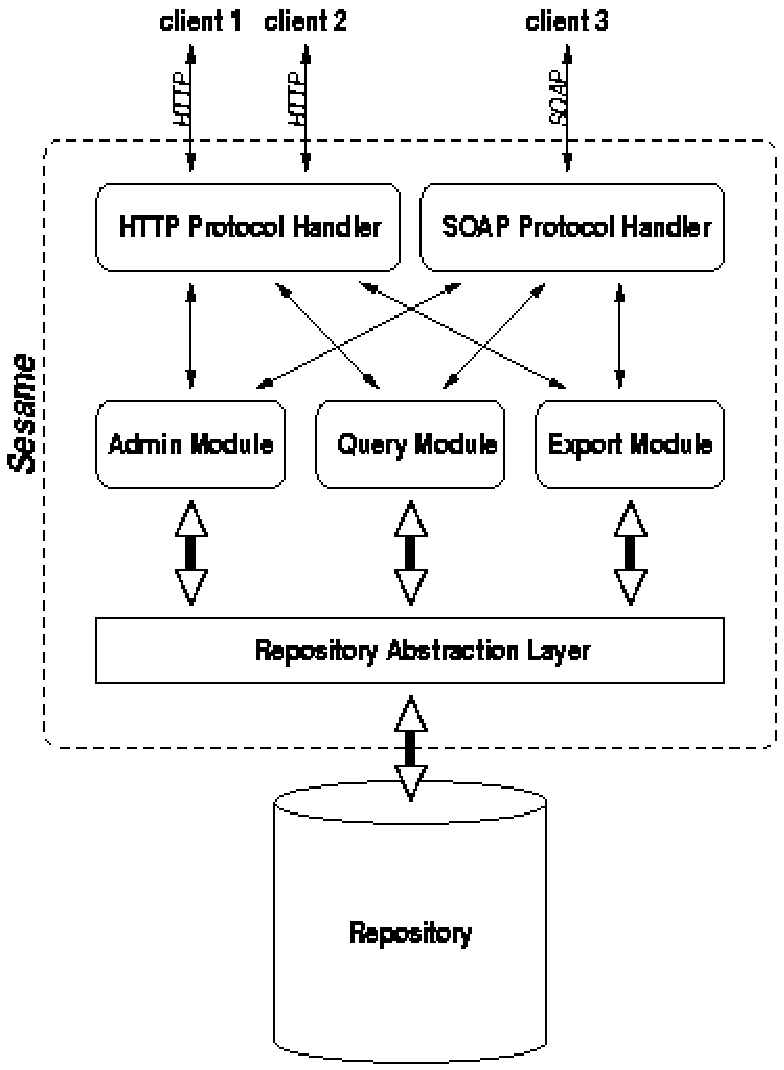


Figure 7.3 Sesame’s architecture

For persistent storage of RDF data, Sesame needs a scalable repository. Naturally, a Data Base Management System (DBMS) comes to mind, as these have been used for decades for storing large quantities of data. In these decades, a large number of DBMS’s have been developed, each having their own strengths and weaknesses, targeted platforms, and API’s. Also, for each of these DBMS’s, the RDF data can be stored in numerous ways.

As we would like to keep Sesame DBMS-independent and it is impossible to know which way of storing the data is best fitted for which DBMS, all DBMS-specific code is concentrated in a single architectural layer of Sesame: the *Repository Abstraction Layer* (RAL).

This RAL offers RDF-specific methods to its clients and translates these methods to calls to its specific DBMS. An important advantage of the introduction of such separate layer is that it makes it possible to implement Sesame on top of a wide variety of repositories without changing any of Sesame’s other components. Section 4.3 describes a number of possible repository implementations.

Sesame’s functional modules are clients of the RAL. Currently, there are three such modules:

- The RQL query module.

This module evaluates RQL queries posed by the user (see Section 5.1).

- The RDF administration module.

This module allows incremental uploading of RDF data and schema information, as well as the deleting of information (see Section 5.2).

- The RDF export module.

This module allows the extraction of the complete schema and/or data from a model in RDF format (see Section 5.3).

Depending on the environment in which it is deployed, different ways to communicate with the Sesame modules may be desirable. For example, communication over HTTP may be preferable in a Web context, but in other contexts protocols such as RMI (Remote Method Invocation)[8] or SOAP (Simple Object Access Protocol) [Box et al., 2000] may be more suited.

In order to allow maximal flexibility, the actual handling of these protocols has been placed outside the scope of the functional modules. Instead, protocol handlers are provided as intermediaries between the modules and their clients, each handling a specific protocol.

The introduction of the repository abstraction layer and the protocol handlers makes Sesame into a generic architecture for RDF (S) storage and querying, rather than just a particular implementation of such a system. Adding additional protocol handlers makes it easy to connect

Sesame to different operating environments. The construction of concrete RAL’s will be discussed in the next section.

Sesame’s architecture has been designed with extensibility and adaptability in mind. The possibility to use other kinds of repositories has been mentioned before. Adding additional modules or protocol handlers is also possible. The only part that is fixed in the architecture is the RAL.

4.2. The Repository Abstraction Layer

As we have seen in the previous section, the Repository Abstraction Layer (RAL) offers a stable, high level interface for talking to repositories. This RAL is defined by an API that offers functionality to add data to, or to retrieve or delete data from the repository. RAL-implementations translate calls to the API methods into operations on the underlying repository.

Rather than adopting or extending an existing RDF API, such as the “Stanford API” proposed by Sergey Melnik [Melnik, 2000], we have created a completely new API.

The main differences between our proposal and the Stanford API are that:

a. The Stanford API is very much targeted at data that is kept in memory, whereas our API is considerably more “lightweight” as all data is returned one-at-a-time in data streams.

b. Our API supports RDF Schema semantics, such as subsumption reasoning, whereas the Stanford API only offers RDF-related functionality.

The advantage of returning data in streams (point a) is that at any one time only a small portion of the data is kept in memory. This streaming approach is also used in the functional modules, and even in the protocol handlers which give results as soon as they are available. This approach is needed for Sesame to be able to scale to large volumes of data without requiring exceptionally expensive hardware. In fact, Sesame requires close to zero memory for data and only a small amount of memory for the program to run.

This, together with the option of using a remote data store for the repository (see Section 4.3) makes Sesame potentially suitable for use as infrastructure in highly constrained environments such as portable devices.

Of course, reading everything from a repository and keeping nothing in memory seriously hurts performance.

This performance problem can be solved by selectively caching data in memory[9]. For small data volumes it is even possible to cache all data in memory, in which case the repository only serves as a persistent storage. Sesame’s architecture allows all of this to be done in a completely transparent way, as will be shown in the next section.

4.2.1. Stacking Abstraction Layers

An important feature of the RAL is that it is possible to put one on top of the other. To Sesame’s functional modules (the admin, query and export modules) this is completely transparent, as they will only see the RAL at the top of the stack (see Figure 7.4). The RAL at the top can perform some action when the modules make calls to it, and then forward these calls to the RAL beneath it. This process continues until one of the RALs finally handles the request.

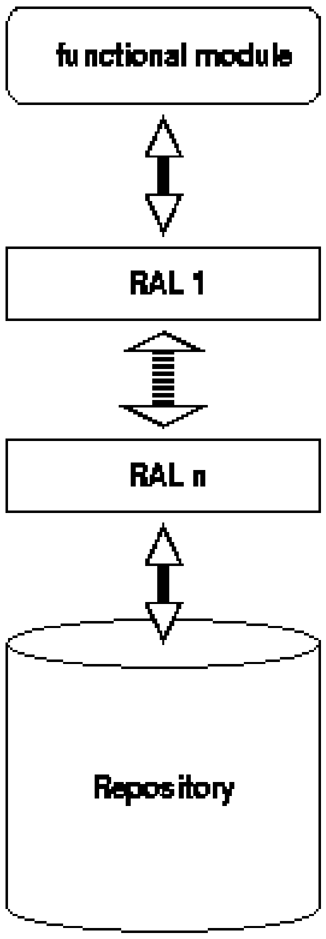


Figure 7.4 RALs can be stacked to add functionality

One good example where this construction makes sense is when implementing a cache. We implemented a RAL that caches all schema data in a dedicated data structure in main memory. This schema data is often very limited in size and is requested very frequently. At the same time, the schema data is the most difficult to query from a DBMS because of the transitivity of the subClassOf and subPropertyOf properties. This schema-caching RAL can be placed on top of arbitrary other RALs, handling all calls concerning schema data. The rest of the calls are forwarded to the underlying RAL.

4.3. The Repository

Thanks to the Repository Abstraction Layer, Sesame can be based on any kind of repository that is able to store RDF. The following is a list of possible concrete implementation of the repository, each with their own advantages.

- DMBS’s

Any kind of database can be used: relational databases (RDBMS), object-relational databases (ORDBMS), etc.

- Existing RDF stores

A number of RDF stores are currently in development ([Guha, 2001, Reggiori, 2001, Beckett, 2001, Wagner, 2001]). Sesame can use such an RDF store if a RAL is written that knows how to talk to that specific RDF store.

- RDF files

Files containing RDF can be used as repositories too. A flat file is not very practical on its own, as it will be painfully slow in storing and retrieving data. However, when combined with a RAL that caches all of the data in memory it becomes a good alternative for small volumes of data.

- RDF network services

Apart from performance, there is no need for the repository to be located close to Sesame. Any network service that offers basic functionality for storing, retrieving and deleting RDF data can be used by Sesame. An example of a system offering such functionality is, of course, Sesame itself. Many of the RDF stores mentioned above can also be approached as Web services.

The last option in particular is very interesting. An initial query is sent to a Sesame server somewhere on the Web. This server can use not only its local repository to answer the query, but also any number of remote repositories that it knows about. In turn, some of these remote repositories might themselves either answer the query using local data-stores, or in turn again approach yet other remote repositories. This opens up the possibility of a highly distributed architecture for RDF(S) storing and querying, that has been unexplored until now, but that is truly in the spirit of the Semantic Web.

4.3.1. PostgreSQL

The first and, so far, only repository that has been used with Sesame is PostgreSQL[10]. PostgreSQL is a freely available (open source) object-relational DBMS that supports many features that normally can only be found in commercial DBMS implementations.

One of the main reasons for choosing PostgreSQL is that it is an object-relational DBMS, meaning that it supports subtable relations between its tables. As these subtable relations are also transitive, we use these to model the subsumption reasoning of RDF Schema.

The RAL that was implemented uses a dynamic database schema that was inspired by the schema shown in [Karvounarakis et al., 2000]. New tables are added to the database whenever a new class or property is added to the repository. If a class is a subclass of other classes, the table created for it will also be a subtable of the tables for the superclasses. Likewise for properties being subproperties of other properties. Instances of classes and properties are inserted as values into the appropriate tables. Figure 7.5 gives an impression of the contents of a database containing the data from Figure 7.2.

The actual schema involves one more table called *resources.* This table contains all resources and literal values, mapped to a unique ID. These ID’s are used in the tables shown in the figure to refer to the resources and literal values. The *resources* table is used to minimize the size of the database. It ensures that resources and literal values, which can be quite long, only occur once in the database, saving potentially large amounts of memory.

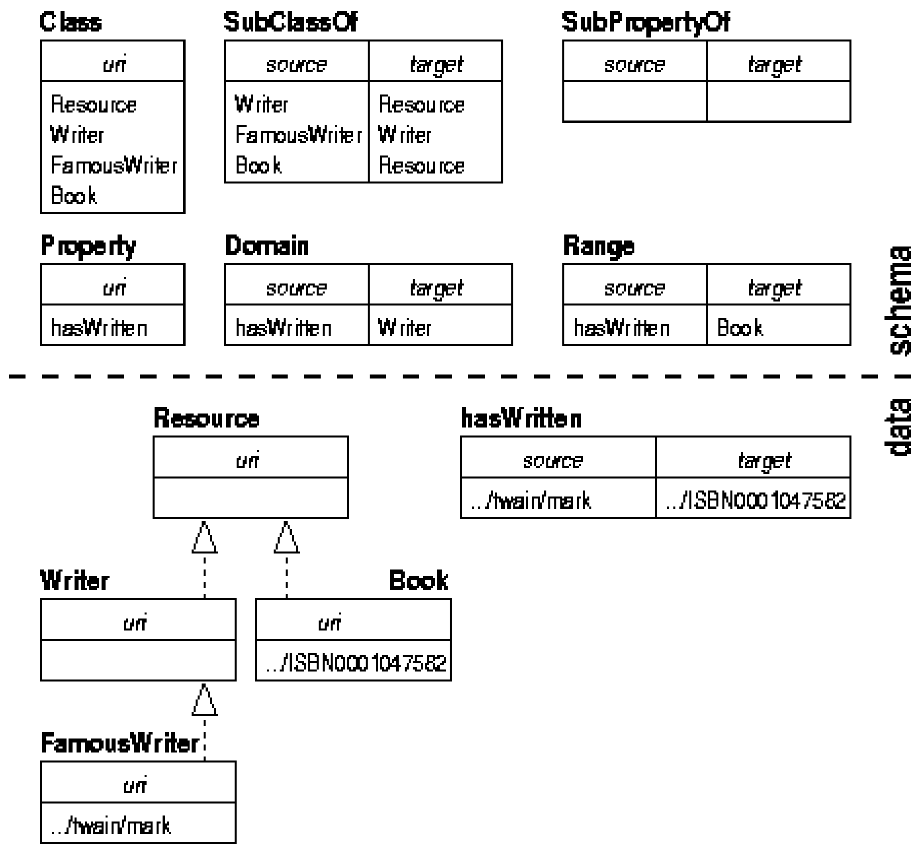


Figure 7.5 Impression of the object-relational schema currently used with PostgreSQL

5. Sesame’s Functional Modules

In this section, we briefly describe the three modules that are currently implemented in Sesame.

5.1. The RQL Query Module

As we have seen in Section 4, one of the three modules currently implemented in Sesame is an RQL query engine. RQL [Karvounarakis et al., 2000, Alexaki et al., 2000] is a proposal for a declarative language for RDF and RDF Schema. It is being developed within the European IST project C-Web and its followup project MESMUSES by the Institute of Computer Science at FORTH, in Greece.

In Sesame, a version of RQL was implemented that is slightly different from the language proposed by [Karvounarakis et al., 2000]. The Sesame version of RQL features better compliance to W3C specifications, including support for optional domain- and range restrictions as well as multiple domain- and range restrictions. See [Broekstra and Kampman, 2001] for details. The Query Module follows the path depicted in Figure 7.6 when handling a query. After parsing the query and building a query tree model for it, this model is fed to the query optimizer which transforms the query model into an equivalent model that will evaluate more efficiently.

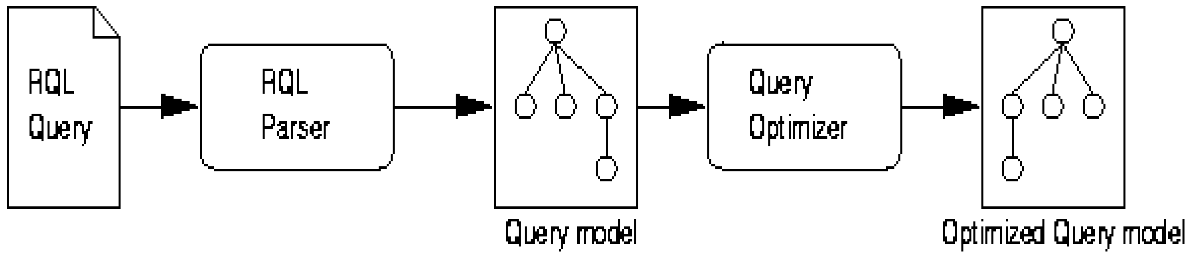


Figure 7.6 A query is parsed and then optimized into an query object model

The optimized model of the query is subsequently evaluated in a streaming fashion, following the tree structure into which the query has been broken down. Each object represents a basic unit in the original query and evaluates itself, fetching data from the RAL where needed. The main advantage of this approach is that results can be returned in a streaming fashion, instead of having to build up the entire result set in memory first.

In Sesame, RQL queries are translated (via the object model) into a set of calls to the RAL. This approach means that the main bulk of the actual evaluation of the RQL query is done in the RQL query engine itself.

For example, when a query contains a join operation over two subqueries, each of the subqueries is evaluated, and the join operation is then executed by the query engine on the results.

Another approach would be to directly translate as much of the RQL query as possible to a query specific for the underlying repository. An advantage of this approach is that, when using a DBMS, we would get all its sophisticated query evaluation and optimization mechanisms for free. However, a large disadvantage is that the implementation of the query engine is directly dependent on the repository being used, and the architecture would lose the ability to easily switch between repositories.

This design decision is one of the major differences between Sesame and the RDF Suite implementation of RQL by ICS-FORTH (see [Alexaki et al., 2000]). The RDF Suite implementation relies on the underlying DBMS for query optimisation. However, this dependency means that RDF Suite cannot as easily be transported to run on top of another storage engine.

A natural consequence of our choice to evaluate queries in the RAL is that we need to devise several optimization techniques in the engine, since we cannot rely on any given DBMS to do this for us. The Admin Module In order to be able to insert RDF data and schema information into a repository, Sesame provides an admin module. The current implementation is rather simple and offers two main functions:

1. incrementally adding RDF data/schema information;
2. clearing a repository.

The admin module retrieves its information from an RDF(S) source (usually an online RDF(S) document in XML-serialized form) and parses it using a streaming RDF parser (currently, we use the SiRPAC RDF parser [Barstow and Melnik, 2000]). The parser delivers the information to the admin module on a per-statement basis: *(S; P;O).* The admin subsequently checks each statement for consistency with the information already present in the repository, and infers implied information if necessary, as follows: if *P* equals *type*, then the admin infers that *O* must be a class.

- if P equals *subClassOf*, then the admin infers that both *S* and *O* are classes.
- if *E* equals *subPropertyOf*, then the admin infers that both *S* and *O* are properties.
- if *P* equals *domain* or *range*, the admin infers that *S* must be a property and *O* must be a class.

In all these cases, the admin module checks whether the inferred information is consistent with the current contents of the repository. and if so, the inferred information is added to the repository. If the admin module encounters a duplicate statement (i.e. a fact that is already known in the repository),this is reported but otherwise ignored.

5.2. The RDF Export Module

The RDF Export Module is a very simple module. This module is able to export the contents of a repository formatted in XML-serialized RDF. The idea behind this module is that it supplies a basis for using Sesame in combination with other RDF tools, as all RDF tools will at least be able to read this format. Some tools, like ontology editors, only need the schema part of the data. On the other hand, tools that don’t support RDF Schema semantics will probably only need the non-schema part of the data. For these reasons, the RDF Export Module is able to selectively export the schema, the data, or both.

6. Experiences

Our implementation of Sesame can be found at http://sesame.aidministrator.nl (July 2001), and is freely available for non-commercial use. This implementation follows the general architecture described in this paper, using the following concrete implementation choices for the modules:

- As discussed above, the repository is realised by PostgreSQL.
- A protocol handler is realised using HTTP.
- The admin module uses the SiRPAC RDF parser.
- In this section, we briefly report on our experiences with various aspects of this implementation.

6.1. Using RQL

As we have seen in Section 5.1, Sesame supports querying using a declarative language called RQL. RQL is very powerful language that offers very expressive querying capabilities. One of the most distinguishing features of RQL is its built-in support for RDF Schema semantics and the possibility to combine data and schema information in a single query. However, RQL currently lacks support for semantically querying *reified* statements. The reason for this is mainly that reification is poorly defined in the RDF specification. The direct result of the lack of support is that it is not possible to query such constructs semantically. When confronted with reified statements, RQL queries will have to be formulated in terms of the structure of such a statement.

6.2. Application: On-To-Knowledge

Sesame is currently being deployed as the central infrastructure of the European IST project On-To-Knowledge[11]. On-To-Knowledge aims at developing ontology-driven knowledge-management tools. Figure 7.7 shows how Sesame serves as the central data repository for a number of tools:

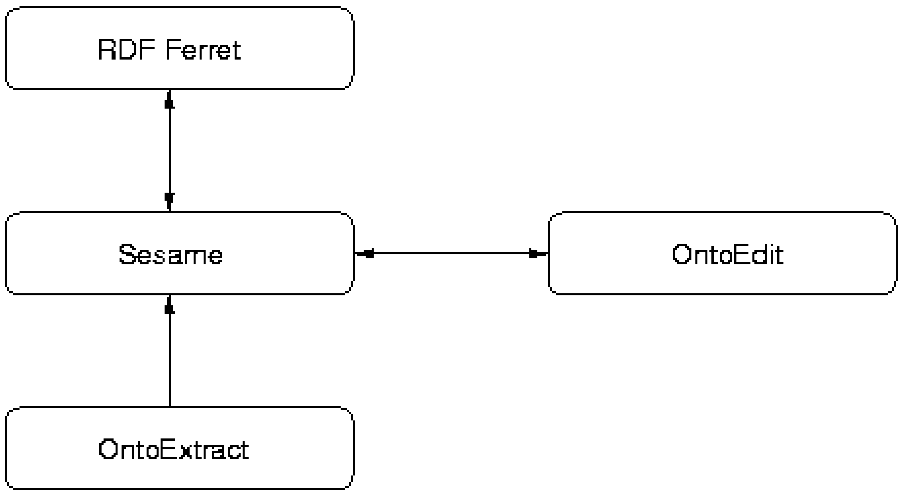


Figure 7.7 Sesame is positioned as a central tool in the On-To-Knowledge project

- OntoExtract, developed by CognIT a.s., extracts ontological conceptual structures from natural language documents. These ontologies are uploaded for storage in Sesame.
- The resulting ontologies can be downloaded into OntoEdit, an editor for ontologies developed by the the Institute

AIFB of the University of Karlsruhe. When the user has edited an ontology, the result isagain stored in Sesame.

- The resulting ontologies are downloaded into RDF Ferret, a user front-end, developed by BT Adastral Park Research Labs, that provides search and query facilities for webdata based on the ontologies.

Because Sesame is a server-based application, the integration of all this functionality is done simply by establishing HTTP connections to Sesame. We are currently in the process of applying this architecture in a number of knowledge-management applications.

6.3. Ontologies and RDF Schema

While developing Sesame, many unclarities in the RDF Schema specification were uncovered. One of the reasons for this is that RDF Schema is defined in natural language: no formal description of its semantics is given. As a result of this, the RDF Schema specification even contains some inconsistencies.

Another reason why RDF Schema is so hard to understand is that RDF Schema is self-describing in the sense that the definition of its terms is itself done in RDF Schema. This leads to strange circular dependencies in the term definitions (e.g. the term *Class* is both a subclass of and an instance of *Resource*,which is itself an instance of *Class* again). In fact, primitives from different meta-levels of description have been mapped to identical terms, resulting in a rather unclear specification (see also [Nejdl et al., 2000]).

One of the consequences of the circular dependencies is that RDF Schema is not only a language for,but also a part of ontologies. This means that all primitives defined in RDF Schema (i.e. *subClassOf, sub-PropertyOf, domain, range*, etc.) are also in the ontology. We would argue that this is counterintuitive.

At the very least this approach deviates from approaches taken by most other ontology languages.

6.4. Using PostgreSQL

Our experiences with the database schema on PostgreSQL, as shown in Section 4.3.1, are not completely satisfactory. Both data retrieval and data insertion are not as fast as we would like. Especially incremental uploads of schema data can be very slow, since table creation is very expensive in PostgreSQL. Even worse,when adding a new subClassOf relation between two existing classes, the complete class hierarchy starting from the subclass needs to broken down and must then be rebuilt again because subtable relations can not be added to an existing table; the subtable relations have to be specified when a table is created. Once created, the subtable relations are fixed.

6.5. Scalability issues

We have been experimenting with several data sets and/or ontologies that are currently available on the Web. The largest set of data that we have uploaded and subsequently queried was the collection of nouns from Wordnet[12], consisting of about 400,000 RDF statements. This data set almost completely exists of RDF data (i.e. hardly any schema information). While we have not performed any structured benchmark testing, the following points are noteworthy.

First of all, all experimenting has been done using a desktop computer (Sun UltraSPARC 5 workstation, 256MB) to run Sesame. Java Servlets running on a web server were used as protocol handlers to communicate over HTTP. The database schema described in Section 4.3.1 in combination with PostgreSQL version 7.1.1 was used as repository.

The uploading of the information is not as fast as we would like, mainly due to the database schema being used. Just adding a data statement to the database involves doing the following steps:

- Check whether the property is already known. If not, add it and create a table for it.
- Check whether the subject is already known, adding it if not.
- Check whether the object is already known, adding it if not.
- Add a row representing the statement to the appropriate table.

Most of these steps have to be performed in sequential order, which is very time-intensive. Uploading the Wordnet nouns took approximately 94 minutes, which comes down to 71 statements per second. As was to be expected, the upload did not show any significant signs of slowing down as the amount of data in the repository increased (the amount of data is really not very large by DBMS standards).

Querying the information proved to be quite slow too, for exactly the same reasons. Due to the distributed storage over multiple tables, retrieving data from the repository means doing many joins on tables, hindering performance.

7. Future directions

7.1. DAML+OIL

Currently, Sesame understands the semantics of RDF and RDF Schema. We would like to extend this to more powerful languages like DAML+OIL [Horrocks et al., 2001]. DAML+OIL is an extension of RDF Schema and offers additional primitives for creating schemata. Examples of the additional expressive power of DAML+OIL are:

- the use of arbitrary class expressions, including disjunction, conjunction and negation (complement) of classes
- cardinality constraints on properties, expressing the minimal and maximal number of values a property can have for each object
- symmetric, transitive and inverse properties

Since DAML+OIL allows more expressiveness and has more inferencing capabilities, a reasoner/query language that understands its semantics is significantly more complicated.

7.2. Other repositories

We are planning to implement support for other DMBS/schema combinations so that we can compare the pros and cons of each approach. A first option will be to implement a RAL based on a traditional relational DBMS, i.e. one that only uses standard SQL queries. Such a RAL can be used on lots of DBMS's as almost all DBMS's support these queries.

7.3. Admin Module

The Admin Module currently offers very limited functionality for administrating the contents of repositories. More fine-grained functionality is needed for the module to be really useful. We are currently investigating the options to accomplish this. One of the options is to extend RQL with primitives for updating and deleting data, just like SQL does.

Conclusion

In this paper we have presented Sesame, a flexible architecture for storing and querying both RDF data and RDF Schema information. Sesame is an important step beyond the currently available storage and query devices for RDF, since it is the first publicly available implementation of a query language that is aware of the RDF Schema semantics.

An important feature of the Sesame architecture is its abstraction from the details of any particular repository used for the actual storage. This makes it possible to port Sesame to a large variety of different repositories, including relational databases, RDF triple stores, and even remote storage services on the Web.

Sesame itself is a server-based application, and can therefore be used as a remote service for storing and querying data on the Semantic Web. As with the storage layer, Sesame abstracts from any particular communication protocol, so that Sesame can easily be connected to different clients by writing different protocol handlers.

We have constructed a concrete implementation of the generic architecture, using PostgreSQL as a repository and using HTTP as communication protocol handlers.

Important next steps to expand Sesame towards a full fledged storage and querying service for the Semantic Web include its extension from RDF Schema to DAML+OIL and implementations for different repositories, notably those that can live elsewhere on the Web.

References

[Alexaki et al., 2000] Alexaki, S., Christophides, V., Karvounarakis, G., Plexousakis, D., and Tolle, K.(2000). The RDFSuite: Managing Voluminous RDF Description Bases. Technical report, Institute of Computer Science, FORTH, Heraklion, Greece. See http://www.ics.forth.gr/proj/isst/RDF/RSSDB/rdfsuite.pdf.

[Barstow and Melnik, 2000] Barstow, A. and Melnik, S. (2000). SiRPAC: Simple RDF Parser and Compiler. http://www.w3.org/RDF/Implementations/SiRPAC/.

[Beckett, 2001] Beckett, D. (2001). Redland RDF Application Framework. http://www.redland.opensource.ac.uk/.

[Box et al., 2000] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F.,Thatte, S., and Winer, D. (2000). Simple Object Access Protocol (SOAP) 1.1. W3c note, World Wide Web Consortium. See http://www.w3.org/TR/SOAP/.

[Brickley and Guha, 2000] Brickley, D. and Guha, R. (2000). Resource Description Framework (RDF) Schema Specification 1.0. Candidate recommendation, World Wide Web Consortium. See http://www.w3.org/TR/2000/CR-rdf-schema-20000327.

[Broekstra and Kampman, 2001] Broekstra, J. and Kampman, A. (2001). Query Language Definition. On-To-Knowledge (IST-1999-10132) Deliverable 9, Aidministrator Nederland b.v. See http://www.ontoknowledge.org/.

[Cattel et al., 2000] Cattel, R., Barry, D., Berler, M., Eastman, J., Jordan, D., Russell, C., Schadow, O.,Stanienda, T., and Velez, F. (2000). The Object Database Standard: ODMG 3.0. Morgan Kaufmann.

[Chamberlin et al., 2001] Chamberlin, D., Florescu, D., Robie, J., Simeon, J., and Stefanescu, M. (2001). XQuery: A Query Language for XML. Working draft, World Wide Web Consortium. See http://www.w3.org/TR/xquery/.

[Fensel et al., 2000] Fensel, D., van Harmelen, F., Klein, M., Akkermans, H., Broekstra, J., Fluit, C., van der Meer, J., Schnurr, H.-P., Studer, R., Hughes, J., Krohn, U., Davies, J., Engels, R., Bremdal, B., Ygge, F., Lau, T., Novotny, B., Reimer, U., and Horrocks, I. (2000). On-To-Knowledge: Ontologybased Tools for Knowledge Management. In Proceedings of the eBusiness and eWork 2000 (EMMSEC 2000) Conference, Madrid, Spain.

[Guha, 2001] Guha, R. (2001). rdfDB. http://web1.guha.com/rdfdb/.

[Horrocks et al., 2001] Horrocks, I., van Harmelen, F., Patel-Schneider, P., Berners-Lee, T., Brickley, D., Connoly, D., Dean, M., Decker, S., Fensel, D., Hayes, P., Heflin, J., Hendler, J., Lassila, O., McGuinness, D., and Stein, L. A. (2001). DAML+OIL. http://www.daml.org/2001/03/daml+oil-index.html.

[Karvounarakis et al., 2000] Karvounarakis, G., Christophides, V., Plexousakis, D., and Alexaki, S.(2000). Querying community web portals. Technical report, Institute of Computer Science, FORTH, Heraklion, Greece. See http://www.ics.forth.gr/proj/isst/RDF/RQL/rql.pdf.

[Lassila and Swick, 1999] Lassila, O. and Swick, R. R. (1999). Resource Description Framework (RDF): Model and Syntax Specification. Recommendation,WorldWideWeb Consortium. See http://www.w3.org/TR/REC-rdf-syntax/.

[Melnik, 2000] Melnik, S. (2000). RDF API Draft. public draft, Database Group, Stanford University. See http://www-db.stanford.edu/˜melnik/rdf/api.html.

[Nejdl et al., 2000] Nejdl, W., Wolpers, M., and Capella, C. (2000). The RDF Schema Revisited. In Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik, Modellierung 2000, St. Goar. Foelbach Verlag, Koblenz.

[Reggiori, 2001] Reggiori, A. (2001). RDFStore. http://rdfstore.jrc.it/.

[Wagner, 2001] Wagner, H. (2001). Extensible open rdf. http://eor.dublincore.org/.

Footnotes

[1]Aidministrator Nederland b.v., jeen.broekstra@aidministrator.nl

[2]Aidministrator Nederland b.v., arjohn.kampman@aidministrator.nl

[3]Faculty of Sciences, Vrije Universiteit Amsterdam, frank.van.harmelen@cs.vu.nl

[4]See http://www.aidministrator.nl/

[5]See http://wwww.ontoknowledge.org/

[6] rei facere (lat.) “to make into a thing”

[7] See http://www.ics.forth.gr/

[8] See http://java.sun.com/j2se/1.3/docs/guide/rmi/spec/rmiTOC.html

[9]A good DBMS implementation will also cache query results to improve performance

[10]See http://www.postgresql.org/

[11] http://www.ontoknowledge.org/

[12] This collection can be found in RDF form at http://www.semanticweb.org/library/

8. Enabling Task-Centered Knowledge Support through Semantic Markup

Rob Jasper[1] and Mike Uschold[2]

1. The Evolving Web

The current evolution of the web can be characterized from various perspectives:

- Locating Resources—The way people find things on the web is evolving from simple keyword search to more sophisticated semantic techniques both for search and navigation.
- Users—Web resources are evolving from being primarily intended for human consumption, to being intended for use both by humans and machines—most notably, software agents.
- Semantics—The web is evolving from containing information resources that have little or no explicit semantics, to having a rich semantic infrastructure.

There is considerable activity going on, and significant progress being made in each of these areas, by various communities, including research, standards organizations, and commercial interests.

Locating Resources. Commercial companies have made great progress in developing search engines for the web, primarily focusing on keyword-based search and text categorization techniques.

Users. More and more, software programs are accessing and processing web pages automatically. Meta-search engines are one good example. There is a growing number of B2C web sites who operate in a similar manner, by checking and comparing many different web sites—e.g. for selling books, or online travel. This works fine, but often requires customization by humans to figure out what the information on a site 'means'. By looking at the text, and analyzing the underlying structure of the web site, it is possible to write CGI scripts, perform some computation, and collate results back to the user. For various reasons, these sites are not a panacea. But more importantly, perhaps is the fact that there is no systematic and agreed way to represent and interpret the semantics of the information. This brings us to the semantic web.

The Semantic Web. There is a tremendous amount of activity by the W3C and research groups in addressing the issue of semantics. From the perspective of the W3C, the idea of a semantic web is to make the information machine-sensible. The research community is working with the W3C to create the dream of a semantic web (Berners-Lee *et al* 2001, DAML 2001). A major part of this will entail representing and reasoning with semantic metadata, and/or providing semantic markup in the information resources. A key benefit is that this will make it much easier and much faster for humans to find what they require, say compared to today's 'non-semantic' search engines. However, this remains an unproven and largely untested hypothesis. The main intent of the semantic web is to give machines much better access to information resources so they can be information intermediaries in support of humans.

Currently, much semantic markup has focused on providing common languages and infrastructure to enhance information retrieval and support e-business applications. While we believe this work is important, these applications often fail to recognize or support the underlying tasks users are performing. Users are seldom just searching for documents but are in the process of performing some larger task (e.g., selling a car, moving to a new city).

Consider the interface to most information retrieval or knowledge management systems. These interfaces typically capture a set of keywords, or at best, a complex Boolean query, but fail to identify the task a user is performing. Understanding the user and their underlying task is critical in determining which resources are most appropriate. For example, a system that understood whether a user’s task was purchasing, selling, or repairing a car would be much better at supporting the user’s ultimate goal. Other researchers have identified the need for understanding the underlying task a user if performing [Dumais, 1999]; but little work has been done on capturing and exploiting this information.

A place to do things. In his presentation at the Knowledge Technologies 2001 conference, Reid Smith, VP of knowledge management at Slumberger introduced another perspective from which to view the evolution of the web.

Web tasks. The web will evolve from being primarily a place to *find* things, to being a place to *do* things as well [Smith, 2001].

This perspective is the focus of this paper. We describe some important first steps in this evolution of the web into a place to perform tasks. Provision of semantics for the web is a necessary foundation on which to build our task-centered approach. It enables new kinds of applications that better support a user’s broader goals. We describe a semantics-based prototype focused on supporting user’s tasks in the domain of aerospace customer support.

2. Web Problem Solving

People use the web in a variety of contexts to solve many different kinds of problems. The effectiveness of task-centered knowledge support relies on identifying a convenient mechanism for expressing the user’s intentions and matching those intentions with the relevant resources and integrating them into a consistent view. In the short run, this requires that the application anticipate the kinds of problems that users want to solve and that those problems occur with some degree of regularity.

In this section we note some key distinctions or dimensions that define a notional space of web problem solving (see Figure 8.1). Different points and regions in this space require different techniques. We identify two dimensions of variation: frequency of occurrence and how easy it is to anticipate a specific task that some user might have. Our task-centered approach supports a space of problems that occur with some regularity and that are possible, but not easy to anticipate.

2.1. Dimensions of variation

Anticipatability. This dimension represents a continuum ranging from very easy to anticipate, to very difficult to anticipate. In the domain of U.S. history, a user may have a simple task of determining who was the 19th president. This task is fairly easy to anticipate. The answer can easily be found using a standard search engine. Tasks that are easy to anticipate will often have existing web resources addressing them. In this case there is a whole web page [IPL, 2001] devoted to the 19th president, Rutherford Birhard Hayes. At the other end of this continuum, are tasks that are virtually impossible to anticipate. An example would be the task of determining whether Rutherford Birhard Hayes is related to AI researcher Pat Hayes. We were unable to locate a web page that had the answer to this question. A task that is somewhere in the middle of this continuum is that of finding out how many vice presidents were older than the president they served. It is rather harder to anticipate this specific question than the first one. However, it is possible to set up a database that contains information about past presidencies which can answer this and a wide variety of other similar questions. Each question is not by itself easy to anticipate.

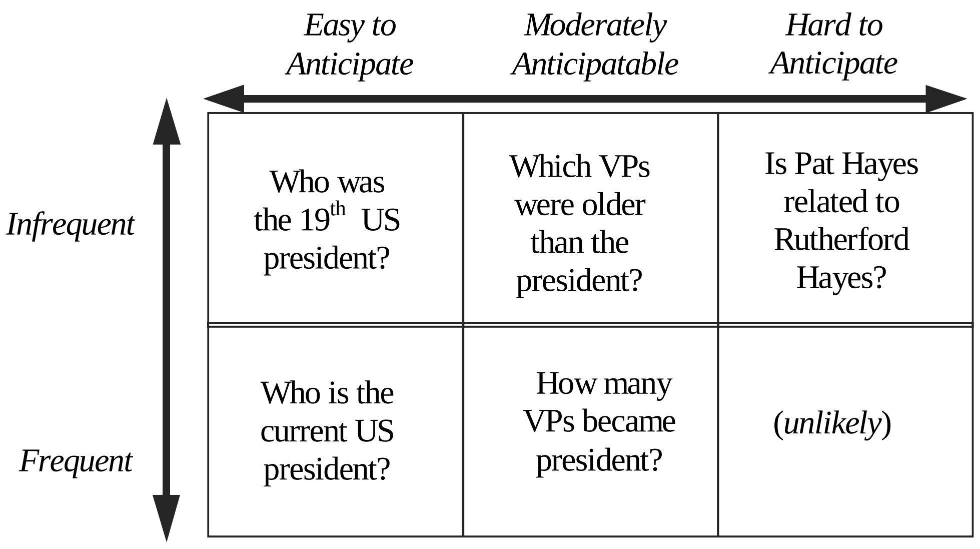

Easy to Anticipate
Moderately Anticipatable
Hard to Anticipate
Infrequent
Frequent
Who was the 19th US president?
Which VPs were older than the president?
Is Pat Hayes related to Rutherford Hayes?
Who is the current US president?
How many VPs became president?
(unlikely)

Figure 8.1 Notional Space of Problems

Frequency—The other dimension represents a continuum of problems ranging from frequently occurring to infrequently occurring. A simple task that may frequently arise would be to find out who the current president is, or what the capitol city is for a country or state.

These dimensions are distinct and largely orthogonal (see Figure 8.1). Easy to anticipate tasks may arise frequently ("Who is the current president?") or infrequently ("Who is the 19th president?"). Tasks in the middle of the anticipatability continuum, may also arise frequently ("How many vice presidents became president?") or less frequently (Which VPs were older than the president they served?). The dimensions are partially correlated in that frequently arising tasks will often be easier to anticipate. Also, very few tasks that are very difficult to anticipate will arise frequently. By limiting the domain (e.g., U.S. Presidents) you can better anticipate the kinds of tasks or problems people might be working on.

2.2. Supporting Technologies

Figure 8.2 depicts how different technologies may be used to support problem solving in different areas in this space. Some commercial applications attempt to support portions of this space, but in a very limited way. Ask Jeeves (www.ask.com) focuses on leveraging the responses from previous searches. This requires the ability to match natural language queries against categories of questions asked previously by users. Jeeves attempts to answer very general questions, in a virtually unrestricted domain. It is only successful to the extent that similar questions have been asked before. It has limited ability to leverage domain knowledge as the application doesn't restrict the domain at all.

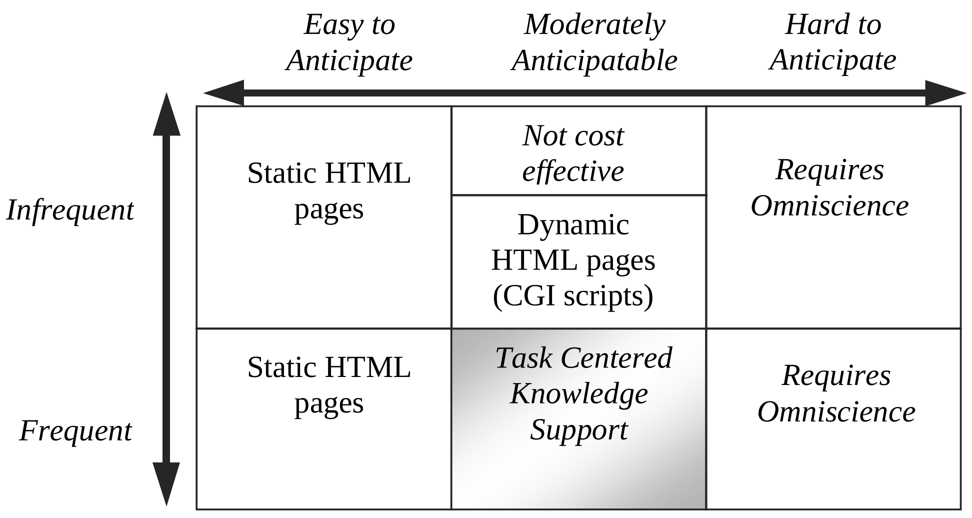


Figure 8.2 Problem Approaches.

The interface encourages the expressions of queries as questions, which may in some instances correspond to the goal of the user. For example, the question, "How do you build a home?" may indicate some interest in the user's desire to build a home. In fact, the results from Jeeves on this question led to better results than the simple query "build home" on a standard search engine site. The Jeeves interface provides some additional insight into the task the user is performing over standard search engines. The query "build home" could also imply that the user is trying to build a home page. The Jeeves interface provides at least some hope of disambiguation between home building and home page building. Nevertheless, the interface doesn't elicit the task the user is performing, only the question they are asking. In some cases these coincide, in other cases they don't. In general, task information cannot be leveraged from a purely question-based interface.

Specific kinds of questions and tasks can easily be supported by standard HTML or dynamic HTML. This is especially if the tasks are sufficiently easy to anticipate that someone has already put together a web page that can help solve it. Suppose the user is trying to determine who was the 19th president of the United States. This is easy to anticipate, which in part explains why someone has built a web page featuring the 19th president Rutherford Birhard Hayes. On the other hand, if your task was to determine if President Hayes was related to Pat Hayes (very difficult to anticipate) you would have a much more difficult time.

Our goal in describing this framework is primarily to state that task-centered knowledge support focuses on the space of frequently asked questions not supported easily by standard static or dynamic HTML interfaces. In the middle area along the anticipatability dimension, we can define spaces that are domain specific and context rich. Limiting the domain provides for simpler mechanism for expression of the underlying task and allows for leveraging of domain knowledge. Boeing's Service Engineering organization provides an example of a context rich specific domain to illustrate the concept of task-centered knowledge support.

3. The Domain

In this section, we give an overview of the domain of aerospace customer support and describe a problem-solving scenario that we are supporting.[3]

3.1. Aerospace Customer Service

Airline operators require assistance from manufacturers in solving service-related problems with their airplanes. There is a constant stream of incoming requests that need to be handled. Solving these kinds of problems requires significant human and computational infrastructure. People handling these requests rely on dozens of different resources, scattered throughout the company. In addition, they must also coordinate their work with several other people:

The customer: people responsible for maintenance and safety of the aircraft they operate.

Federal Aviation Administration (FAA) officials and representatives responsible for safety and certification of aircraft in the U.S.

- Service Representatives (SR) functioning as first line of support to customers, typically located onsite with the airline.
- Help Desk workers responsible for coordinating activities and making sure that the requests are given to the appropriate people for efficient and effective resolution.

- Engineers: specialists in aircraft engineering and maintenance, including the original designers.
- A typical flow of events is:

1. Airline customers experiencing some problem with their airplane contact an onsite service representative (SR). Many general and routine problems can be fixed directly by the SR. Nevertheless, they are generalists and often don’t have the knowledge or expertise to solve specialized engineering problems.
2. SRs enter problems requiring specialized knowledge into a help desk system that describes the specific problem. The requests for service are called help desk requests or HD requests, for short.
3. Each HD request is routed to the engineer whose expertise is most appropriate to the given request. The engineer will coordinate their activities with other engineers as appropriate. The engineers also rely on a number of paper and online resources in responding to the HD request. Different resources are required for different kinds of problems.
4. Critical or safety related changes that need FAA approval are passed to the appropriate FAA representative.

3.2. Scenario

In this section, we outline a scenario for how a task-centered approach can support aerospace customer service. A key requirement is quick and reliable location of the appropriate engineering resources required to solve the customer's problem. Different kinds of tasks require different resources. The same kind of task will require different resources depending on the nature of the underlying request.

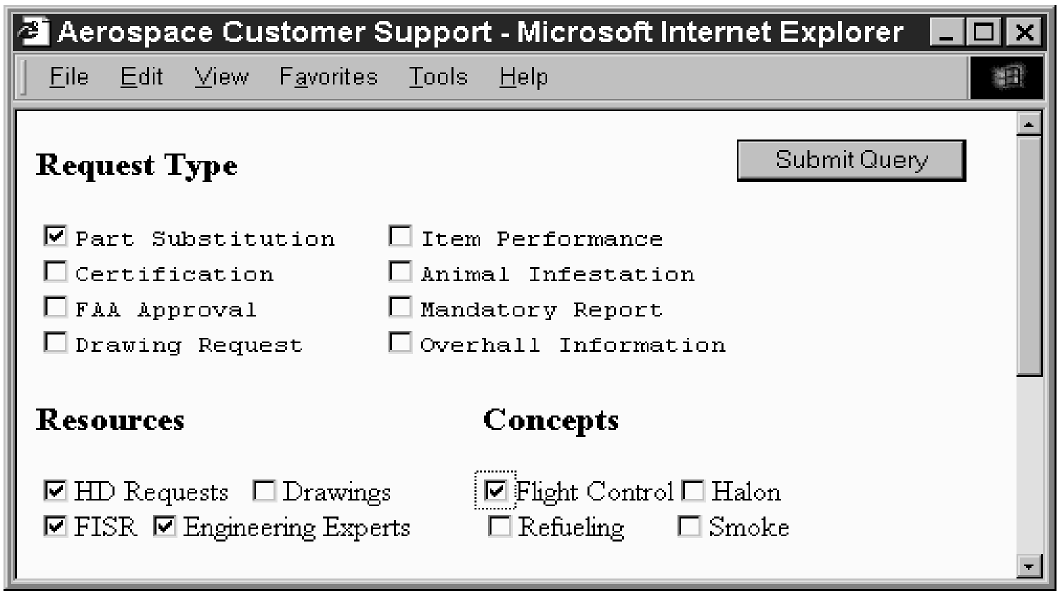
Required resources include engineering databases, design documents, drawings, as well as the appropriate individuals for bringing closure to the problem. A large amount of the expertise, an engineer brings to the table is location of these resources. The kinds of resources needed are dictated by the specific task the engineer is working on. Tasks can be grouped by several factors including, aircraft model, engineering area (e.g., systems, structures, payloads), location (e.g., fuselage, empennage), or problem category. Example problem categories are shown in Figure 8.3 as request types. A few examples are: part substitution, FAA approval, animal infestation and drawing request.

In our prototype, we support location of relevant resources through understanding of the underlying task and key concepts from the original request. For example, an engineer might be working on the task of part substitution in the area of flight control. This requires them to locate related HD

Requests, engineering experts and fleet in service summary reports (FISRs).

To submit such a query, the user fills out a web form (see Figure 8.3). For the prototype, we have enumerated typical problem categories (also known as a request types) and four kinds of resources: HD Requests, Drawings, Fleet Issues Summary Reports (FISRs), and Engineering Experts. We have a small handful of concepts to select from, for demonstration purposes. In the future, we expect that these would come from a controlled vocabulary of concepts.

The user begins by checking boxes that identify the category of task (i.e., request type) they are working on. In some cases, multiple categories are appropriate. Each task category, or combination, has a set of associated resource types that are relevant.



Request Type

[✓ Part Substitution] Item Performance
Certification Animal Infestation
FAA Approval Mandatory Report
Drawing Request Overhall Information

Resources

[✓ HD Requests] Drawings
[✓ FISR] [✓ Engineering Experts]

Concepts

[✓ Flight Control] Halon
Refueling Smoke

Figure 8.3 Query Web Form

The system automatically identifies these through a set of predefined mappings. In the future, we could apply some form of online learning (e.g., relevance feedback) to improve resource selection.

In addition, the user identifies concepts central to the current task (or request) by checking boxes. A future enhancement would be to extract these concepts from the underlying HD request. For demonstration purposes, we used a fixed set of concepts that had to be identified by the user. The selected concepts act as a search filter, so that only relevant resources are returned. At this point, they can press the *submit query* button, to locate the relevant resources.

In addition to these factors, the content of the incoming HD message often contains keywords that help identify the appropriate resources. We believe the combination of these factors and the underlying text of the message can be used to identify the resource relevant to the task the engineer is solving.

4. Enabling Infrastructure

This is the main section in this paper—here we describe the details of our infrastructure and architecture for using semantic metadata to enable task-centered knowledge support. We start by presenting the main components of the architecture, and we discuss some of the reasons for our choices. Section 4.2 gives full details of the information/knowledge repository, and Section 4.3 describes how queries are handled. In the following section, we give a complete worked example.

4.1. Main Components

Figure 8.4 summarizes our prototype architecture for enabling task-centered knowledge support. The two central classes of functions provided by this architecture are:

- *Semantic Metadata Services*—representing, querying, and inferencing over metadata about diverse engineering resources (e.g., databases, documents, engineers)
- *User Interaction*—task identification and presentation of links to appropriate resources.

4.1.1. Semantic Metadata Services

We chose Frame Logic [Kifer, 1990] and Resource Description Framework (RDF) for representing facts, rules and queries about the metadata describing key resources required to solve customer problems. We chose the Simple Logic-based RDF Interpreter (SiLRI) developed at University of Karlsruhe [Decker, 1998] for processing the queries. The RDF Model & Syntax specification [Lassila, 1999] defines a formal set-theoretic data model, which can have several isomorphic representations. The RDF triples representation is used by the SiLRI engine. The underlying technology behind SiLRI has been commercialized by Ontoprise [Ontoprise, 2001].

Concept-based search– Because Boeing is such a large and diverse enterprise, language tends to be specialized. This not only causes problems during verbal communication, but can lead to difficulties in querying for resources (e.g., databases) controlled by another group. To help reduce such problems, we chose to integrate a large aerospace Thesaurus into our architecture [Clark, 2000]. The Thesaurus was developed by the Boeing Technical Libraries for the purpose of classifying and retrieving documents. It has been under development for many years, and consists of a vast network of approximately 37,000 concepts with approximately 100,000 links between them. This Thesaurus has been enhanced and exploited in a knowledge-based search application for locating experts in a given subject area.

This demonstrated that an existing corpus of knowledge could be suitably enhanced and exploited in a context not originally anticipated, *i.e.,* searching for experts, rather than for documents. The Thesaurus can be thought of as a conceptual vocabulary, or "lightweight ontology". Current research at Boeing is focused on continuing to enhance the representational aspects. The idea is to continue to exploit the Thesaurus by improving it as required for a variety of other applications. In doing so, we were faced with the realization that people will resist acceptance of a global vocabulary, and therefore ways must be developed to reap the advantages of a standard vocabulary, while allowing individuals to continue to use their own terms locally. Terms and their associated relations in the thesaurus were automatically converted into RDF syntax for use by SiLRI.

4.1.2. User Interaction

To support interaction with the user, we chose to build a template engine that supports 1) collection of user inputs, 2) processing of those inputs, and 3) generating the content of the output to the user. User input parameters and RDF queries represented in F-logic specify how templates are to be instantiated and expanded by a template engine. The queries are embedded in the un-instantiated template, and are sent to the SiLRI engine for evaluation. Evaluation may result in additional or partially bound, but still unevaluated expressions. The query evaluation drives the template instantiation process, which continues until there are no unevaluated query expressions left in the template. The expanded template contains the *content* of the output to the user, in XML format. Before it is presented to the user, it is first converted to HTML by the browser (IE 5.0) using a pre-defined style sheet, written in XSL (see Figure 8.4).

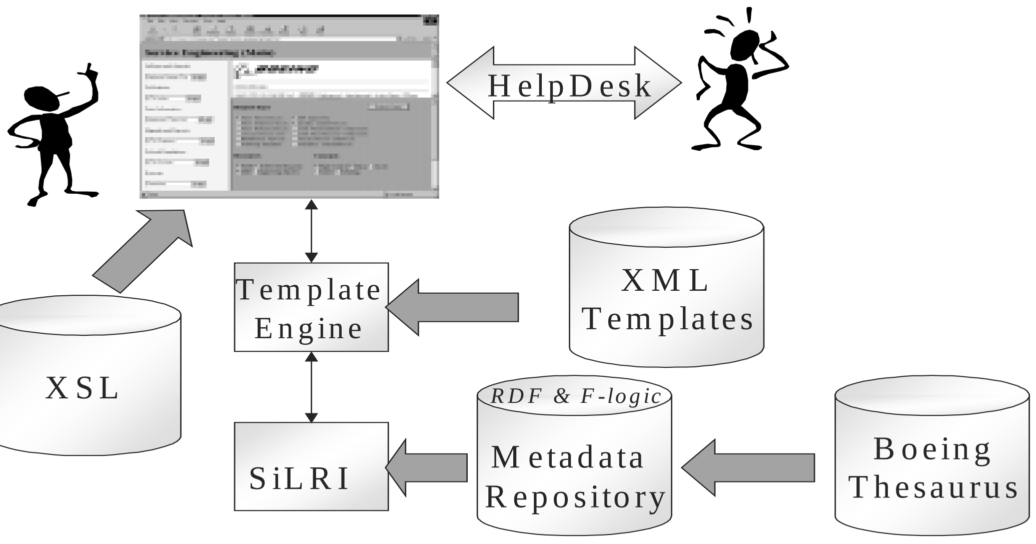


Figure 8.4 Prototype Architecture.

4.1.3. Rationale

The choice of RDF in popularity and provided a means of representing and storing metadata independent from the resources being described. This was an important consideration in the Boeing environment, where central management and control over these resources doesn’t exist. Many of the resources used by the customer support organization are not owned or controlled by them. Other mechanisms for representing meta-data including Topic Maps [TopicMaps, 2001] didn’t have the kinds of support tools required when we started the project. Since then, more tools have been developed to support topic maps. SiLRI provided a convenient mechanism for querying the underlying RDF and, as a bonus, supported representation of and inferencing over rules. This turned out to be an important part of our architecture.

We chose a template-driven approach to the user interface because of its generality and the ability to update the look-and-feel without programming. Originally, the templates contained HTML with embedded queries to the SiLRI engine. After experimenting with this approach for several weeks, we kept wanting to extend the template language to support sorting, filtering, and other common operations. We realized that XSL could support many of the operations we desired without having to build them ourselves. At this point, we chose to use XML as our template language rather than HTML and use the facility in IE 5.0 to translate the XML into HTML using XSL.

4.2. Information/Knowledge Repository

Here we give the formal details of how the information described in the previous section is represented and used. There are various kinds of information that need to be stored and/or referred to in the resource metadata, encoded in F-logic.

- Concepts - the Boeing thesaurus, converted to F-logic syntax.
- Resources - FISRs, HD Requests, Engineering Experts
- Rules - which characterize important relationships, and are used to specify queries.

4.2.1. Concepts

We have translated the Boeing Thesaurus into Frame Logic (F-logic) syntax. F-Logic is a declarative language that mixes features of object-oriented and frame-based languages. It has a model-theoretic semantics and a sound and complete resolution-based proof theory. The main components of F-logic syntax are summarized below:

- *F [S1=>A ; S2=>>B]* declares that
    - *F* is a frame with two slots, *S1* and *S2*
    - slot *S1* has values of frame type *A*
    - slot *S2* is multi-valued and has values of frame type *B*
- *F1::F2* declares that *F1* is a subtype of *F2*.
- *I:F[S1->V; S2->>{V1,V2, …, Vn}]* declares that
    - *I* is an instance of frame type *F*
    - slot *S1* of *I* has value *V*
    - multi-valued slot *S2* of *I* has values *V1–Vn*

Note that double arrows are used for defining the frames (e.g. '=>' or '=>>'). Single arrows are used for defining instances (e.g. '->' or '->>'). Also, two right angle brackets in a row (e.g. '->>' or '=>>') denotes a multi-valued slot, otherwise a slot is single-valued.

Example. A concept from the thesaurus has four slots: name, related_to, broader_than and narrower_than. These are standard for a thesaurus. The definition for a concept, and a few examples are given below.

```
Concept::Object
[Name => String;
 related_to =>> Concept;
 broader_than =>> Concept;
 narrower_than =>> Concept].

halon:Concept
[name -> "halon";
 related_to ->> flight_control]

flight_control:Concept
[name -> "flight control";
 related_to ->> halon]
```

We also define a simple rule for indicating that concepts are near each other. F-Logic rules are written as backward implications, just like Prolog rules.

- near(1, C1, C2) is true when C1 is 'related_to' C2, or C1 is a broader or narrower term than C2.

```
FORALL C1,Y  near(1, C1,C2) <-
   C1[related_to ->> C2]    or
   C1[broader_term  ->> C2] or
   C1[narrower_term ->> C2]
```

By creating rules of this sort, one can specify different degrees of relatedness, which can be exploited to provide greater flexibility to the user in locating resources.

4.2.2. Resources

While it makes sense to refer to the F-logic code as metadata from the perspective of its role in characterizing resources, from the perspective of an F-logic user, this is more likely to be thought of as a knowledge base. Hence, we shall refer to the F-logic KB, which is the metadata for the resource base.

4.2.2.1 Engineering Experts

We define a frame, or class called Person, which has three slots: a name, an employee id, and a list of subjects that they have expertise in. The 'subjects' that fill the latter slot must be of type, 'Concept', another F-logic frame. The F-logic syntax defining the frame Person, and two instances is given below.

```
Person::Object
[Name   => String;
 ID => Integer;
 SubjectsExpertIn =>> Concept].


person1:Person
[Name -> "Robert Jasper";
 ID -> "139988";
 SubjectsExpertIn ->> {refueling, flight_control, smoke}].


person2:Person
```

```
     [Name -> "John Thompson";
      ID -> "42549";
      SubjectsExpertIn ->> {smoke, halon}].
```

For convenience in specifying queries, we define the following predicates. Again, the style is very Prolog-like.

- has_expertise(P, C) is true when P is an instance of Person, and the slot SubjectsExpertIn contains Concept C.
- has_expertise_related_to(P, C) is true when P is an instance of Person, and either P has_expertise in C, or there is another concept, C0, which is 'near' to C (in the concept thesaurus), and P has_expertise in C0.

```
     FORALL P,C has_expertise(P,C) <-
         P:Person[SubjectsExpertIn ->> C].


     FORALL P,C has_expertise_related_to(P,C) <-
         has_expertise(P,C).


     FORALL P,C has_expertise_related_to(P,C) <-
         (EXISTS C0 has_expertise(P,C0) and near(1,C0,C)).
```

4.2.2.2 Fleet Issues Summary Reports (FISRs).

We define a frame called FISR, which has five attributes, a URL, a model number, a title, a list of references that pertain to this FISR, and a set of concepts that this FISR relates to. The last slot has the metadata tag that refers to the resource, which is used when searching for FISRs relevant to the task at hand. In our prototype, this tagging is done manually, but much of the information already exists in another form. Techniques such as wrapper induction [Ashish, 1997], could also be used.

We also define a separate frame for the references. These have four slots, a label, a date, a URL and a location, which indicates where the hard copy of the FISR is physically located. These two frames, with examples are given below, in F-logic.

```
FISR::Object
[URL => String;
 Model => String;
 Title => String;
 References =>> Reference;
 Related_Concepts =>> Concept].
Reference :: Object
[RefLabel => String;
 RefDate  => Date;
 RefURL   => URL;
 RefLocation =>> String].
id_787_0066:FISR
[URL-> "yadayada.boeing.com/cgi?model=787&…";
 Model -> "787";
 Title -> "Lateral Flight Control";
 References       ->> {id_787_8801, id_787_039A};
Related_Concepts->> flight_control].


id_787_8801:Reference
[RefLabel -> "787 8801";
 RefDate  -> "14-JAN-88":String;
 RefLocation ->> {"Joe's Office", "Renton Library"};
 RefURL-> "yadayada.boeing.com/787_8801-fisr-ref.html"].
```

For convenience in specifying queries, we define the following predicates to determine which FISRs are related to a given concept.

- fisr_related_to(C, U, L, M) is true when there is a FISR related to Concept C, and U is the FISR's URL, L [abel] is the FISR's Title, and M is the FISR's Model. The relationship may be direct, or it may be via an intermediate concept, C0, as in the example with engineering experts.

- fisr_directly_related_to(C, U, L, M) is true when there is a FISR directly related to concept C and U is the FISR's URL, L [abel] is the FISR's Title, and M is the FISR's Model.

These are given below as straightforward recursive definitions in F-logic.

```
FORALL C, U, L, M
fisr_related_to(C, U, L, M) <-
    fisr_directly_related_to(C, U, L, M).
```

```
FORALL C, U, L, M      //for Concept, Url, Label, Model
fisr_related_to(C, U, L, M) <-
    (EXISTS C0 fisr_directly_related_to(C0, U, L, M) and
      near(1,C0,C) and
      (EXISTS F  F:FISR[URL -> U; Title -> L; Model -> M].

FORALL F, C, U, L, M
fisr_directly_related_to(C, U, L, M) <-
    F:FISR[Related_Concepts ->> C] and
    F:FISR[URL -> U; Title -> L; Model -> M].
```

4.2.2.3 HD Requests (HDs)

We define a frame, HD_Request which has slots for the URL, subject, task type, and related concepts. There is a predicate called hd_related_to which is exactly analogous to fisr_related_to. This is summarized below, we include fewer details, to avoid repetition.

- hd_related_to(C, U, S, T) is true when there is a HD request related to Concept C and U is the HD's URL, S is the HD's Subject, and T is the HD's task type. The relationship may be direct, or it may be via an intermediate concept, C0, as in the above examples.

  ```
  HD_Request::Object
  [URL => String;
  ```

```
 Subject => String ;
 TaskType => String;
 Related_Concepts =>> Concept].
```

4.3. Handling Queries

An illustrated above, we have added meta-data which explicitly relates resources to concepts. The concepts that the user specifies act like search keywords except that we are employing concept-based search [Clark, 2000], which enables resources to be located which are tagged with concepts that do not exactly match any string that is input by the user. The metadata for the resources is stored as F-logic facts and rules, and acts as a database to be queried. In response to the user's request for information (enacted by pressing the *submit query* button in Figure 8.3), the system will generate an XML file containing all the relevant resources. This is then rendered via an XSL style sheet. Determining exactly which resources are related to the specified concepts requires querying the metadata. Furthermore, we require that the degree of relatedness can be variable and tailored to different users needs. The question is: how can you, from a single web page, issue many different queries, specifying different tasks and/or different concepts, and produce a different set of resources, just the ones the user needs?

To support this kind of response, tailored to the user’s specific tasks, we require a more flexible and general mechanism for generating web pages, than is provided using standard CGI scripts alone. The key idea, is to in effect, add a template feature to the XML language. Viewed another way, we have designed an XML-based template language. By convention, these have an .xmlt file extension. These files are pre-processed by a Perl script that expands the templates, and outputs an .xml file, which is rendered on the web page using an XSL style sheet. This is the response to the user’s query.

Templates contain standard XML mixed with variable references and queries written in F-logic. Note that we have *two entirely different kinds of queries*. The first is issued by the user by filling out the web form and clicking the SUBMIT QUERY button. The other kind is a query to the metadata that is encoded using the template language, and processed by the F-logic inference engine, SiLRI. The first kind is a query from the user’s perspective, so we refer to these as *user queries.* The second kind is invisible to the user—we refer to those as a *metadata queries.* Our architecture is general, so that any language and query engine could be used to express and process the metadata queries. In our prototype, all metadata queries are expressed in F-logic and handled by SiLRI.

The .xmlt file is transformed into an .xml file through a process of instantiation and expansion which occurs on the server through CGI scripts written in Perl. There are two kinds of variables in the template, which differ according to when they get instantiated. The first kind gets instantiated when the user submits their query. We refer to these as *global variables,* because wherever they occur in the templates in the .xmlt file, they refer to the same thing, and are instantiated to have the same value. They are bound in the usual fashion via CGI. The other variables are called *metadata variables*, because they refer to elements in the metadata, and are instantiated during the process of answering metadata queries. They are local to an individual template—i.e. different metadata variables can have the same name, if they are in different templates in the same .xmlt file.

The raw .xmlt template file contains a number of global variables whose values are specified by the user when they fill out the web form. In our prototype, these variables identify the category of task the user is working on, the kinds of resources they are interested in, and the concepts that the resources are related to Figure 8.4. When the user submits their query, an interim, partially instantiated template is created with all the global variables instantiated. If there are *no* metadata queries in the template, then this is the final result; no significant template language functionality would have been exercised. If there *are* metadata queries, the partially instantiated template is the starting point for the expansion process. At this point, the template will still contain unbound metadata variables, local to individual templates. In order for them to be bound, they must also be embedded in a metadata query. The expansion engine delegates processing of the queries to SiLRI. With the queries answered, some of the metadata variables will now be bound. However, nesting is possible, so the resulting template may still contain unbound metadata variables and unevaluated metadata queries. Templates are expanded recursively, until all metadata queries are executed and all free metadata variables are bound. This process is illustrated with a worked example in the next section. The final XML document is translated by the IE browser into HTML via an XSL style sheet. This could have easily been done via a server-based XSL engine such as Xalan [Apache, 2001].

We employ the F-logic language to enable tailoring of rules, which can be used to fine-tune what is meant by ‘related-to’. For example, one can easily specify degrees of relatedness based on how far away concepts are to each other in the Thesaurus. A user could submit their queries for relevant resources using different degrees of ‘related to’. This would have the effect of increasing the recall but decreasing the precision, or vica versa.

The power and flexibility of this approach derives from the fact that the Perl script queries the metadata to retrieve zero or more resources of each type. Each resource results in an instantiation of the template, in XML.

5. Worked Example

The template language extends standard XML by adding three special tags and a syntax for referring to template variables that will be bound to values as the template is expanded. The three tags are: <GLOBAL-SUBST> , <META-SUBST>, and <HTML-TEMPLATE>. Template variables are strings prefixed with the symbol: '@' (e.g. '@Cnpt') . They always occur inside an <HTML-TEMPLATE>; An '@' symbol occurring outside an <HTML-TEMPLATE> has no special significance. It is treated as raw XML. Below is an example showing how these are used. All template language features are in boldface. Everything else is standard XML.

```
<GLOBAL-SUBST>
<HTML-TEMPLATE>
<FISRS>
<META-SUBST QUERY="FORALL URL,Label,Model <-
selected("@show_fisrs") and
fisr_related_to(@Cnpt, URL, Label, Model).">
<HTML-TEMPLATE>
<FISR>
<URL><![CDATA[\/@URL]]></URL>
<Model>@Model</Model>
<Title>@Label</Title>
</FISR>
</HTML-TEMPLATE>
```

```
</META-SUBST>

</FISRS>


<HDS>

<META-SUBST QUERY="forall URL,Label  <-

selected("@show_hds") and

EXISTS X hd_related_to(@Cnpt, URL, Label, X).">

<HTML-TEMPLATE>

<HD>

<URL><![CDATA[\/@URL]]></URL>

<Title>@Label</Title>

</HD>

</HTML-TEMPLATE>

</META-SUBST>

</HDS>


<EXPERTS>

<META-SUBST QUERY="FORALL N,Id <-

selected("@show_experts") and

EXISTS P has_expertise_related_to(P,@Cnpt) and

P[Name -> N; ID->Id].">

<HTML-TEMPLATE>

<EXPERT>

<Name>@N</Name>

<ID>@Id</ID>

</EXPERT>
```

```
</HTML-TEMPLATE>
</META-SUBST>
</EXPERTS>
</HTML-TEMPLATE>
</GLOBAL_SUBST>
```

The *<GLOBAL-SUBST>* and *<META-SUBST>* tags are for specifying the scope of substitutions for global variables and metadata variables, respectively. The *<HTML-TEMPLATE>* tag is used in conjunction with a specification of variable substitutions of the following form:

{[VarA=valA1,VarB=valB1, …, VarN=valN1], [VarA=valA2,VarB=valB2, …, VarN=valN2], …}

In this case, there are two complete sets of variable substitutions. This would result in the *<HTML-TEMPLATE>* getting instantiated and included in the final .xml file twice - once for each set of substitutions. Multiple inclusions of instantiated templates, is what we mean by 'expansion'. The use of an *<HTML-TEMPLATE>* immediately following the *<GLOBAL-SUBST>* tag is a special case, in that the query string only passes a single set of variable substitutions, so that template is instantiated only once. The server generates this substitution specification when the user presses the *submit query* button. It is contained in the query string that's passed to the CGI script. The relevant part of the query string for our example is:

show_fisrs=T&show_hds=T&show_experts=T&Cnpt=flight_control

This is the substitution specification for the global variables. Performing this substitution results in the instantiation of the outermost *<HTML-TEMPLATE>;* i.e. the one that occurs in between the *<GLOBAL_SUBST>* tags. The first three are boolean variables which are 'T' if and only if the corresponding resource type box is checked (see Figure 8.3).

The substitution specifications resulting from the <*META-SUBST*> tag are more interesting, in that multiple instantiations of the <*HTML-TEMPLATE*> may result. In the example, we saw that there were four FISRs. This is because the F-logic query for related FISRs returned four sets of variable bindings as follows:

{[Url="http://…foo…", Model=797, Label="Refueling Valves"]

[Url="http://…bar…", Model=787, Label="Halon Concentration … Barrier"],

[Url="http://…baz…", Model=787, Label="Lateral Flight Control"],

[Url="http://…gaz…", Model=787, Label="Warm Flight Deck Temperatures"]}.

This is the substitution specification that is used for the instantiation and expansion of the second <*HTML-TEMPLATE*>. The <*FISRS*> and <*/FISRS*> tags are outside the <*HTML-TEMPLATE*> tags, and contain no global variables, so they pass through to the output .xml file unchanged. The resulting .xml code for the FISRS is:

```
<FISRS>
```

```
     <FISR>
     <URL><![CDATA[\/yadayada.boeing.com/cgi?model=797&foo…]]><
/URL>
     <Model>797</Model>
     <Title>Refueling Valves</Title>
     </FISR>
     <FISR>

<URL><![CDATA[\/yadayada.boeing.com/cgi?model=787&bar…]]></URL
>
     <Model>787</Model>
     <Title>Halon Concentration for Smoke Barrier</Title>
     </FISR>
     <FISR>
     <URL><![CDATA[\/yadayada.boeing.com/cgi?model=787&baz…]]><
/URL>
     <Model>787</Model>
     <Title>Lateral Flight Control</Title>
     </FISR>
     <FISR>

<URL><![CDATA[\/yadayada.boeing.com/cgi?model=787&gaz…]]]></UR
L>
     <Model>787</Model>
     <Title>Warm Flight Deck Temperatures</Title>
     </FISR>
```

```
</FISRS>
```

The query to find related HD request fails, because there were none found in the F-logic database. It returns the null set—thus the next <HTML-TEMPLATE> results in no .xml code at all. Again, the <HDS> </HDS> pass through to the output .xml file unchanged. The resulting .xml code for the HDs is just:

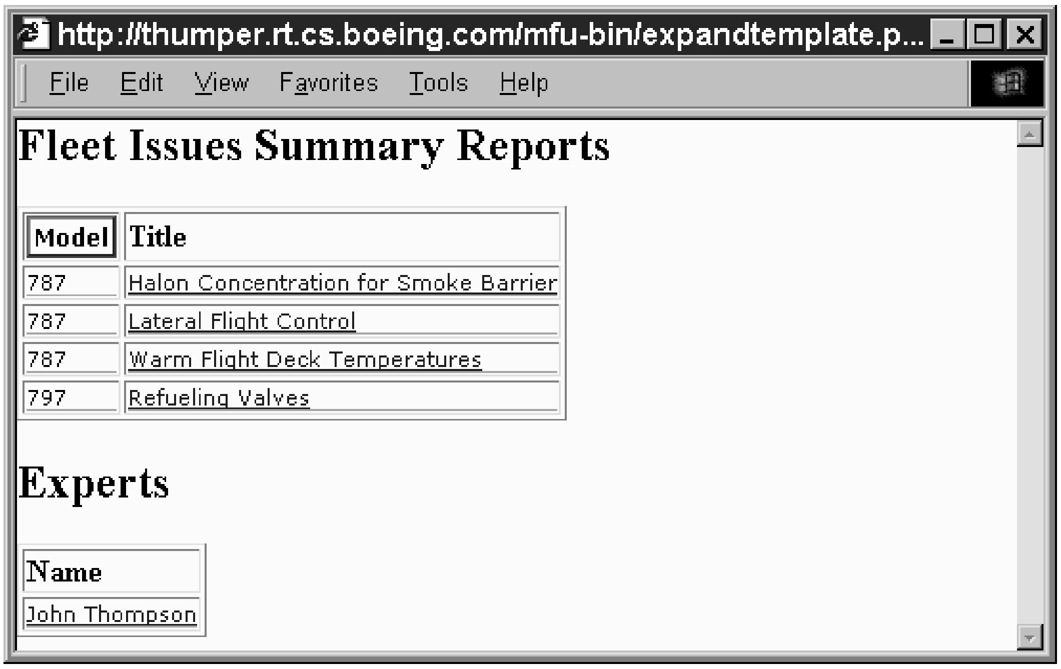
```
<HDS>
</HDS>
```

Finally, the query to find related experts, results in the single set of substitution bindings:

```
{ [Name="John Thompson", Id="42549] }.
```

Thus, the <HTML-TEMPLATE> is instantiated exactly once. The resulting xml code for the experts is:

```
<EXPERTS>
<EXPERT>
<Name>John Thompson</Name>
<ID>42549</ID>
</EXPERT>
</EXPERTS>
```


http://thumper.rt.cs.boeing.com/mfu-bin/expandtemplate.p...
File Edit View Favorites Tools Help

Fleet Issues Summary Reports

Model	Title
787	Halon Concentration for Smoke Barrier
787	Lateral Flight Control
787	Warm Flight Deck Temperatures
797	Refueling Valves

Experts

Name
John Thompson

Figure 8.5 Sample Output.

The final step is to present the information to the user in a convenient format. We used the XSL processor embedded in the IE 5.0 browser.

Conclusion

The semantic web opens opportunities for fundamentally redefining our relationship with the web by defining resources in such a way that they are not only human-sensible but also machine-sensible. Users are seldom just searching for documents on the web but are often in the process of performing some larger task. Current interfaces often don’t provide any means for the user to communicate their intentions; therefore systems can’t leverage task information in support of the user. Making web resources machine-sensible will gradually shift the primary consumers of raw web content from humans to agents who will play the role of information intermediaries. These agents will collect and integrate information based on the task users are currently performing.

Unless the semantic web community provides a means of capturing and exploiting task information, newly built semantics-based systems will likely be only moderately better at supporting users than current systems. Our modest prototype simply highlights the need and promise of taking a task-centered approach to the semantic web.

Future research should focus on better mechanisms for identifying and characterizing user tasks in a way that can be exploited by semantics based tools. We also believe there is fruitful research mixing semantics-based technologies with existing information retrieval technologies. It’s doubtful that all resources will ever be fully characterized in such a way that will eliminate the need for standard IR techniques. In addition, there is significant opportunity to exploit online machine learning techniques to improve performance of task-centered systems over time.

References

[Apache, 2001] Apache Project (2001), Xalan. See http://xml.apache.org/xalan/.

[Ashish, 1997] Ashish N and Knoblock C (1997). Wrapper generation for semi-structured internet sources. In

Proceedings of the Workshop on Management of Semi-structured data, 1997.

[Clark, 2000] Clark, P., Thompson, J., Holmback, H, Duncan, L. (2000). Exploiting a Thesaurus-Based Semantic Net for Knowledge-Based Search. In Proc 12th Conference on Innovative Applications of AI (AAAI, IAAI' 2000), pages 988-995.

[DAML, 2001] See http://www.daml.org/.

[Decker, 1998] Decker, S., Brickley, D., Saarela, J., Angele, J. (1998). A Query Service for RDF. The Query Languages Workshop, 1998. http://www.w3.org/TandS/QL/QL98/pp/queryservice.html.

[Dumais, 1999] Dumais, Susan (1999). Inductive Learning and Representation for Text Categorization. Talk at University of Washington. See http://murl.microsoft.com/LectureDetails.asp?177.

[IPL, 2001] Internet Public Library, IPL (2001). See http://www.ipl.org/ref/POTUS/rbhayes.html.

[Kifer, 1990] Kifer M., Lausen G., Wu J. (1990). Logical Foundations of Object-Oriented and Frame-Based Languages, Journal of the ACM.

[Lassila, 1999] Lassila, O., Swick, R. (1999). The Resource Description Framework (RDF) Model & Syntax. W3C Proposed Recommendation. http://www.w3.org/TR/PR-rdf-syntax/.

[Ontoprise, 2001] Ontroprise (2001). See http://www.ontoprise.de/.

[Smith, 2001] Smith, R (2001). What's Required in Knowledge Technologies - A Practical View, Proceedings of Knowledge Technologies 2001. See http://www.gca.org/attend/2001_conferences/kt_2001/default.htm

[TopicMaps, 2001] Topic Maps (2000). See http://www.topicmaps.org/.

Footnotes

[1]Rob Jasper jasperr@seattleu.edu Seattle University 900 Broadway Seattle WA, USA 98122-4340

[2]Mike Uschold michael.f.uschold@boeing.com Mathematics and Computing Technology The Boeing Company PO Box 3707, MS 7L-40 Seattle WA, USA 98124-2207 Work phone: 425-865-3605 FAX: 425-865-2965

[3]The details of this description are fictional to protect Boeing’s propriety information. It serves the purpose of illustrating our technology.

9. Knowledge Mobility: Semantics for the Web as a White Knight for Knowledge-Based Systems

Yolanda Gil[1]

"No Man is an island

Entire of itself.

Every man is a piece of the continent…."

-- John Donne (1572-1631)

1. Introduction

One of the challenges for knowledge-based systems is interoperation with other software, intelligent or not. In recent years, our research group has participated in various integration efforts, where interoperation was supported through translation techniques, mostly at the syntactic level and more recently through ontology-based approaches. In this article, I argue that the interoperation challenge will not be met with current approaches, since they entail trapping knowledge into formal representations that are seldom shareable and often hard to translate, seriously impairing its mobility. I propose an approach to develop knowledge bases that captures at different levels of formality and specificity how each piece of knowledge in the system was derived from original sources, which are often Web sources. If a knowledge base contains a trace of information about how each piece of knowledge was defined, it will be possible to develop interoperation tools that take advantage of this information. The contents of knowledge bases will be more mobile and no longer be confined within a formalism. The Semantic Web will provide an ideal framework for developing knowledge bases in this fashion. We are investigating these issues in the context of TRELLIS, an interactive tool to elicit from users the rationale for choices and decisions as they analyze information that may be complementary or contradictory. Starting from raw information sources, most of them originating on the Web, users are able to specify connections between selected portions of those sources. These connections are initially very high level and informal, and our ultimate goal is to develop a system that will help users to formalize them further.

2. The Need for Knowledge Mobility

This section argues the need for knowledge mobility from two viewpoints. First, knowledge-based systems need to be built to facilitate interoperability and thus the knowledge they contain needs to be more accessible to external systems. Second, our knowledge bases capture knowledge in such a way that it is very hard to get out or translate what they contain.

2.1. No Knowledge Base is an Island

Knowledge-based systems are no longer built to function in isolation. Every application that we have built with the EXPECT architecture [Blyhe et al., 2001; Kim and Gil 2000; Gil and Tallis, 1997; Gil and Melz, 1996] in recent years has been integrated within a larger end-to-end system in order to provide the overall functionality required. This section describes several integration efforts that illustrate important challenges in terms of knowledge mobility:

- *Significant differences in representation languages*. We have found on several occasions that different systems use representations and languages that adequately support certain functionality or problem solving capabilities. Requiring that all the systems in the integration adopt the same representation may be an option in some cases, but many times doing this may be technically challenging or simply unfeasible in practice. An alternative approach that ends up being more acceptable in practice is to allow each problem solver to use different languages and representations and then have ways to map back and forth between the two.

- *Discrepancies in modeling styles.* Even when different knowledge bases are developed in similarly expressive languages, different knowledge engineers practice different modeling styles and approaches. Translators between different languages help import knowledge bases written by different developers, but they only translate one syntactic expression into another and do not address deeper differences in representational styles. The rationale for modeling the original knowledge as a certain suite of expressions is never captured in the knowledge base.
- *Maintaining consistency among related pieces of knowledge*. A single domain description may be modeled piecemeal in separate parts of the knowledge base to reflect different views or functions of the description. The individual pieces may not be explicitly related in the knowledge base, especially when different pieces are only used by different components of the overall integration. In general, very few of the many and varied relations between individual pieces of knowledge are captured in the resulting knowledge bases.

2.1.1. Plan Generation and Evaluation

Figure 9.1 shows the architecture of a knowledge-based system for Workarounds Analysis that we developed for the DARPA High Performance Knowledge Bases Battlespace Challenge Problem [Cohen et al., 1999]. This system estimates the delay caused to enemy forces when an obstacle is targeted (e.g., a bridge is destroyed or a road is mined), by reasoning about how they could bypass, breach, or improve the obstacle in order to continue movement (e.g., by repairing a damaged bridge or installing a temporary one, fording a river, removing mines, choosing an alternative route). Several general ontologies relevant to battlespace (e.g., military units, vehicles), anchored on the HPKB upper ontology, were used and augmented with ontologies needed to reason about workarounds (e.g., engineering equipment). These ontologies were represented in LOOM, a knowledge representation and reasoning system based on description logic [MacGregor 1991]. The system also included two main problem solvers. A Course of Action Generation problem solver creates several alternative solutions to the problem. Each solution lists several engineering actions for that workaround (e.g., deslope the banks of the river, then install a temporary bridge, etc.), includes information about the resources used (e.g., what kind of earthmoving equipment or bridge is used), and states temporal constraints among the individual actions to indicate which can be executed in parallel. This problem solver was developed using a state-of-the-art planner [Veloso et al., 1995]. A Temporal Estimation/Assessment problem solver evaluates each of the alternatives and selects one as the most likely choice for an enemy workaround. This problem solver was developed in EXPECT. It used the information about engineering equipment and techniques included in the ontologies as well as several dozen problem solving methods to estimate how long it would take to carry out a given workaround plan generated by the Course of Action Generation problem solver. This system was the only one to attempt full coverage in this DARPA Challenge Problem, and demonstrated best performance at this task.

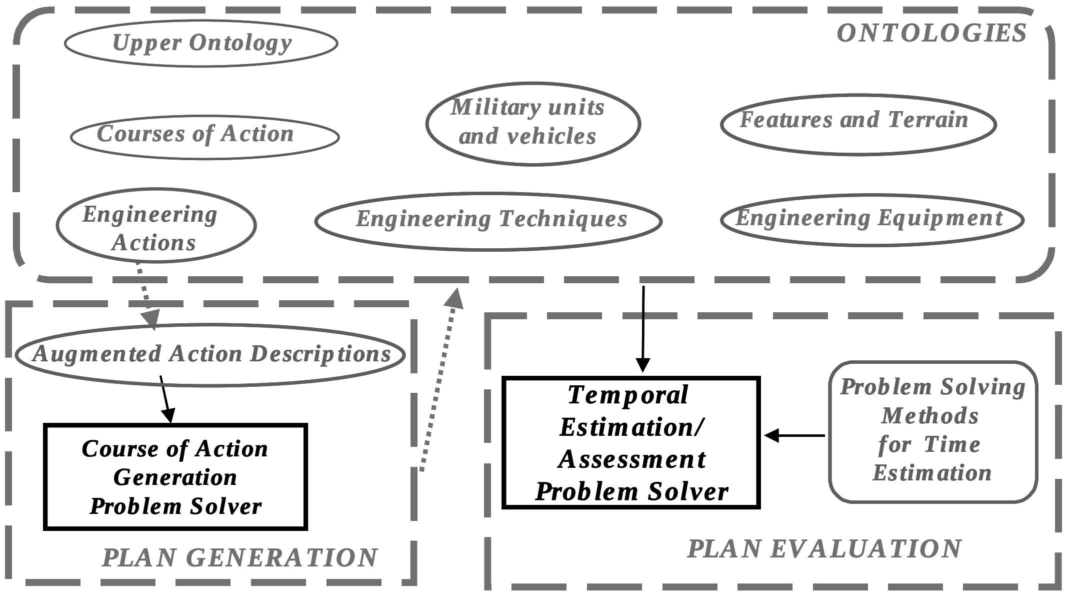

ONTOLOGIES
Upper Ontology
Courses of Action
Military units and vehicles
Features and Terrain
Engineering Actions
Engineering Techniques
Engineering Equipment
Augmented Action Descriptions
Course of Action Generation Problem Solver
PLAN GENERATION
Temporal Estimation/ Assessment Problem Solver
Problem Solving Methods for Time Estimation
PLAN EVALUATION

Figure 9.1 A Knowledge-Based System for Plan Generation and Evaluation.

One of the challenges for integration illustrated by this system is differences in representation. We found a huge gap between the representations used in the state-of-the-art knowledge representation and reasoning systems and those of a special-purpose reasoner, in our case a state-of-the-art plan generation system. The ontology of engineering actions shown in Figure 9.1 was sufficient to support the temporal estimation problem solver, but it needed to be significantly augmented by knowledge engineers to develop the action descriptions required by the planner. These descriptions could not be represented declaratively in the ontology, and thus had to be separately developed and maintained. Conversely, the planner generated very detailed plans that included many causal and temporal links among the actions, as well as steps and bindings that were helpful for plan generation but had no counterpart in an engineering plan. Thus, only portions of the final plan were incorporated back in the knowledge base. It is important to notice that the issue was not a difference in content, but in the expressivity of the language regarding planning-specific knowledge. Although it may be possible (though certainly non-trivial) to coerce the planner's representations into the ontology, it is not clear that it is a superior representation for that kind of knowledge.

Another problem was maintaining consistency in the knowledge base. We needed to keep three different parts of the knowledge base closely aligned: the ontology engineering actions, the ontology of engineering techniques, the methods for time estimation, and the planner's augmented engineering actions. Since each of these ontologies required different expertise, different knowledge engineers were resposible for each of them. The EXPECT knowledge acquisition tools [Kim and Gil, 1999] were very useful in pointing out inconsistencies between the methods and some of the ontologies, but the planner's actions were not accessible to EXPECT. Consistency across the different parts of the knowledge base had to be maintained largely by hand. Some of the performance problems during the evaluation of this system were traced to these kinds of inconsistencies.

2.1.2. Mixed-Initiative Plan Generation

Another active area of research within our project is mixed-initiative tools to help users create strategic plans, in particular in air campaign planning. In order to plan what air operations will be conducted in a campaign, military planners use a strategies-to-task methodology where low-level tasks are derived from higher level national and campaign goals. For example, a campaign objective like attaining air superiority results in an operational subobjective of suppressing enemy air sorties, which in turn creates an operational task of damaging key airbase support facilities. Ultimately, these objectives are turned into missions stating which specific aircraft, crews, and airfields will be used to accomplish the lower-level tasks. At the higher, strategic levels of planning users prefer to maintain control of the planning process and specify the objectives themselves, leaving automatic plan generation to the lower levels of the planning process. A series of plan editors were built to allow a user to define objectives and decompose them into subobjectives, possibly invoking automated plan generation tools to flush out the plan at the lower levels. Because the plan creation process is mostly manual (at least at the higher levels), it is prone to error. This is aggravated by the size of the plans (several hundreds of interdependent objectives and tasks) and by the number of different people involved in its creation. In order to help users detect potential problems as early as possible in the planning process, we developed INSPECT, a knowledge-based system built with EXPECT that analyzes a manually created air campaign plan and checks for commonly occurring plan flaws, including incompleteness, problems with plan structure, and unfeasibility due to lack of resources. For example, INSPECT would point out that if one of the objectives in the plan is to gain air superiority over a certain area then there is a requirement for special facilities for storing JPTS fuel that is currently not taken into account. Without INSPECT, this problem might only be found when a logistics expert checks the plan, a day or more after the execution of the overall plan is started. INSPECT reasons about the kinds of objectives in the plan, notices that gaining air superiority requires providing reconnaissance missions, which are typically flown in stealth aircraft that need specific kinds of fuel (e.g., JPTS) that need to be stored in special facilities.

INSPECT was very successfully received by users, since it always found unintentional errors in the plans created by INSPECT’s very own designers.

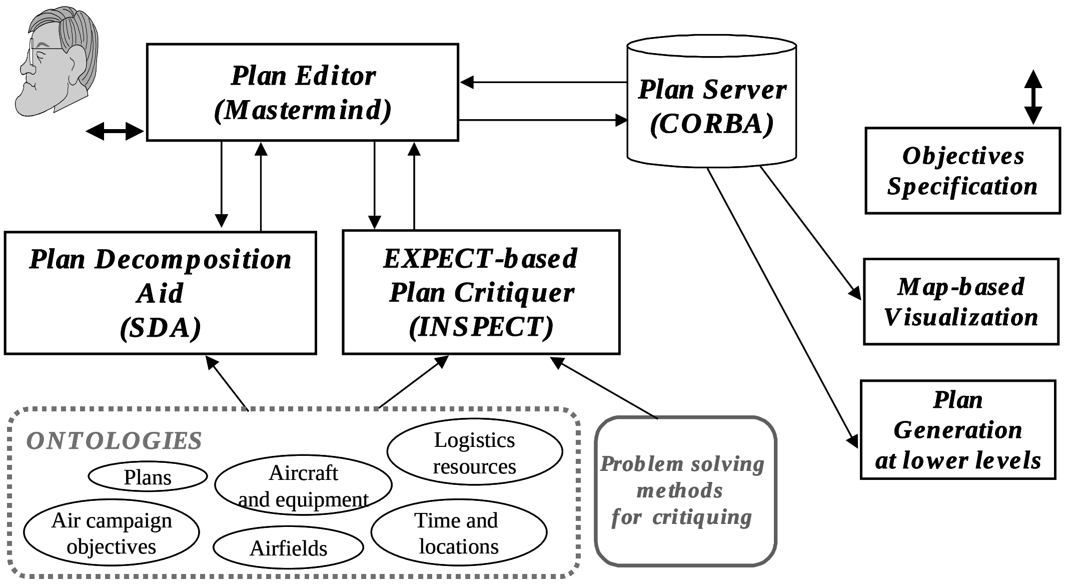


Figure 9.2 An Integration Framework for a Plan Critiquer.

INSPECT was integrated in several architectures of very different nature. Figure 9.2 shows an integration of INSPECT with a few knowledge-based tools within a pre-existing software architecture that was CORBA-based. There was a plan server that contained not only air campaign plans but also land and maritime activities in joint operations. The plan server was not amenable to any major changes, so many of the plan details and constraints that were used by the knowledge-based tools never made it there. Figure 9.2 also shows the different ontologies in INSPECT's knowledge base, whose integration and development is described elsewhere [Valente et al., 1999].

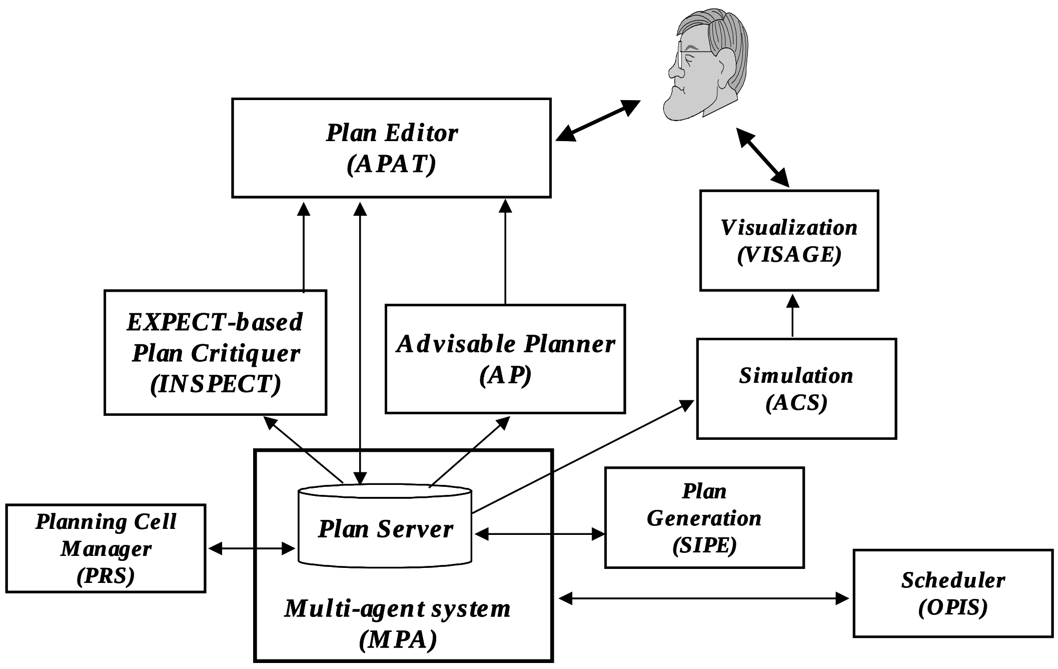


Figure 9.3 Integrating a Plan Critiquer in a Multi-Agent System.

Figure 9.3 shows another integration, this time with several other planning aids within a multi-agent planning architecture. This integration is another illustration of the representational mismatch among different systems. The plan server in this architecture contained rich representations of plans regarding their hierarchical decomposition, causal and temporal dependencies, and overall planning process management. It lacked, however, rich representations of objectives that were needed by the plan editor and by INSPECT in order to analyze the requirements of each objective, and as a result these representations were not incorporated in the plan server and were not available to any other tool. We suspect that other modules found similar discrepancies between their particular representation formalisms and the plan server.

2.1.3. Critiquing Courses of Action

Another important integration effort was to develop an end-to-end system to help users develop military courses of action, depicted in Figure 9.4. Users input a sketch on a map to specify the activities of each force and a statement of the objectives of the course of action. Several critiquers were developed by different groups to analyze different aspects of the course of action in order to give users feedback about possible problems in their design. We developed a critiquer with EXPECT to check problems related to plan structure and use of resources. The other critiquers had complementary capabilities since each of them addressed different kinds of critiques.

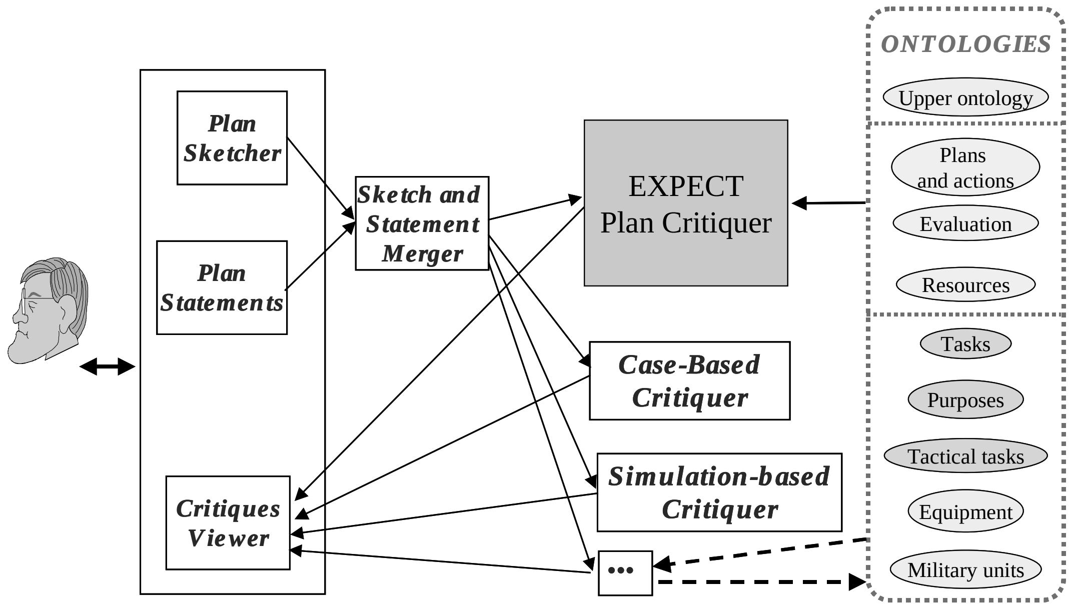


Figure 9.4 Critiquing Courses of Action.

The application-specific ontologies used by our system were developed by other groups involved in this effort. Importing these ontologies into our knowledge base was a significant effort, as it was transforming the input course of action into a format consistent with our representation.

The simpler translation issues arise in translating original ontologies in MELD or KIF into LOOM, although sometimes mapping different names was necessary (e.g., to map "TranslationLocationChange" into "Move"). More complex translations are required because of deeper representational mismatches. For example, some expressions used to describe the course of action stated that a certain task had protect as a purpose when there was something to protect, while our representation created an instance of protect as the purpose of the task. At times, some of the decomposition structure of the course of action was implicit in the order of the statements, and we had to reconstruct that structure so that it would be available in our knowledge base. We used the OntoMorph rewrite engine to support these more complex mappings, which are described in more detail in [Chalupsky, 2000]. In essence, supporting this integration was challenging even though the languages used by the different systems were comparable in terms of their expressivity. The differences in modeling approaches and representations were quite significant and required a complex translation system like OntoMorph.

2.1.4. Translation among Intelligent Agents

As part of the initial stages of a new project to develop ontology-based translation services for multi-agent systems, we conducted a study of an existing multi-agent system (not developed by us) to determine the requirements of such a translation service [Gil, 2000]. This multi-agent system was developed to support non-combatant evacuation operations, and was used to organize groups of helicopters to take people out of a dangerous area and into a safe heaven. The agents participating included a suite of interface agents that interact with users to get the initial helicopter team composition and critical locations as well as unexpected threats to the mission (e.g., explosions), a route planner, a threat (SAM site) finder, and helicopter agents that can work in teams in a simulated environment. All of these agents were initially developed in radically different contexts, and originally did not even share the application domain.

In a typical run, a few thousand messages are exchanged among the agents. We analyzed the content of messages that are representative of the communications regarding significant events and activities during the mission. Some of the findings include:

- Syntactic translators are relatively easy to build and are often readily available for many formats and languages. Yet, in the years ahead we will see large communities of heterogeneous agents where pairwise translations will become impractical. Each agent will have to advertise its capabilities and requirements in one (or just a few) formats, and rely on translators and mediators to map those requirements into those of other agents. In this particular system, syntactic translations were often required even though the developers often agreed to message formats that would minimize these needs. Even when the same kind of information was transmitted different formats were used. For example, the lat-long coordinates sometimes use the convention of an "X" and "Y" label, in other cases the coordinates were preceded by "-lat" and "-long" (this happens when the critical locations are given to helicopters by the interface agents), or just ordered within parentheses (this is the format used by the threat finder to send threat sites to the helicopter agents). A complex case of syntactic mapping was required because the threat finder takes queries in SQL format.

- Mismatches across representations were numerous and often due to modeling with different granularities and levels of abstraction. For example, the route generated by the route planner needed to be transformed to a coarser-grained route in order to be useful to the helicopter agents. The route planner sends a route composed of points that are 9 meters apart. The helicopters are tasked one route segment at a time, and when a helicopter is given a route that is of a small size (like 9 meters) it will say that the route is achieved without even taking off. Dropping details from a plan is often needed when planning agents exchange plans. This may involve dropping steps, dropping parts of steps, or may require a completely different plan altogether. Another example of the need for such transformations is that the helicopters consider routes to be composed of point-to-point segments, and the threat finder needed to be given a rectangular area to initiate its search. The route segment was made into the diagonal of the rectangular area, and as a result there were threats returned that were in the area but not very close to the route itself. Similarly, the interface agents specified helicopter landing zones as areas, while the helicopter agents take only a point as landing zone.

- Maping among different representations may be complex, and invocations of third-party agents may be necessary. The route planner and the threat finder use a lat-long coordinate system, the helicopters use an "x,y, cell" format, and the interface agents use UTM coordinates. Although it is possible to build ontology-based translators to mediate these communications, it may be more practical to use the ontologies to simply detect the mismatch and then invoke a third party to do the conversion since many such conversion systems are already available. Other cases when invocation of other agents would be useful arise when an agent can fulfill a request but needs information that the requesting agent may not have. For example, the route planner required a map URL, the interface agents require a range around a location, etc.

This particular multi-agent system illustrates the challenges in supporting interoperation among intelligent systems, even in relative small scale and numbers. It would not be simple to replace a subset of the agents involved by other agents of equivalent functionality, which could probably not be done without some additional changes and adaptations in the original remaining agents.

2.1.5. Summary

Current efforts in the knowledge-based systems community aim to support interoperability through shared ontologies and diverse translation tools. Drawing from past experiences within our own research project, this section illustrates several points:

- Shared ontologies do not address all integration issues. Content can be agreed upon, but representational needs are determined by the functionality required of the individual problem solvers and components within the system.
- Maintaining consistency in the knowledge used by several components is a great challenge. Some components may have some consistency checking facilities, but that is

often not the case with all the components in the overall system.

- Differences in modeling methodologies make translation an arbitrarily complex task. Syntactic translation and mappings are adequate only when such differences are minor.

To paraphrase John Donne, no knowledge based-system is an island. There is an increasing demand to have individually developed intelligent systems become part of larger end-to-end application, and current paths to integration are not able to support interoperability appropriately.

2.2. Educating Knowledge Bases

Large knowledge bases contain a wealth of information, and yet browsing through them often leaves an uneasy feeling that one has to take the developer's word for why certain things are represented in certain ways, why other things were not represented at all, and where might we find a piece of related information that we know is related under some context. Although the languages that we use are quite expressive, they *still force knowledge into a straightjacket:* whatever fits the language will be represented and anything else will be left out. Many other things are also left out, but for other reasons such as available time and resources or perhaps lack of detailed understanding of some aspects of the knowledge being specified.

Furthermore, *knowledge ends up represented piecemeal,* compartmentalized in whatever expressions the modeling language supports. Many of the connections between different pieces of knowledge are never stated, nor can they be derived by the system given what it knows. We see no value in representing redundant information or alternative ways to deduce the same facts: if the system can derive something in one way that may be more than sufficient.

Knowledge base developers may consult many sources presenting contradictory or complementary information, analyze the different implications of each alternative belief, and decide what and how to model the knowledge. In essence, developers often capture in the knowledge base only their final beliefs about some body of knowledge. *The rationale for modeling the knowledge the way it appears in the knowledge base is not captured declaratively*. Only consistent and complete information is captured. No indication of inconsistent but possible statements is added to the knowledge base.

As Minsky argues it [Minsky, 1970]:

There is a real conflict between the logician's goal and the educator's. The logician wants to minimize the variety of ideas, and does not mind a long thin path. The educator (rightly) wants to make the paths short and does not mind -- in fact, prefers -- connections to many other ideas.

Knowledge base developers seem to prefer the role of logicians rather than seeing themselves as educators of intelligent systems.

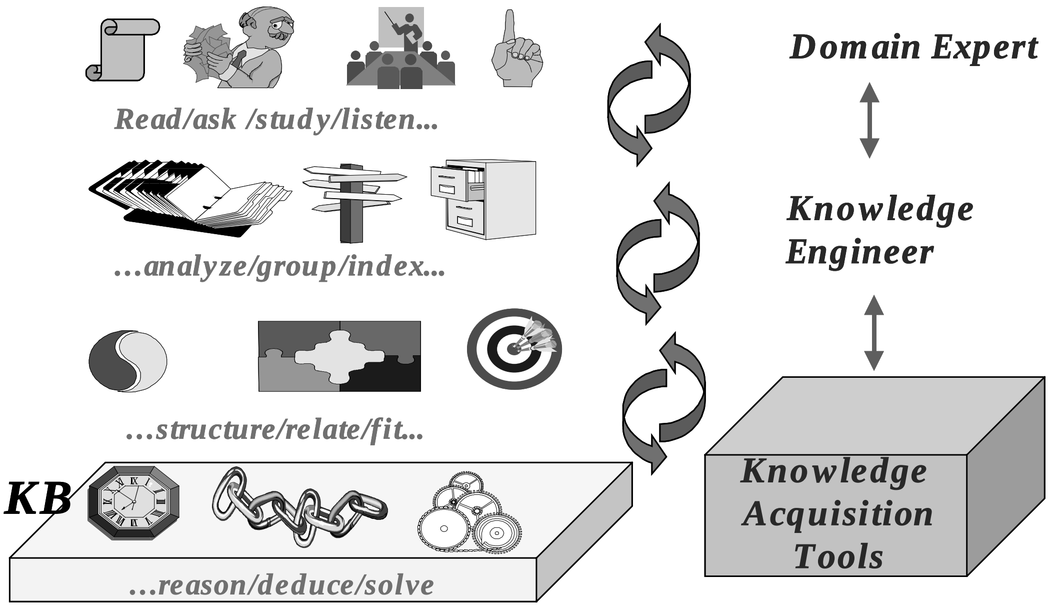

Read/ask /study/listen...
...analyze/group/index...
...structure/relate/fit...
KB
...reason/deduce/solve
Domain Expert
Knowledge Engineer
Knowledge Acquisition Tools

Figure 9.5 How Knowledge Bases are Built Today.

Figure 9.5 illustrates the limited kinds of knowledge that are captured in the final knowledge base. Developers (at least non-experts) start by consulting manuals and tutorial material, asking questions, and requesting clarifications. Their main task is to analyze and different information sources, grouping information, indexing related definitions and terms, and gathering as much raw material as possible in order to understand what needs to be represented and how. Next, they organize the information in semi-formal ways by structuring it in tables, itemized lists, and detecting opposite and complementary descriptions. Finally, they build the knowledge base itself by turning the refined results of this analysis into formal expressions that fit in the particular knowledge representation language used. Whatever documentation ends up in the knowledge base will be the only trace left of all the design and analysis process that was done to create it. None of the documentation is captured in declarative languages. The rationale of the knowledge base design is lost: the original sources, the analysis and structuring of the knowledge therein, and the tradeoffs that were considering before deciding on the final formalization. As a result:

- It is hard to extend knowledge bases. When the knowledge base needs to be extended or updated, the rationale for their design is lost and needs to be at least partially reconstructed. The knowledge sources are no longer readily available and may need to be accessed.

- It is hard to reuse knowledge contained in knowledge bases. While it is the case that entire knowledge bases can be reused and incorporated into new systems, it is harder to extract only relevant portions of them that are appropriate in the new application. Parts of the knowledge base may be too inaccurate for the new task, or may need to be modeled in a different way to take into account relevant aspects of the new application.

In summary, knowledge has a very modest degree of mobility once it is incorporated into our current systems. Some researchers are creating shared upper ontologies that can serve as a common substrate for the more detailed knowledge in specific knowledge bases, thus facilitating interoperation and reuse. Some have argued that the brittleness in our knowledge bases can be addressed by providing common-sense and other background knowledge to these systems. These approaches may be part of the solution, but it will not address some of the issues brought up here. Current intelligent systems are hard to integrate, maintain, and understand because their *knowledge bases have not been truly educated on the topics they are supposed to know about.*

3. A New Generation of Knowledge Bases: Resilient Hyper Knowledge Bases

In order to empower intelligent systems, we believe we need to allow them access to the roots and rationale of the knowledge they contain. Furthermore, the knowledge base should not just contain a body of formalized expressions; rather, we should extend our view of a knowledge base to include a variety of formats and representations as well as alternative renderings of the same (or related) knowledge. They should include as many connections as possible, as stated in the original sources and as they result from the analysis of the knowledge base developer. This approach would create a new generation of knowledge bases: Resilient Hyper Knowledge Bases (RHKBs), that will be more resilient to change and reuse and will be heavier in connections and hyperlinks.

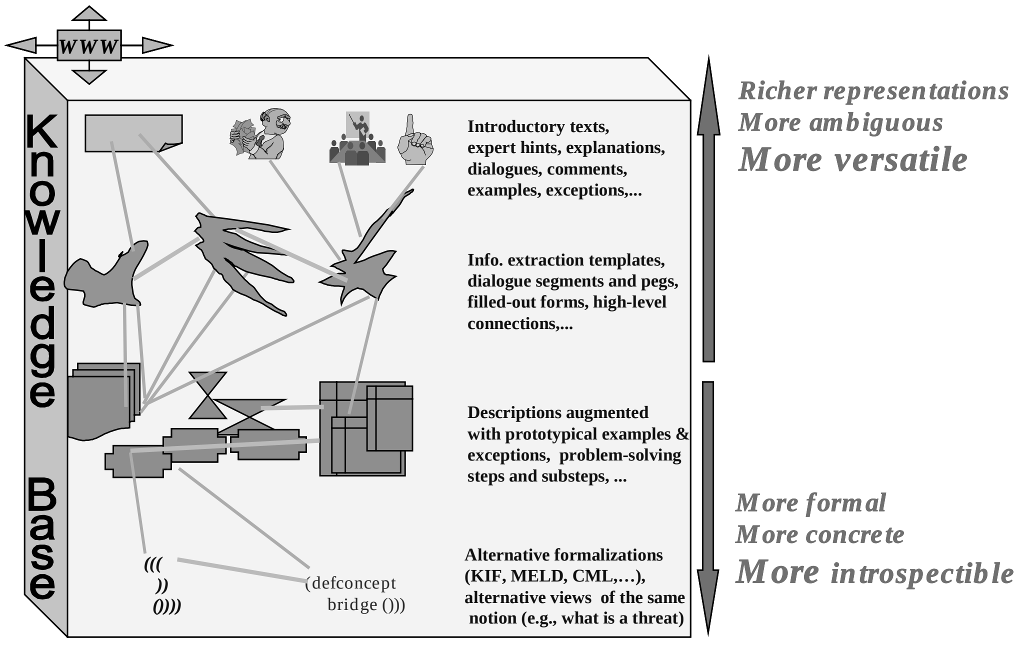

WWW
Knowledge Base
Introductory texts, expert hints, explanations, dialogues, comments, examples, exceptions,...
Info. extraction templates, dialogue segments and pegs, filled-out forms, high-level connections,...
Descriptions augmented with prototypical examples & exceptions, problem-solving steps and substeps, ...
Alternative formalizations (KIF, MELD, CML,...), alternative views of the same notion (e.g., what is a threat)
((( )) ()))
(defconcept bridge ()))
Richer representations
More ambiguous
More versatile
More formal
More concrete
More introspectible

Figure 9.6 A Resilient Hyper Knowledge Base (RHKB).

Figure 9.6 depicts a Resilient Hyper Knowledge Base in contrast with the current approaches illustrated in Figure 9.5. Originally, the development of the knowledge base starts with documentation, examples, dialogues (perhaps with experts), detailed explanations of special cases, notes on exceptions, hints and comments on specific topics, etc. From these, the developer will extract templates, relevant dialogue segments, itemized lists and tables to organize information, etc. This should be done while always keeping a trail of connections to the original sources. The developer will also indicate some connections between different portions of the original sources. It is our experience that many of the original sources either exist or are converted into resources on the Web, and as a result the developer can exploit the hyperlinks and connections that already exist in these original sources. As the developer continues this analysis, additional sources may be sought and incorporated at the higher levels, further enriching the grounding of the final knowledge base that is being developed.

Next, the developer can identify descriptions and associate with them prototypical examples as well as exceptions, pieces of problem solving knowledge in terms of steps and substeps, tables of parameters and values, etc. Any of these new distillations will continue to be connected to any other pieces in the knowledge base that they were derived from. The developer can also mark alternative views on the same knowledge, indicate contradictory statements by different sources, and dismiss some pieces of knowledge that may not seem useful for current purposes.

Finally, a developer can turn the more structured components into formalized expressions, in one or more languages and formalisms. Contradictory statements can be formalized and connected and marked as contradictory, for someone to pick and choose as they incorporate knowledge into a reasoning engine.

During this process, the developer can annotate the reasons for making certain decisions regarding which knowledge to model and how to model it. These annotations will help further in understanding the rationale for the development of the knowledge base.

We have described here the process with four stages to show the incremental nature of this analysis, but there may be as many levels of refinement as the nature of the knowledge and the final system may require.

Notice that in the higher levels of refinement, the representations may be richer, more versatile, but at the same time more ambiguous. In some sense, plain human language (i.e., text) may be the most mobile vehicle to state knowledge. The many users of the World Wide Web use the same pages for a variety of purposes and tasks, the ultimate signature of true knowledge reuse. At the lowest levels of refinement, the representations are more formal, more concrete, and also more introspectible, lending themselves more to automated analysis and reasoning.

There are many benefits to this approach:

- *Knowledge can be extended more easily.* The formalized, final expressions may not necessarily contain every detail in every knowledge source, but if the need arises the system is better positioned to digest additional knowledge. This could be done in two ways: the developer could incorporate the additional knowledge or perhaps the system could use some automated tools to extract that knowledge itself (since it has access to the sources and how they were originally processed).

- *Knowledge can be reused and translated at any level.* Another system can be built by reusing only the higher levels of the design process and incorporating other sources to create different final formalized expressions. Other developers can tap into any intermediate results of the analysis and do not have to start from scratch. Knowledge does not have to be reused only at the lowest level as it is today.
- *Knowledge can be integrated and translated at any level to facilitate interoperability.* Translators can be built to transform and map knowledge at higher levels. The rationale and meaning of different pieces of knowledge can be available to support translation and interoperation.
- *Intelligent systems will be able to provide better explanations.* We find that many users are reluctant to accept the solutions presented by the systems and ask for

explanations not of how the system derived an answer automatically but instead ask for explanations of why the system starts out with a certain fact or belief. When users are shown the reasons for certain assumptions and the fact that certain sources were consulted to make that assumption they are reassured in the competence of the system to provide those answers. Capturing this trail within the knowledge base will enable the system to generate these kinds of justifications and explanations.

- *Content providers will not need to be knowledge engineers.* Although only those trained in the art of designing, modeling, and writing formal expressions can build the more refined portions of RHKBs, anyone can contribute to the higher levels. Many people in diverse disciplines acquire the analytical skills that suffice to organize and digest knowledge sources.

Many existing tools for text extraction (e.g, to extract significant event descriptions from news articles) and discourse analysis (e.g., to segment text into meaningful portions) could be used to support these earlier steps of the analysis. Existing approaches to derive interdependencies among pieces of knowledge may be used to help users create connections among diverse pieces of knowledge. Other tools can be developed to support transformations at the lower levels (e.g., to turn tables into instances and role values). The overhead that may be incurred in creating knowledge bases using this approach is, in our view, not as significant compared to the analysis efforts that developers undergo. It may even save developers time if others can look up the rationale trail instead of asking them directly detailed questions about the portion of the knowledge base they are developing.

The approach presented here has many relations to software engineering methodologies to capture the rationale for software development, and to higher-level languages and frameworks to develop knowledge-based systems. Unfortunately, these methodologies are not common practice among developers of knowledge bases for lack of adequate tools to support developers in this process. Moreover, these methodologies are aimed at software and knowledge engineers and are not very accessible to other potential knowledge base developers, such as end users and/or domain experts.

The Semantic Web will provide an ideal substrate to ground knowledge bases into their original knowledge sources, and to contain the progressively defined pieces of knowledge and the connections among them. More and more every day, knowledge originates and ends in the Web, and we find ourselves extracting knowledge from the Web, processing it inside of a knowledge base, then putting the results back on the Web. It only makes sense to integrate knowledge bases (their content and their reasoning) more closely with the Web.

4. TRELLIS: Building Resilient Hyper Knowledge Bases

The TRELLIS project is our first step towards enabling the creation of RHKBs. In previous work within the EXPECT project, we have investigated several approaches to developing knowledge acquisition tools to enable end users to extend a knowledge base, including analysis of Interdependency Models, scripts to plan acquisition dialogue, exploiting problem solving methods and other background knowledge, and creating English-based structured editors [Blythe et al., 2001; Kim and Gil, 2000; Gil and Tallis, 1998; Swartout and Gil 1995]. EXPECT helps users enter knowledge at the lower levels of an RHKB, and has been shown to be quite effective in several user evaluations with subjects not familiar with programming and formal languages. TRELLIS acquires more informal knowledge and is aimed to support the higher levels of development of RHKBs.

The key innovative ideas behind our approach are:

- *Supporting users to create knowledge fragments from the original sources as well as from other fragments.* The key is to capture how a developer progressively generates new knowledge that results in added value to the original raw information sources. Our goal is to support users to highlight key salient information from large reports and documents, to add new knowledge fragments based on their analysis and integration of existing information, and to finally create semi-formal fragments.
- *Capturing and exploiting semantic interrelationships among information items.* TRELLIS will 1) facilitate semantic markup of relationships between different pieces of knowledge, 2) exploit semantic markups in given problem solving contexts, and 3) suggest additional relationships based on those already specified by the user. Users will be encouraged and rewarded to add valuable annotations over raw information sources, since the more annotations they add the more help the system can provide in their work. When the user chooses to do little or no annotation, the system will provide weaker support (based on default heuristics and strategies) and will still help the user as much as possible.

- *Extensible semantic markup of information items and their relationships*. Users will be able to draw from a core semantic markup language that will contain a basic domain-independent vocabulary to formulate annotations. They will also be able to extend this core language with additional terminology useful in their particular domain. Using this language, users will be able to annotate not only the information items themselves, but they will also be able to annotate the relationships among them, which will enable them to qualify and describe interdependencies between different information sources and how they relate to a new conclusion or assessment added by the developer. In essence, links between the information items will be first class citizens in the knowledge base.

Figure 9.7 shows an overview of the architecture of TRELLIS. A User typically starts searching the Web for a certain document, or indicating a pointer to a specific Web resource that contains useful information. Each is considered an information item. Information items may include raw information sources (an image, a text document, a video, etc.) as well as products of previous analysis (by the user or by other users.) All the information items are in some sense the knowledge base that TRELLIS operates on, and we refer to it as the Semantically Marked Information Base, or SMIB. We refer to an information item as an EI2 (Extended Information Items).

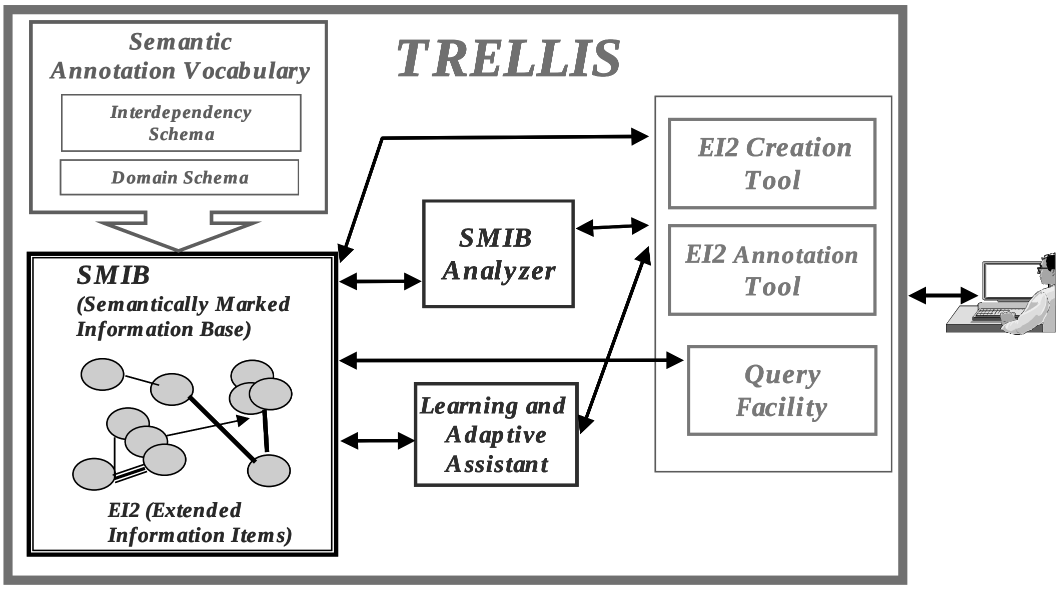
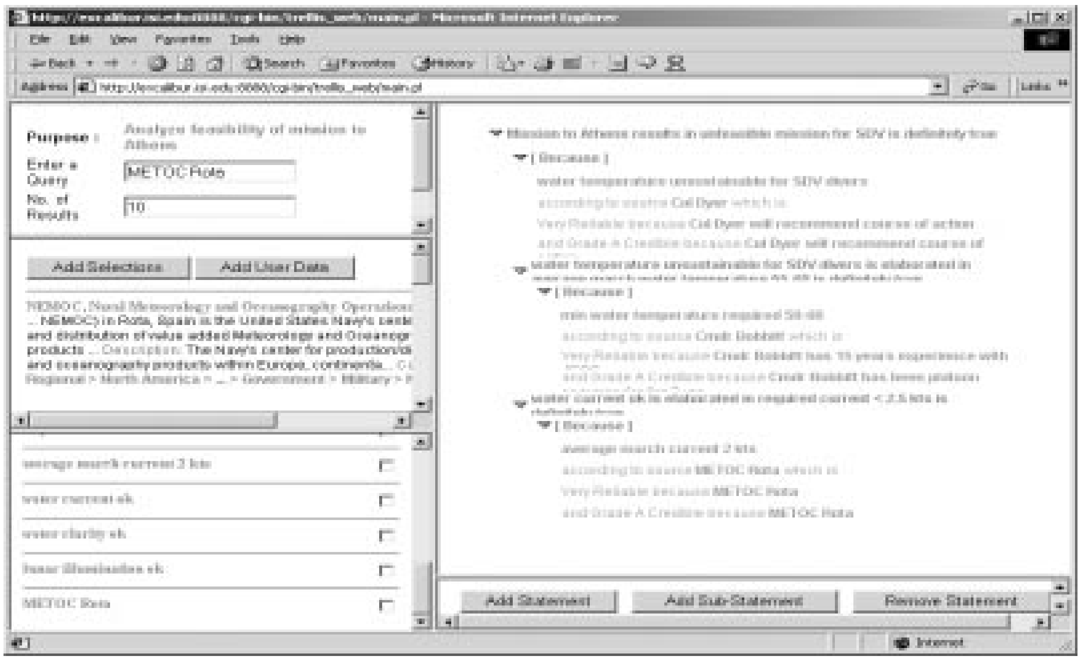

TRELLIS
Semantic Annotation Vocabulary
Interdependency Schema
Domain Schema
SMIB (Semantically Marked Information Base)
EI2 (Extended Information Items)
SMIB Analyzer
Learning and Adaptive Assistant
EI2 Creation Tool
EI2 Annotation Tool
Query Facility

Figure 9.7 Overview of the TRELLIS Architecture.

Users extend the SMIB using two tools: the Annotation Tool and the Creation Tool. They can use the EI2 Annotation Tool to add semantic annotations to an EI2 to describe its contents and to relate them to other EI2. For example, an EI2 may be annotated as containing a map, or an interesting event. The Annotation tool can also be used to relate EI2. The tool will provide an editor with a set of connectors. An example is a connector to denote that two EI2s are contradictory. This way, the user may link an EI2 that contains a description of a product as having a tag price of $20 to another EI2 that has the same product with a price of $25.

The Annotation tool draws on a library of semantic annotations and connectors that will be based on a core domain-independent language defined by the Semantic Annotation Vocabulary. An Interdependency Schema defines a vocabulary for connectors based on a variety of dimensions: pertinence, reliability, credibility, structural (x is example of y, x is part of y, x describes y, etc.) causality (x1 x2...xn contribute to y, x1 x2...xn indicate y, etc.) temporal ordering (x before y, x after y, x simultaneous with y, etc.), argumentation (x may be reason for y, x supports y, etc.). The Domain Schema contains a core vocabulary to annotate the content of documents that extends the Interdependency Schema with domain terms. Our plan is that TRELLIS will provide a core vocabulary, and users will be able to extend it with additional terms.

The Creation Tool enables users to create new EI2. For example, a user may create an EI2 as an assessment that he or she formulates based on existing EI2. If a combination of some subparts of EI2 lets a user conclude or decide on some definition, then the subparts can be captured into a new Information Item, that drops all other irrelevant parts of the original EI2. A new EI2 can be added by extracting or summarizing some of the previous results.


File Edit View Favorites Tools Help
Back Search Favorites History
Address http://excalibur.isi.edu:8080/cgi-bin/trellis_web/main.pl
Go Links
Add Selections
Add User Data
Add Statement
Add Sub-Statement
Remove Statement
Internet

Purpose : Analyze feasibility of mission to Athens

Enter a Query: METOC Rota

No. of Results: 10

NEMOC, Naval Meteorology and Oceanography Operations ... NEMOC) in Rota, Spain is the United States Navy's center ... and distribution of value added Meteorology and Oceanography products ... Description: The Navy's center for production/distribution of meteorology and oceanography products within Europe, continental ... Regional > North America > ... > Government > Military > [illegible]

average march current 2 kts

water current ok

water clarity ok

lunar illumination ok

METOC Rota

Mission to Athens results in unfeasible mission for SDV is definitely true

[Because]

water temperature unsustainable for SDV divers

according to source Col Dyer which is

Very Reliable because Col Dyer will recommend course of action

and Grade A Credible because Col Dyer will recommend course of [illegible]

water temperature unsustainable for SDV divers is elaborated in [illegible] is definitely true

[Because]

min water temperature required 58-68

according to source Cmdr Bobbitt which is

Very Reliable because Cmdr Bobbitt has 15 years experience with [illegible]

and Grade A Credible because Cmdr Bobbitt has been [illegible]

water current ok is elaborated in required current < 2.5 kts is definitely true

[Because]

average march current 2 kts

according to source METOC Rota which is

Very Reliable because METOC Rota

and Grade A Credible because METOC Rota

Figure 9.8 A Snapshot of the Current TRELLIS Interface.

Figure 9.8 shows a snapshot of the current user interface of TRELLIS. In this case, a user is using TRELLIS to decide whether a mission to take Navy SEALs to Athens is feasible. Given the Web sources consulted and the indicated capabilities of the SEAL team (shown on the left), the user has entered the rationale for deciding that the operation is not feasible.

We plan to extend TRELLIS with learning and adaptive techniques in order to offer to the user improved and extended capabilities over time. As users annotate more EI2 and create new EI2 that capture their analysis, TRELLIS will be able to exploit this information and become increasingly more proactive. We also plan to add a Query Facility that will allow users to search the SMIB based on the semantic annotations of the EI2. It will include a structured editor to guide users to formulate queries using the semantic annotation vocabulary defined in the schemas.

In summary, TRELLIS provides users with tools that enable them to specify information in increasingly more structured form, and to specify semantic annotations that can be exploited for processing and integration of separate information items.

Conclusions

Integrating knowledge-based systems within end-to-end systems remains a challenge. In this article, we have shown some practical examples of the difficulties of enabling interoperability among knowledge bases. We have argued that current approaches will only partially solve the interoperability problems, since the knowledge will continue to be trapped in low-level representations with no roots or connections to the rationale that was used in creating them. We have presented a different approach that would create Resilient Hyper Knowledge Bases, which contain knowledge from its initial rendering in unstructured raw information sources and captures how it was refined and interrelated until the developer created the final formal representations. This new generation of knowledge bases will be more resilient to changes and integrations, and will contain highly interconnected knowledge fragments at all levels of abstraction. It will also enable people who have no training in knowledge engineering to contribute content, at least in the initial stages of development. The Semantic Web is an ideal substrate for these knowledge bases, and will be a great contributor to solving interoperability issues.

The spirit of the Web was clearly to empower all people to access and publish information. The Semantic Web will enable them to turn this information into knowledge that can be understood by machines. We must embrace this spirit, and not continue to keep knowledge bases out of the reach of the public at large. Or each other.

Acknowledgements

I would like to thank Jim Blythe, Jihie Kim, Larry Kite, Surya Ramachandran, and Pablo Yanez for their helpful discussions on TRELLIS, and Varun Ratnakar for implementing its interface. I would also like to thank Jim Blythe, Jihie Kim, Surya Ramachandran, Bill Swartout , and Andre Valente, for their participation in our group's integration efforts described in this paper, which motivated much of this discussion, as well as other researchers that contributed to the integration that are too many to list here. The TRELLIS project is supported by a grant from the Air Force Office of Scientific Research (AFOSR) with award number F49690-00-1-0337. The integrations of EXPECT’s knowledge-based systems described here were funded under the DARPA High Performance Knowledge Bases (HPKB) program with award number F30602-97-1-0195, the DARPA/Rome Planning Initiative (ARPI) with award number DABT 63-95-C-0059, the DARPA Joint Forces Air Component Commander program (JFACC) with award number F30602-97-C-0118, and the DARPA Control of Agent-Based Systems (CoABS) with award number F30602-97-1-0195.

References

[Blythe et al., 2001] Jim Blythe, Jihie Kim, Surya Ramachandran, and Yolanda Gil. "An Integrated Environment for Knowledge Acquisition." Proceedings of the 2001 International Conference on Intelligent User Interfaces (IUI-2001), Santa Fe, New Mexico, January 2001.

[Chalupsky, 2000] Hans Chalupsky. "OntoMorph: A Translation System for Symbolic Knowledge". In Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR-2000), Breckenridge, CO, April 2000.

[Cohen et al., 1999]. Paul R. Cohen, Robert Schrag, Eric Jones, Adam Pease, Albert Lin, Barbara Starr, David Easter, David Gunning and Murray Burke. "The DARPA High Performance Knowledge Bases Project". In Artificial Intelligence Magazine. Vol. 19, No. 4, pp.25-49, 1998.

[Gil, 2000] Yolanda Gil. "An Analysis of the CoABS TIE-1 Messages: Opportunities and Challenges for Agent Translation Services". USC/ISI Internal Project Report. Marina del Rey, CA, 2000.

[Gil and Melz, 1996] Yolanda Gil and Eric Melz. "Explicit Representations of Problem-Solving Strategies to Support Knowledge Acquisition." Proceedings of the Thirteen National Conference on Artificial Intelligence (AAAI-96), Portland, OR, August 4-8, 1996.

[Gil and Tallis, 1997] Yolanda Gil and Marcelo Tallis. "A Script-Based Approach to Modifying Knowledge Bases". Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97), pp. 377-383, AAAI Press, July 27-31 1997.

[Kim and Gil, 1999] Jihie Kim and Yolanda Gil. "Deriving Expectations to Guide Knowledge Base Creation." Proceedings of the 6th National Conference on Artificial Intelligence (AAAI-99), pp. 235-241, AAAI/MIT Press, July 18-22 1999.

[Kim and Gil, 2000] Jihie Kim and Yolanda Gil. "Acquiring Problem-Solving Knowledge from End Users: Putting Interdependency Models to the Test." Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-2000), Austin, TX, July 30-August 3, 2000.

[MacGregor, 1991] Robert MacGregor. "The Evolving Technology of Classification-Based Knowledge Representation Systems". In J. Sowa (Ed.), Principles of Semantic Networks. San Mateo, CA: Morgan Kaufmann. 1991.

[Minsky, 1970] Marvin Minsky. "Form and Content in Computer Science". Journal of the ACM, 17(2), pp. 197-215, April 1970.

[Swartout and Gil, 1995] Bill Swartout and Yolanda Gil. "EXPECT: Explicit Representations for Flexible Acquisition". In Proceedings of the Ninth Knowledge Acquisition for Knowledge-Based Systems Workshop, February 26-March 3, 1995. Banff, Alberta, Canada.

[Valente et al., 1999] Andre Valente, Thomas Russ, Robert MacGregor and William Swartout. "Building and (Re)Using an Ontology of Air Campaign Planning". In IEEE Intelligent Systems 14:1, pp. 27-36, Jan-Feb, 1999.

[Veloso et al., 1995] Manuela Veloso, Jaime Carbonell, Alicia Perez, Daniel Borrajo, Eugene Fink and Jim Blythe. "Integrating Planning and Learning: The PRODIGY Architecture." Journal of Theoretical and Experimental AI, 7(1), 1995.

Footnotes

[1]USC/Information Sciences Institute

10. Knowledge Modelling for study of domains and inter-domain relationship – Knowledge Discovery

Sanjeev Thacker[1], Amit Sheth[1], and Shuchi Patel[1]

1. Introduction

Relationships are fundamental to supporting semantics [Wiederhold, 1997], [Sheth, 1996], and hence to the Semantic Web [Berners-Lee, Hendler & Lassila, 2001], [Fensel & Musen, 2001]. Till date, focus has been on simple relationships such as is-a and is-part-of, as in DAML/OIL [Ont]. In this work, we adapt our earlier work on MREF [Shah & Sheth, 1998] to develop a framework for supporting complex relationships. A framework to manage complex relationships as discussed here becomes the basis for knowledge discovery from the information interlinked by the Semantic Web.

Our work primarily builds upon earlier research in integrating information systems that has also been applied to exploiting web-accessible distributed across heterogeneous information sources. Primary focus of information integration systems has been to model these diverse data sources and integrate the data by resolving the heterogeneity involved to provide global views of domains (one point access) for querying. We shift the focus from modeling of the information sources for purpose of querying to extracting useful knowledge from these information sources. This, we believe, can be achieved by modeling the complex relationships among the domains to study and explore the interaction that exists between them. In addition to information source and relationship modeling, operations are also modeled as part of the knowledge to exploit the semantics involved in performing complex information requests across multiple domains. The system's framework provides a support for knowledge discovery. Knowledge representation and support for relationships, which are fundamental to the concept of Semantic Web are described in this chapter.

Consider the capability provided by current research prototypes to support integration of information from diverse sources of data over a domain to provide the user with a unified structured (homogeneous) view of that domain for querying. On an integrated view of "earthquakes" one can ask queries of the nature "find information of all the quakes that occurred in California since 1990". However, there is still a limitation on the type of queries that can be answered using such integrated domain views. Assuming views on earthquakes and nuclear tests did exist, can one answer the question "Do nuclear tests cause earthquakes?" How can one study such relationships between the two domains based on the data available on diverse web accessible sources? Let us consider a known relationship between air pollution and vegetation. Assuming the necessary views did exist can a question like "How does air-pollution affect vegetation" be answered using only the integrated views? These queries are beyond the realm of the existing systems. There is a compelling need to be able to model the semantic correlation of data across the domains and then be able to pose complex information requests.

Support for semantic correlation involving complex relationships has been is demonstrated in the InfoQuilt[2], which provides a framework and a platform to answer complex information requests of the above nature. The novel features of InfoQuilt are:

- A mechanism to express and understand the complex semantic inter-domain relationships
- A powerful interface which can use the semantic knowledge about domains and their relationships to construct complex information requests known as Information Scapes (IScapes). Most information integration systems (e.g., SIMS [AKS96]) focus on efficiently retrieving data on a single domain only. IScapes, on the other hand, are requests that could span across one or more domains and involve inter-domain relationships too. These information requests have a much higher degree of expressiveness compared to keyword based search techniques provided by web-based search tools and database queries that focus only on the syntax and structure of the data.
- Ability to analyze the data available from multitude of heterogeneous autonomous sources to explore and study potential relationships using IScapes. Although InfoQuilt

provides the tools necessary for data analysis, human involvement is a part of the process. This forms the basis of the knowledge discovery supported by InfoQuilt.

- A framework to support complex operations that can be used for post-processing of data retrieved from the sources (*e.g.* statistical analysis) and as complex operators needed to define inter-domain relationships.

Major issues addressed in building the InfoQuilt system are as follows:

- A large portion of the data on the web is either unstructured or semi-structured. There are syntactic, structural and semantic heterogeneities across different sources providing information about the same domain [Kashyap & Sheth, 1997], [Sheth, 1999]. In addition to these inconsistencies there is overlapping of information
- The knowledge of domains and their sources need to be modeled so that the system is able to automatically identify the sources of data that could possibly provide useful information with respect to the information request while carefully pruning the others, maximizing the ability to obtain faster and more comprehensive results from the same available sources
- Representation of complex relationships spanning across multiple domains involves modeling the semantics involved in their interaction

- Posing complex information requests containing embedded semantic information requires the system to able to understand the semantics of the request to retrieve relevant results corresponding to a request
- Exploring new relationships is of special interest to us which might require an ability to post-process the data obtained from the sources in several ways

This chapter focuses on our work on semantic correlation and complex relationships, in the hope of supporting "deeper semantics" in the Semantic web. The chapter is organized as follows. Section 2 focuses on our approach to modeling the domains, inter-domain relationships, information sources available to the system, and functions that form the system’s knowledge base. Section 3 addresses the use of IScapes in our system to develop and deploy complex information requests that are answered using the system knowledge and set of resources available to the system. The knowledge discovery, which is the ability of InfoQuilt to study domains, mine and analyze the data available from the sources and explore relationships, is described in Section 4. Section 5 discusses the use of visual toolkits that are provided to construct knowledge base and construct and execute IScapes. We compare our approach to other state of the art information integration systems in Section 6 and present our conclusions and planned future improvements in Section Conclusions.

2. Knowledge Modeling

The representation of information sources in the system and creation of an "integrated structured view" of the domain that the users can use are two of the critical issues that need to be addressed in any information integration system. The two main approaches used for information integration are [Dus97]:

- *Source-Centric:* The integrated view of the domain is modeled first. Both user queries and source views are specified in terms of that view. For each information request, the integration system needs to plan how to answer it using the source views.
- *Query-Centric:* User queries are in terms of views synthesized at a mediator that are defined on source views. After a view expansion at the mediator, the query is translated to a logical plan that is composed of source views.

The term ‘view’ here implies some form of representation of the domain and not a view in a database. InfoQuilt adopts the source-centric approach. The source-centric approach has the following advantages over the query-centric approach. First, the integrated views of domains are created independent of their source views. Second, the source views are independent of each other and are described in terms of their domain model. Hence, adding, removing or modifying a source can be done dynamically without affecting any other source views. In the query-centric approach, adding and removing sources additionally requires redefining the domain view and the sources need to be mapped with each other making schema integration difficult. Lastly the source-centric approach is more suitable for sources other than database sources.

This work has been built upon work done in [Ber98, Laksminarayanan, 2000]. InfoQuilt system knowledge comprises of knowledge about the domains, resources accessible to it, inter-domain relationships and complex operations like functions and simulations. We use ontologies to describe the domains to the system. An ontology is meant to capture useful semantics of the domain such as the domain characteristics, the set of terms and concepts of interest involved and their meanings. Section 2.1 describes how a domain is represented using an ontology. We will use the terms ontology and domain interchangeably in the rest of the chapter unless explicitly specified. There can be multiple resources accessible to the system and an ontology can have several resources that provide relevant data on it. Section 2.2 describes how we define resources in terms of the ontologies. Section 2.3 describes how complex relationships between two or more domains are modeled. Finally, section 2.4 describes how we support operations such as simulations and functions in the system.

InfoQuilt uses XML as the knowledge representation language. We provide a visual tool called *Knowledge Builder*, that lets a user easily create and update the knowledge base of the system. Section 2.5 describes the Knowledge Builder and its functionalities in detail.

2.1. Ontology (Domain Modeling)

An ontology should be able to provide a good understanding of the domain it represents. This includes related terms/concepts, their definitions and/or meanings, their relationships with each other, and characteristics of the domain. It also helps in resolving differences between the models of the domain used by the available sources. This is done by mapping the data available from all resources for that domain from the local model used by the resource to the model specified by the ontology, which the user can then use. The user can create a classification of real-world entities and represent them as ontologies in the system. This classification is based on the human perception of the world broken down into several domains and their sub-domains and so on (real world objects as perceived from a modeling standpoint). The representation of an ontology comprises of related terms (attributes), its place in the domain hierarchy, domain rules, and functional dependencies. The ontology of a sub-domain is said to inherit from the ontology of its parent domain. A child ontology inherits all attributes, domain rules and functional dependencies of its parent ontology.

Example 1: Consider the domain of disasters. They could be sub-categorized as natural disasters and man-made disasters. Natural disasters could be further sub-categorized into earthquakes, volcanoes, tsunamis, etc. Similarly, man-made disasters can have sub-categories like nuclear tests. An example hierarchy is shown in Figure 10.1. An ontology's place in the domain hierarchy is represented by specifying its parent ontology, if any. However, it is not necessary for an ontology to be a part of some hierarchy.

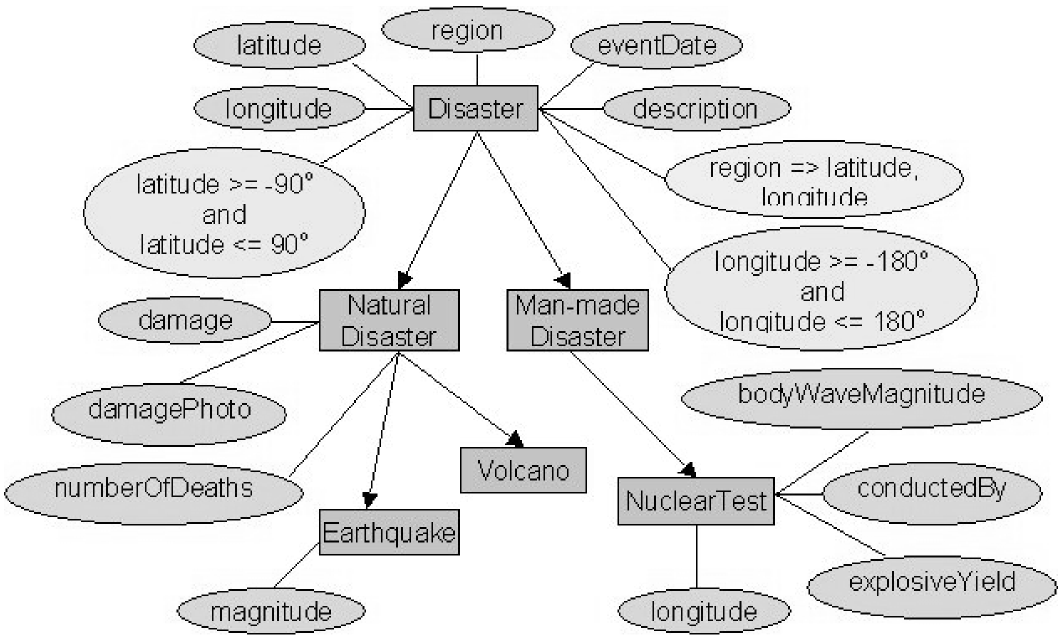


Figure 10.1 Example domain hierarchy

2.1.1. Attributes

The terms and concepts of the domain are represented as attributes of the ontology. As mentioned earlier, ontology inherits all attributes of its parent ontology. The meaning of each attribute and its syntax (type and format) is standardized so that it has a precise interpretation and use.

Example 2: An earthquake can have attributes like the date it occurred, region where it occurred, latitude and longitude for its epicenter, a description, number of people that died, magnitude and an image showing some damage or fault line as seen in Figure 10.1. We will also use the following notation to represent ontology.

```
Earthquake ( eventDate, description, region, magnitude,
             latitude, longitude, numberOfDeaths,
             damagePhoto );
```

2.1.2. Domain rules

The characteristics and semantics of a domain are described by domain rules and functional dependencies. Domain rules describe a condition that always holds true for all entities in the domain. These can be used for query validation and optimization of query plans.

Example 3:

A simple fact that the latitude of an earthquake should lie between -90^o and 90^o can be described by the following rule on the ontology *Earthquake*:

```
latitude >= -90 and latitude <= 90
```

The ontology can then be represented as follows:

```
Earthquake ( eventDate, description, region, magnitude,
             latitude, longitude, numberOfDeaths,
             damagePhoto,
             latitude >= -90 and latitude <= 90 );
```

2.1.3. Functional Dependencies (FD)

A functional dependency specifies that two entities having the same values for all attributes appearing on the left-hand side (LHS) of the FD will have the same values for the corresponding variables appearing on the right-hand side (RHS). The domain rules and FDs help in understanding the characteristics of the domains. Web sources constitute a large portion of the set of resources available to the system. The integrated domain model is a comprehensive model and it is very likely to come across resources that do not provide certain attributes of the domain model. The information about the FDs of a domain is used by the system to retrieve information (attributes) that is missing from a resource by using another resource (an *associate resource*) thereby deducing extra information and retrieving more comprehensive results with the same available resources.

Example 4:

A functional dependency on the ontology Earthquake could be that the region implies the latitude and longitude of the place. This can be represented as:

```
region -> latitude longitude
```

The ontology can therefore be represented as follows:

```
Earthquake ( eventDate, description, region, magnitude,
```

latitude, longitude, numberOfDeaths,

damagePhoto,

latitude >= -90 and latitude <= 90,

region -> latitude longitude);Assume that the system has access to two sources that provide information about earthquakes - *SignificantEarthquakesDB* that provides the *eventDate*, *description, region, magnitude, latitude, longitude, numberOfDeaths,* and *damagePhoto* for significant earthquakes (say, with a magnitude > 5) all over the world and *EarthquakesAfter1990* that provides *eventDate, region, magnitude, numberOfDeaths,* and *damagePhoto* for earthquakes that occurred after January 1, 1990. Suppose the user has the following information request:

"Find all earthquakes with epicenter in a 5000 mile radius area around the location at latitude 60.790 North and longitude 97.570 East."

Assume that the calculation of the distance between two locations, given their latitudes and longitudes, is possible. Also, the ontology for earthquake does have attributes *latitude* and *longitude* that specify these. So the system should be able to answer the query. However, the resource *EarthquakesAfter1990* does not supply values for these attributes. Hence, we would need to return only the information that was available from the resource *SignificantEarthquakesDB*. The results that the user sees will have all significant earthquakes (with magnitude > 5) that occurred in that region. But to be able to return *all* earthquakes, the system can use the knowledge about the domain that *region -> latitude longitude.* The system can check if it can deduce the latitudes and longitudes of epicenters of any earthquakes in EarthquakesAfter1990 by checking the regions of earthquakes available from *SignificantEarthquakesDB*. For all the earthquakes for which the latitude and longitude of the epicenter could be deduced, the system can now check if they fall within the 5000 mile radius area. Using the resource *EarthquakesAfter1990* to answer this query could not have been possible without the use of the FD.

2.2. Resources

Information sources in InfoQuilt are described in terms of their corresponding ontologies. This description is meant to supply details about the resource, which can be used for efficient planning. It consists of a set of attributes, binding patterns (BP), data characteristic (DC) rules, and local completeness (LC) rules. Sections 2.2.1 to 2.2.4 describe each of them and the rationale for including them in the knowledge base. The goal is: [Levy et al., 1996].

- To be able to add, modify and delete resources for an ontology dynamically without affecting the systems knowledge
- To be able to specify the sources in a manner such that one can declaratively query them

- To be able to identify the exact usefulness of resources in the context of a particular IScape and prune the others

2.2.1. Resource Attributes

Since the way the resource perceives its data can be different than our perception represented by the ontology of the domain, the data needs to be mapped after it is retrieved. This allows for interoperability between sources with heterogeneous data [Guntamadugu, 2000]. Additionally, since the ontology is ideally a comprehensive perception of the domain, it is common to come across resources that supply only a part of the information. In other words, a resource may not provide values for all the attributes of the ontology. The resource's description therefore lists the attributes that it can provide.

Example 5:

Consider the resources *SignificantEarthquakesDB* and *EarthquakesAfter1990*. The example mentions a list of attributes that the resources provide values for. The resources would therefore be represented as follows:

```
SignificantEarthquakesDB ( eventDate, description, region,
                           magnitude, latitude, longitude,
                           numberOfDeaths, damagePhoto );


EarthquakesAfter1990 ( eventDate, region, magnitude,
                       numberOfDeaths, damagePhoto );
```

We will use the notation above to represent resources.

2.2.2. Binding Pattern (BP)

The sources accessible to the system that are not databases do not have the ability to execute queries. Most of the web sources fall in this category. However, some web sources have a limited query capability. This is supported by allowing users to search, based on some attributes. These web sources require that values for some attributes be provided to retrieve results. The source can be queried only by specifying values for those attributes. Additionally, some sources may provide finite number of optional ways to query (different sets of attributes). For example, a user can query a site providing information about movies by actors, director, title, etc. Such query limitations and characteristics of resources are important to consider while planning a query. Some of the common motivations for having such limitations are [Duschka, 1997]:

- *Privacy/Security.* For example, a company might not want one to be able to get a listing of all its employees but allows a search on an employee by name.
- *Performance.* The data could be indexed and would then be more efficient to query the source on the indexed attributes for efficient retrieval.
- *Efficiency.* The data might be enormous and hence it might not be efficient to query all of it. The values supplied for

bound attributes are used to narrow down the search and limit the data retrieved.

They are represented in the knowledge base as a set of binding patterns (BP). A BP is a set of attributes that the system must be able to supply values for in order to query the resource. If the resource has several BPs, the system can select the most appropriate one. The values for the BP can be obtained from the query constraint or provided by some other resource.

Example 6:

Consider the *Flight* ontology that represents a flight from one city to another within United States. Most web sites for air travel reservation have forms that require the user to specify source, destination, dates of travel, etc. One or more combinations of values could be required as input for the web site to obtain any useful information. One of the possible combinations of BP could be:

[fromCity, fromState, toCity, toState, departureDate]

Suppose we use the AirTran Airway web site (*www.airtran.com*) as a source. The resource would then be represented as follows:

```
AirTranAirways ( airlineCompany, flightNumber, fromCity,
                     fromState,     toCity,     toState,
departureDate,
                 fare, departureTime, arrivalTime,
                 [fromCity, fromState, toCity, toState,
                  departureDate] );
```

2.2.3. Data Characteristic (DC) Rules

Data characteristic rules are similar to domain rules discussed in Section 2.1.2 but they apply only to the specific resource. Knowledge about the data characteristics of a resource can be useful for the system to predict whether a resource will provide any useful results for the particular information request. This can be used to prune the resource if it is possible to infer from the DC rules that the resource will not provide any useful results for the Iscape. It can also be used to optimize the query plan by eliminating a constraint if it can be deduced that the constraint will always be true for all the data retrieved from that resource. The DC rules are also used to select an appropriate associate resource for a resource with BP or missing attributes.

Example 7:

Consider the *AirTranAirways* resource introduced in the previous example that provides data for the ontology Flight. We know that it only provides information about all flights operated by AirTran Airways. This characteristic of the resource can be represented as follows:

```
    AirTranAirways ( airlineCompany, flightNumber, fromCity,
                  fromState,     toCity,     toState,
departureDate,
                  fare, departureTime, arrivalTime,
                  [dc] airlineCompany = "AirTran Airways",
                  [fromCity, fromState, toCity, toState,
                  departureDate] );
```

We use [dc] and [lc] to distinguish data characteristic (DC) rules from local completeness (LC) rules. LC rules are described in Section 2.2.4.

Consider the following query using the ontology *Flight*.

"Find all the flights operated by Delta Airlines from Boston, MA to Los Angeles, CA on February 19, 2001."

Clearly, AirTranAirways will not provide any information about a Delta flight. The system can use this knowledge to deduce that the resource AirTranAirways should not be used to answer this query.

Now suppose the user modifies the query to look for flights operated by AirTran Airways. This time the system knows that it can use AirTranAirways. Additionally, it can also eliminate the check for "AirTran Airways" since all flights available from AirTran Airways are known to be operated by AirTran Airways.

2.2.4. Local Completeness (LC) Rules

A local completeness (LC) rule on a resource has the same format as a DC rule or a domain rule. But it has a different interpretation. A characteristic of web sources is overlapping and incomplete information. Hence, using just one source to answer an information request in many cases does not guarantee retrieval of all the possible information. The general approach to this solution has been to use all the sources (for that ontology) and compute a union of the results retrieved from all of them to provide maximum possible information to every request. However, it is also possible to find sources that do provide complete information about some subset of the domain. LC rules are used to model this. They specify that the resource is complete for a subset(s) of information on a particular ontology. In other words, the information contained in the resource is *all* the information for the subset (specified by the rule) of the domain. Hence, the system cannot retrieve any additional information (for that subset) by querying other sources.

Example 8:

Consider the *AirTranAirways* resource used earlier again. We know that information about *all* flights operated by AirTran Airways will be available from it. It is thus locally complete for all flights with airlineCompany = "AirTran Airways".

```
AirTranAirways ( airlineCompany, flightNumber, fromCity,
                 fromState,        toCity,        toState,
departureDate,
                 fare, departureTime, arrivalTime,
                 [dc] airlineCompany = "AirTran Airways",
                 [lc] airlineCompany = "AirTran Airways",
                 [fromCity, fromState, toCity, toState,
      departureDate] );
```

Now, any information request that needs only the subset of flights that are operated by AirTran Airways can use only

AirTranAirways to retrieve data about all such flights. This would be faster than querying all sources available for Flight.

2.3. Inter-Ontological Relationships

The real-world entities that were classified into domains and sub-domains can be related to each other in various ways. These relationships can be very simple like the "is a" relationship implied by inheritance. For example, a nuclear test "is a" man-made disaster and so on. They can also be very complex, for example, an earthquake can "cause" a tsunami. A relationship may involve more than two domains (ontologies). Such complex inter-ontological relationships represent real-world concepts and characteristics. A novel feature of InfoQuilt is that it provides an infrastructure to model and learn about such complex relationships between different ontologies and to use them to specify information requests. Modeling such inter-ontological relationships requires understanding the semantics involved in their interaction and cannot be expressed using only simple relational and logical operators. The relationships already known can be directly modeled (defined) using the Knowledge Builder toolkit. Additionally, we may want to explore hypothetical relationships and formalize them, if they can be established by using the available information sources. Established relationships can be used to study the interaction between the involved ontologies. Further two relationships "A affects B" and "B affects C" could lead to transitive finding "A affects C".

Example 9:

Consider the relationship between a nuclear test and an earthquake. We use the Earthquake ontology from the example 4 here. Suppose we model nuclear test as follows:

```
     NuclearTest ( testSite, explosiveYield, bodyWaveMagnitude,
testType,
                   eventDate, conductedBy, latitude, longitude,
                   bodyWaveMagnitude > 0,
                   bodyWaveMagnitude < 10,
                   testSite -> latitude longitude );
```

We can say that some nuclear test could have "caused" an earthquake if we see that the earthquake occurred "some time after" the nuclear test was conducted and "in nearby region". Notice the use of operators "some time after" and "in nearby region". These are specialized user defined operators that are not a part of the set of relational operators (<, >, <=, etc.). These are possible due to InfoQuilt’s ability to support user defined functions, as described in Section 2.4. For now, assume that there are two functions called dateDifference and distance available to the system. The function dateDifference takes two dates as arguments and returns number of days from date1 to date2. The function distance takes the latitudes and longitudes of two places and calculates the distance between them. Given that we can use these functions, we can represent the relationship as follows:

NuclearTest Causes Earthquake:

dateDifference(NuclearTest.eventDate, Earthquake.eventDate) < 30

AND

distance(NuclearTest.latitude, NuclearTest.longitude, Earthquake,latitude, Earthquake.longitude) < 10000

The values 30 and 10000 here are arbitrary. In fact, a user can try different values to analyze the data. This is a part of the knowledge discovery support that the system's framework provides, as described further in Section 4.

2.4. Operations

A distinguishing feature of InfoQuilt is its framework to support use of operations. As seen in the previous example, we used the functions dateDifference and distance as operators to describe a complex inter-ontological relationship between NuclearTest and Earthquake. The user can also use them to specify constraints in their information requests.

Example 10:

Consider the information request:

"Find all earthquakes with epicenter in a 5000 mile radius area of the location at latitude 60.790 North and longitude 97.570 East"

The system needs to know how it can calculate the distance between two points, given their latitudes and longitudes, in order to check which earthquakes' epicenters fall in the range specified.

Operations are also used to resolve syntactic heterogeneities between sources by providing a fuzzy matching mechanism to map the two sources. Consider the use of functional dependency region -> latitude longitude from example 4 to solve the missing attribute (latitude and longitude) problem of the source EarthquakesAfter1990. It is highly unlikely that the sources SignificantEarthquakesDB and EarthquakesAfter1990 will have exact same values for the attribute region. The value available from one source could be "Nevada, USA" and that from another source could be "NV, USA". The two are semantically equal but syntactically unequal. [Kashyap & Sheth, 1996]

Another important advantage of using operations is that the system can support complex post-processing of data. An interesting form of post-processing is the use of simulation programs. These independent programs can be integrated with the system. For instance, researchers in the field of Geographic Information Systems (GIS) use simulation programs to forecast characteristics like urban growth in a region based on a model. InfoQuilt supports the use of such simulations like any other operation. They provide valuable additional information that is not often available from the resources directly.

Example 11:

Clarke’s Urban Growth Model [Cla] is a model to forecast the urban growth in a region based on information about the areas in the region where it is known that growth cannot occur due to some reasons, roads, slopes and vegetation in the region. It requires that this information be specified as a set of maps and generates a series of maps showing the progressive urban growth using a specified time step. This simulation can be run on data that can be retrieved from some resource (or multiple resources) that has the maps needed assuming that they are in the format that the program expects.

To be able to dynamically and easily add new operations, update and delete existing ones, InfoQuilt maintains what is known as a *Function Store*. The Function Store contains information about all the functions known to the system. We will use the terms *function* and *operation* interchangeably in the rest of the chapter. The description of a function contains its name, a description of its functionality, a list of arguments that it takes as inputs along with their types and descriptions of what they are, the type of return value, and information about how the system can use it. The user provides an implementation for the function. Once the implementation is provided and it has been added to the Function Store, the system can make use of it as described earlier.

3. Information Scapes

The goal of the InfoQuilt system is to develop a framework for knowledge discovery support through use of the available resources of data. Users can form complex information requests on data available from multiple heterogeneous distributed resources. Note that we use the term "information request" instead of "query". A query generally explicitly specifies the exact sources (tables in a RDBMS) that need to be used and how the data from these sources should be integrated (join conditions in a RDBMS). Additionally, it does not "understand" what the user is asking. An information request, on the other hand, can understand what the user is enquiring about by embedding semantic information within the request[3]. In InfoQuilt, we refer to them as Information Scapes or *IScapes*. This work has been built upon the work done in [Palsena, 2000], [Shah & Sheth, 1998].

Example 12:

Again consider the following information request.

"Find all earthquakes with epicenter in a 5000 mile radius area of the location at latitude 60.790 North and longitude 97.570 East and find all tsunamis that they might have caused."

In addition to the obvious constraints, the system needs to understand what the user means by saying "*find all tsunamis that might have been caused due to the earthquakes*". The relationship that *an earthquake caused a tsunami* is a complex inter-ontological relationship.

Any system that needs to answer such information requests would need a comprehensive knowledge of the terms involved and how they are related. An IScape is specified in terms of the components of the knowledge base of the system such as ontologies, inter-ontological relationships and operations, which the system understands. This helps the system in understanding the semantics of the request. Additionally, it abstracts the user from having to know the actual sources that will be used by the system to answer it and how the data retrieved from these sources will be integrated.

An IScape is defined as a set of information semantically related in a complex manner. We use XML to represent an IScape. It specifies the ontologies involved, inter-ontological relationships, preset constraint, runtime configurable constraint, how the results should be grouped, any aggregations that need to be computed or constraints that need to be applied to the grouped data and finally the information that needs to be returned in the result to the user. The ontologies in the IScape identify the domains that are involved in the IScape and the inter-ontological relationships specify how they are related to each other. The preset constraint and the runtime configurable constraint are filters used to describe the subset of data that the user is interested in. For example, a user may be interested in earthquakes that occurred in only a certain region and had a magnitude greater than 5. The difference between the preset constraint and the runtime constraint is that the runtime constraint can be set at the time of executing the IScape. In the IScape template certain constraints are pre-set, while some others can be configured at execution. The runtime constraint forms an important mechanism to support the knowledge discovery, as described in Section 4. Constraints are essentially conjunctive boolean expressions. The results of the IScape can be grouped based on attributes and/or specified functions. If the results are to be grouped, the user can also specify any aggregations that the user wants to be returned as a part of the result or specify additional constraint on the groups formed (similar to the HAVING clause in the SELECT statement in SQL). The aggregates supported by the system are sum (SUM), average (AVG), count (COUNT), minimum (MIN) and maximum (MAX). Finally, the user specifies a list of all the information that is to be returned as a part of the result. We refer to this as the *projection list*. It could include certain information modeled by the ontologies directly (attributes), aggregates, values of functions evaluated on the data, and results of some simulation programs.

A graphical toolkit known as the *IScape Builder* is available for the user to construct and execute these IScapes and analyze the results. It provides a platform for ease of development and deployment of IScapes using the knowledge base without having to understand the underlying formats used by the system.

4. Knowledge Discovery

InfoQuilt provides a framework that allows users to access data available from a multitude of diverse autonomous distributed resources and provide tools that help them to analyze the data to gain a better understanding of the domains and the inter-domain relationships as well as help users to explore the possibilities of new relationships. This section discusses the conceptual framework using a series of examples, while the next section discusses some of the software components of the InfoQuilt framework.

The inter-ontological relationships defined in the knowledge base of the system describe the interaction that exists between domains as they contain embedded semantic information. Existing relationships provide a scope for discovering new aspects of relationships through transitive learning as discussed in example 13.

Example 13:

Consider the ontologies *Earthquake, Tsunami* and *Environment*. Assume that the relationships *Earthquake affects*

Environment, *Earthquake causes Tsunami* and *Tsunami affects Environment* are defined and known to the system. We can see that since Earthquake causes a Tsunami and Tsunami affects the environment, effectively this is another way in which an earthquake affects the environment (by causing a tsunami). If this aspect of the relationship between an earthquake and environment was not considered earlier, it can be studied further.

Another valuable source of knowledge discovery is studying existing IScapes that make use of the ontologies, their resources and relationships to retrieve information that is of interest to the users. The results obtained from IScapes can be analyzed further by post processing of the result data. For example, the Clarke UGM model forecasts the future patterns of urban growth using information about urban areas, roads, slopes, vegetation in those areas and information about areas where no urban growth can occur.

For the users that are well-versed with the domain, the InfoQuilt framework supports exploring new relationships. The data available from various sources can be analyzed by using charts, statistical analysis techniques, etc. to study and explore trends or aspects of the domain. Such analysis can be used to hypothesize new relationships between domains and to see if the data invalidates the hypothesis or supports it sufficiently as demonstrated by example 14.

Example 14:4

Several researchers in the past have expressed their concern over nuclear tests as one of the causes of earthquakes and suggested that there could be a direct connection between the two. The underground nuclear tests cause shock waves, which travel as ripples along the crust of the earth and weaken it, thereby making it more susceptible to earthquakes. Although this issue has been addressed before, it still remains a hypothesis that is not conclusively and scientifically proven. Suppose we want to explore this hypothetical relationship.

Consider the NuclearTest and Earthquake ontologies again. We assume that the system has access to sufficient resources for both the ontologies such that they together provide sufficient information for the analysis. These resources include non-traditional, non-database sources like web-based sources.

If the hypothesis is true, then we should be able to see an increase in the number of earthquakes that have occurred after the nuclear testing started. We proceed as follows. First, we check to see when nuclear testing began.

IScape 1:

"*Find when was the earliest recorded nuclear test conducted.*"

We find that nuclear testing began in 1950. Next we check the trend of the number of earthquakes that have occurred since the nuclear testing started. It is believed that some earthquakes below the intensity of 5.8 on the Richter scale would have passed unrecorded in the earlier part of the century when measuring devices were less sensitive and less ubiquitous. But for bigger earthquakes, the records are detailed and complete [Whiteford, 1989]. We therefore check the number of earthquakes with a magnitude 5.8 or higher occurring every year in this century.

IScape 2:

"*Find the total number of earthquakes with a magnitude 5.8 or higher on the Richter scale per year starting from year 1900.*"

We can then plot a chart to analyze the trend in the number of earthquakes occurring every year. It reveals that there seems to be a sudden increase in the number of earthquakes since 1950. We modify the query to try to approximately quantify this rise.

IScape 3:

"Find the average number of earthquakes with a magnitude 5.8 or higher on the Richter scale per year for the period 1900-1949 and for the period 1950-present."

We see that in the period 1900-1949, the average rate of earthquakes was 68 per year and that for 1950-present[5] was 127 per year, that is it almost doubled [Whiteford, 1989].

Next, we try to analyze the same data grouping the earthquakes by their magnitudes.

IScape 4:

"For each group of earthquakes with magnitudes in the ranges 5.8-6, 6-7, 7-8, 8-9, and magnitudes higher than 9 on the Richter scale per year starting from year 1900, find the average number of earthquakes."

The results show that the average number of earthquakes with magnitude greater than 7 on the Richter scale have remained practically constant over the century (about 19) [Whiteford, 1989]. We can therefore deduce that the earthquakes caused by nuclear tests usually are of magnitudes less than 7 on the Richter scale. We can then try to explore the data at a finer level of granularity by trying to look for specific instances of earthquakes that occurred within a certain period of time after a nuclear test was conducted in a near by region.

IScape 5:

"*Find nuclear tests conducted after January 1, 1950 and find any earthquakes that occurred not later than a certain number of days after the test and such that its epicenter was located no farther than a certain distance from the test site.*"

Note the use of "*not later than a certain number of days*" and "*no farther than a certain distance*". The IScape does not specify the value for the time period and the distance. These are defined as runtime configurable parameters, which the user can use to form a constraint while executing the IScape. The user can hence supply different values for them and execute the IScape repeatedly to analyze the data for different values without constructing it repeatedly from scratch. This is where runtime configurable constraint is useful. Also, note the use of functions as user-defined operators to calculate the difference in date (dateDifference) and the distance between the epicenter of the earthquake and the nuclear test site (distance). Some of the interesting results that can be found by exploring earthquakes occurring that occurred no later than 30 days after the test and with their epicenter no farther than 5000 miles from the test site are listed below.

China conducted a nuclear test on October 6, 1983 at Lop Nor test site. USSR conducted two tests, one on the same day and another on October 26, 1983, both at Easter Kazakh or Semipalitinsk test site. There was an earthquake of magnitude 6 on the Richter scale in Erzurum, Turkey on October 30, 1983, which killed about 1300 people. The epicenter of the earthquake was about 2000 miles away from the test site in China and about 3500 miles away from the test site in USSR. The second USSR test was just 4 days before the quake.

USSR conducted a test on September 15, 1978 at Easter Kazakh or Semipalitinsk test site. There was an earthquake in Tabas, Iran on September 16, 1978. The epicenter was about 2300 miles away from the test site.

More recently, India conducted a nuclear test at its Pokaran test site in Rajasthan on May 11, 1998. Pakistan conducted two nuclear tests, one on May 28, 1998 at Chagai test site and another on May 30, 1998. There were two earthquakes that occurred soon after these tests. One was in Egypt and Israel on May 28, 1998 with its epicenter about 4500 miles away from both test sites and another in Afghanistan, Tajikistan region on May 30, 1998, with a magnitude of 6.9 and its epicenter about 750 miles away from the Pokaran test site and 710 miles from Chagai test site.

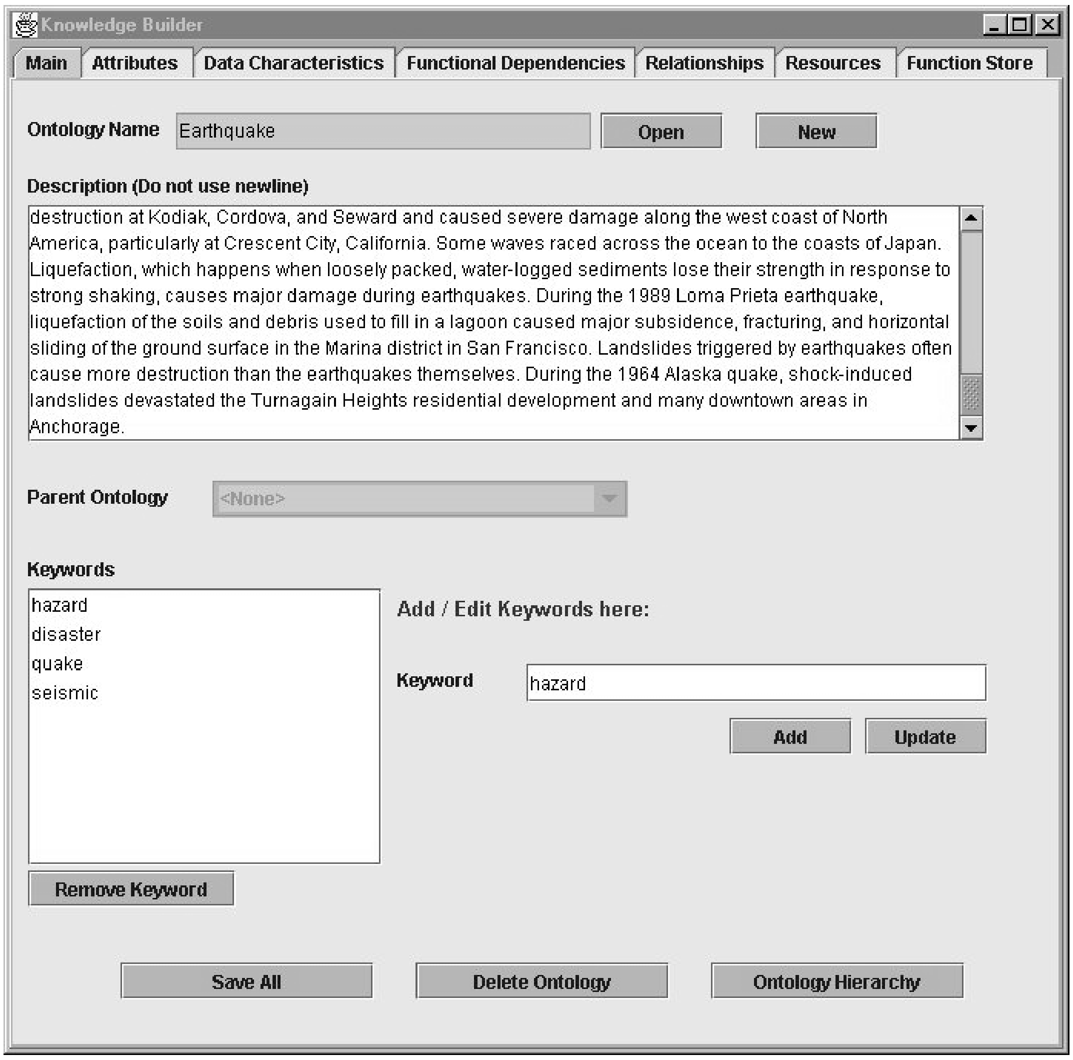
5. Visual Interfaces

This section describes the graphical tools we provide to aid in creation of the knowledge base, creation and execution of Iscapes and for monitoring the execution of Iscapes. These components of the InfoQuilt system are most relevant to the knowledge discovery which is the focus of this chapter, the remaining components - primarily the distributed agent based run-time system is described in [Patel & Sheth, 2001].

5.1. Knowledge Builder

The Knowledge Builder (KB) is the graphical toolkit that helps the user in creating the knowledge base of the system. It allows the user to declaratively specify ontologies, relationships, resources and functions (including simulations). Use of KB provides the following advantages:

- The user does not need to know the format of the XML specification used by the system to represent ontologies, relationships and functions. The KB provides an easy to follow graphical interface that the users can use and thereby provides an abstraction from the details of the formats used internally by the system. See Figure 10.2.



Ontology Name: Earthquake

Description (Do not use newline)

destruction at Kodiak, Cordova, and Seward and caused severe damage along the west coast of North America, particularly at Crescent City, California. Some waves raced across the ocean to the coasts of Japan. Liquefaction, which happens when loosely packed, water-logged sediments lose their strength in response to strong shaking, causes major damage during earthquakes. During the 1989 Loma Prieta earthquake, liquefaction of the soils and debris used to fill in a lagoon caused major subsidence, fracturing, and horizontal sliding of the ground surface in the Marina district in San Francisco. Landslides triggered by earthquakes often cause more destruction than the earthquakes themselves. During the 1964 Alaska quake, shock-induced landslides devastated the Turnagain Heights residential development and many downtown areas in Anchorage.

Parent Ontology: <None>

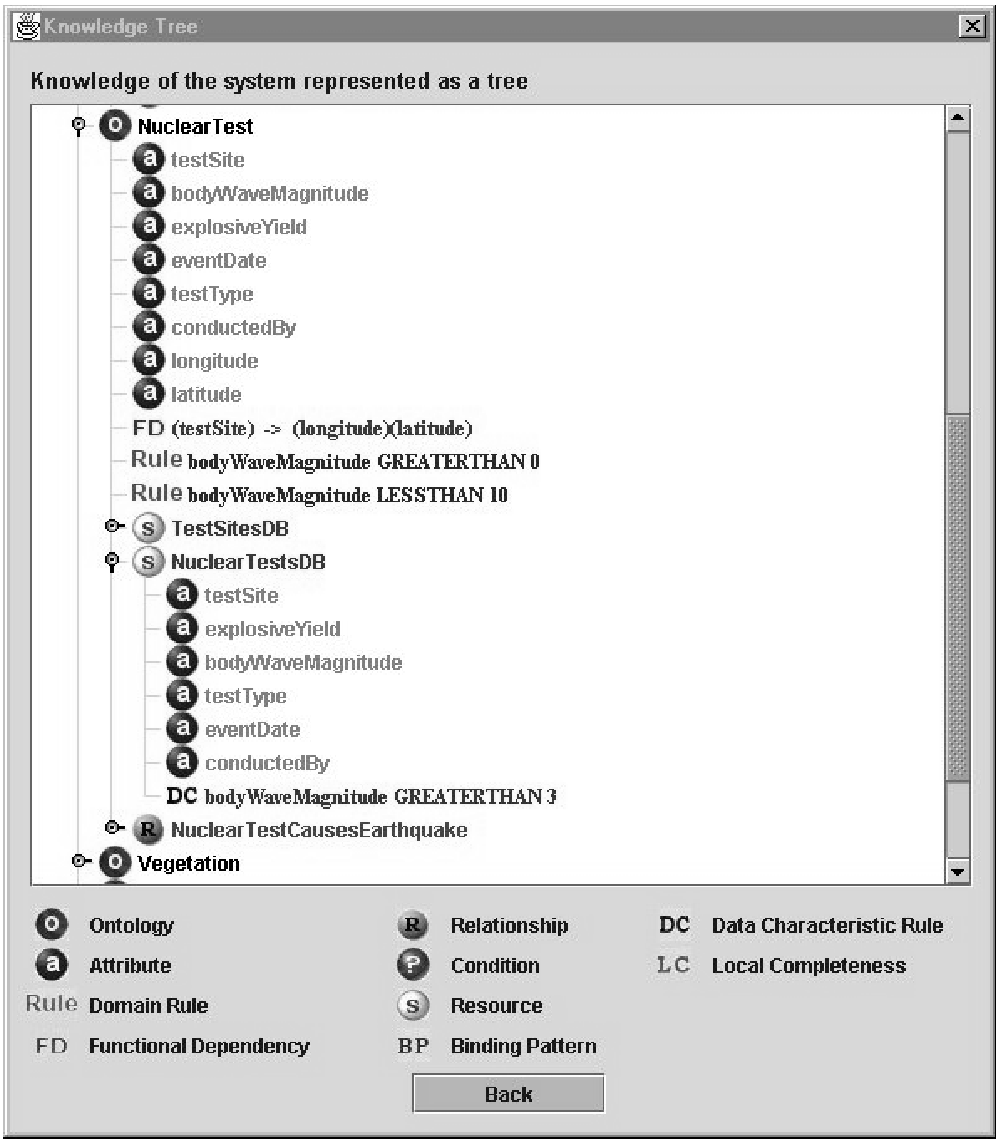
Keywords
hazard
disaster
quake
seismic

Keyword: hazard

Figure 10.2 Knowledge Builder.

- The KB provides tools that help the user relate the information in the knowledge base in a better way. For example, the user can look at the entire knowledge base as a graphical tree that lists all the ontologies defined in the system, their details including rules and FDs defined on them, relationships involving the ontology with their details and resources available for the ontologies with their details like the attributes they supply, the data characteristic rules, the local completeness rules and the binding patterns that they need.

- The KB structures the knowledge in a manner that allows for easy access to knowledge and in the form that can help one understand and learn about a domain.
- The KB also helps the user while trying to modify the knowledge base. For example, a user may want to remove an attribute from an ontology. The KB will not allow this if the attribute is being used in a rule on the ontology, a FD on the ontology, set of attributes provided by some resource, a data characteristic rule on a resource, a local completeness rule on a resource or a binding pattern on a resource. The user is required to manually take care of these before removing the attribute. If the knowledge base is huge, it may be a tedious and error-prone task to go through the specifications of all these to find where the attribute is being used. Even worse, the user would have to go through the XML specifications in the absence of a tool like KB to correctly make the modification. Using the graphical tree display provided by the KB, the user finds it easier locate such uses of the attribute. See Figure 10.3.


Knowledge Tree

Knowledge of the system represented as a tree

```
NuclearTest
    testSite
    bodyWaveMagnitude
    explosiveYield
    eventDate
    testType
    conductedBy
    longitude
    latitude
    FD (testSite) -> (longitude)(latitude)
    Rule bodyWaveMagnitude GREATERTHAN 0
    Rule bodyWaveMagnitude LESSTHAN 10
    TestSitesDB
    NuclearTestsDB
        testSite
        explosiveYield
        bodyWaveMagnitude
        testType
        eventDate
        conductedBy
        DC bodyWaveMagnitude GREATERTHAN 3
    NuclearTestCausesEarthquake
Vegetation
```

O	Ontology	R	Relationship	DC	Data Characteristic Rule
a	Attribute	?	Condition	LC	Local Completeness
Rule	Domain Rule	S	Resource		
FD	Functional Dependency	BP	Binding Pattern		

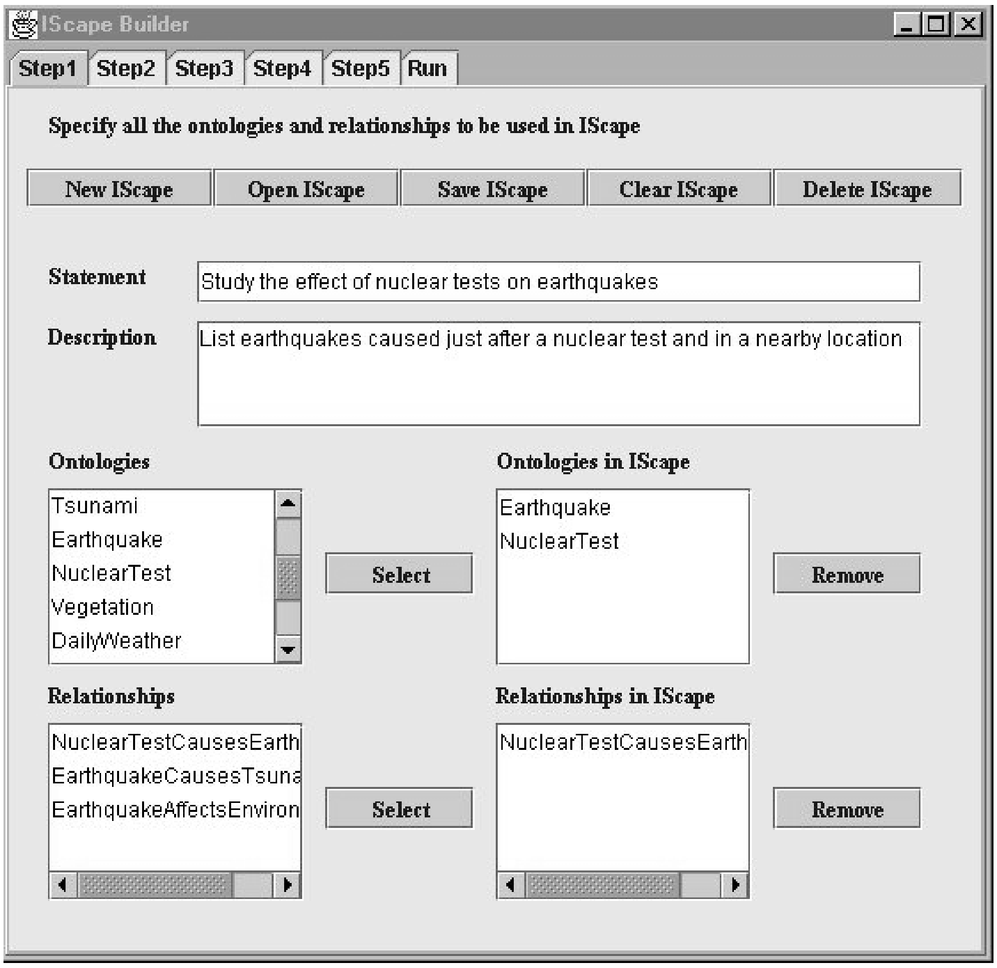
Back

Figure 10.3 Ontology hierarchy.

5.2. IScape Builder

The IScape Builder (IB) is a stand-alone Java application that provides a graphical interface to create and execute IScapes. Use of this tool has the following advantages:

- It provides a simple and intuitive interface that allows the user to create and execute IScapes in a step-by-step manner. See Figure 10.4.



Specify all the ontologies and relationships to be used in IScape

Statement: Study the effect of nuclear tests on earthquakes

Description: List earthquakes caused just after a nuclear test and in a nearby location

Ontologies: Tsunami, Earthquake, NuclearTest, Vegetation, DailyWeather

Ontologies in IScape: Earthquake, NuclearTest

Relationships: NuclearTestCausesEarth, EarthquakeCausesTsuna, EarthquakeAffectsEnviron

Relationships in IScape: NuclearTestCausesEarth

Figure 10.4 Iscape Builder

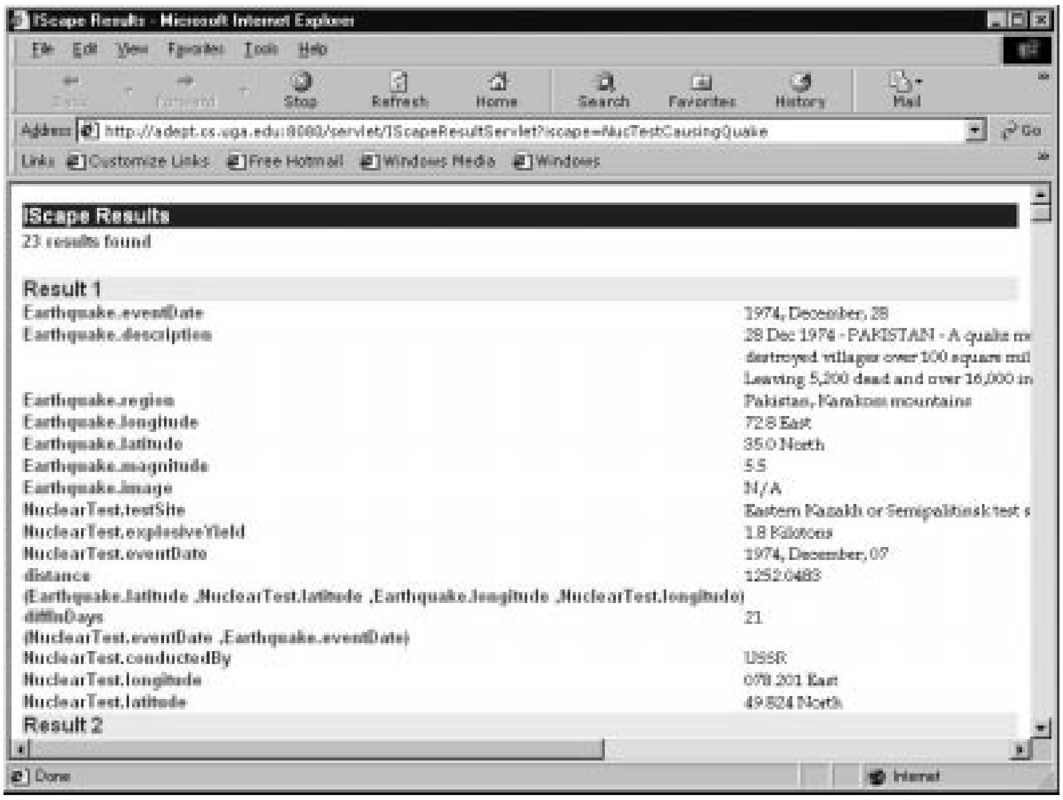
- The user does not need to be aware of the format of the XML specification used internally by the system to represent IScapes.
- It is aware of the knowledge base of the system. The user therefore does not need to look it up to create new IScapes. For example, names of ontologies, relationships and functions appear in combo boxes wherever required. The user can easily make his selection from there.
- It implements basic validity checks to make sure that the IScape being created is valid. For example, if it involves

use of a function call, the tool makes sure that the user has specified where the values for all the arguments to the function should be supplied from. It also performs a type match.

- It can provide various tools to help users better analyze the results of the IScapes. For example, the current version of the IB provides a charting tool, which allows the user to create charts to analyze the results. This helps in providing a learning environment to the users.

5.3. Web-accessible interface to execute IScapes

- We also provide a web-accessible interface that provides a learning environment and allows execution of already existing IScapes.
- The interface is web-accessible. It allows execution of existing IScapes by setting the runtime constraint. However, it does not allow users to create new IScapes. The result of executing the NuclearTest causes Earthquake is demonstrated in Figure 10.5.
- It describes the entire knowledge base of the system that helps the users in understanding the domains that are modeled by the ontologies, how the ontologies are related to each other in complex ways (inter-ontological relationships like affects, causes, etc) and the functions and simulations available in the system.


IScape Results - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail
Address http://adept.cs.uga.edu:8080/servlet/IScapeResultServlet?iscape=NucTestCausingQuake
Go
Links Customize Links Free Hotmail Windows Media Windows
Done
Internet

IScape Results

23 results found

Result 1

Earthquake.eventDate	1974, December, 28
Earthquake.description	28 Dec 1974 - PAKISTAN - A quake re destroyed villages over 100 square mil Leaving 5,200 dead and over 16,000 in
Earthquake.region	Pakistan, Karakom mountains
Earthquake.longitude	72.8 East
Earthquake.latitude	35.0 North
Earthquake.magnitude	5.5
Earthquake.image	N/A
NuclearTest.testSite	Eastern Kazakh or Semipalatinsk test s
NuclearTest.explosiveYield	1.8 Kilotons
NuclearTest.eventDate	1974, December, 07
distance (Earthquake.latitude ,NuclearTest.latitude ,Earthquake.longitude ,NuclearTest.longitude)	1252.0483
diffInDays (NuclearTest.eventDate ,Earthquake.eventDate)	21
NuclearTest.conductedBy	USSR
NuclearTest.longitude	078.201 East
NuclearTest.latitude	49.824 North

Result 2

Figure 10.5 Iscape Results using the Web interface

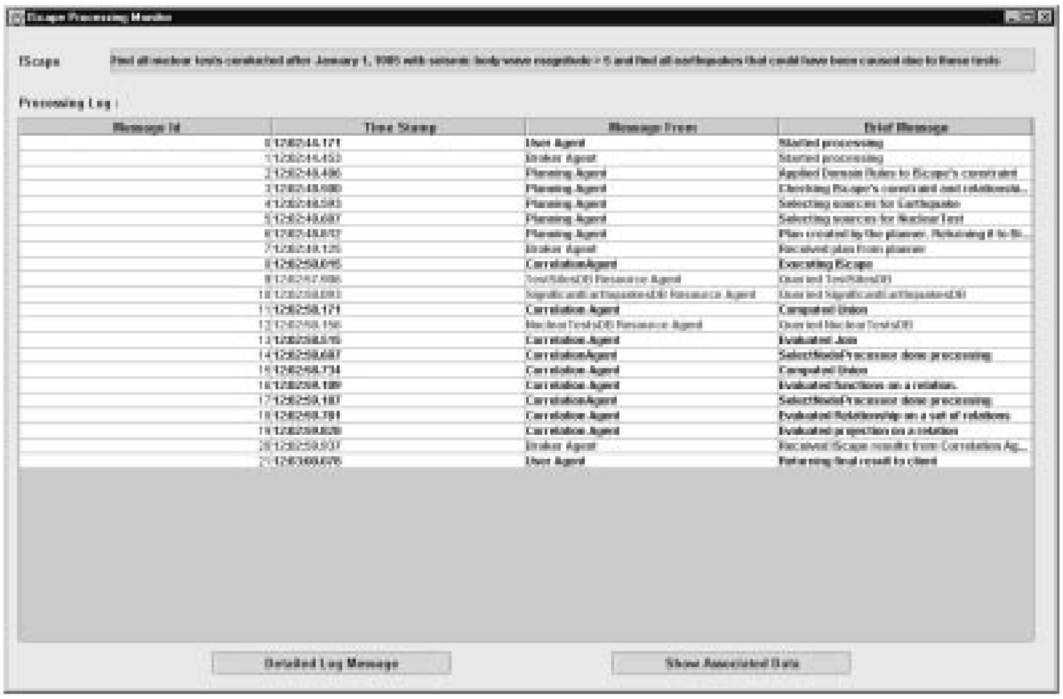
5.4. IScape Processing Monitor

The IScape Processing Monitor (IPM) is a graphical interface used to monitor the execution of IScapes. The user can choose to run the IScape with or without the processing monitor. The web-accessible interface however does not support the monitor.

The various agents in the runtime system generate log entries that are sent to the monitor and displayed as color-coded table. The log entries can be very detailed. They include a time stamp, the name of the agent sending the log, a brief message, a more descriptive message and associated data, if any. The associated data could be, for example, the execution plan generated by the planner or the results at various stages in the IScape execution.

The IPM is very useful in the following ways:

- It helps in monitoring the execution of the IScape as it proceeds and locate any failures easily.
- The availability of time-stamps with all the log entries allows us to evaluate which phases of the IScape processing are taking the most time. This helps us evaluate our system better and identify areas that need improvement.



IScape: Find all nuclear tests conducted after January 1, 1985 with seismic body wave magnitude > 5 and find all earthquakes that could have been caused due to these tests.

Processing Log :

Message Id	Time Stamp	Message From	Brief Message
0	1282244.171	User Agent	Started processing
1	1282244.453	Broker Agent	Started processing
2	1282248.406	Planning Agent	Applied Domain Rules to IScape's constraint
3	1282248.500	Planning Agent	Checking IScape's constraint and relationship...
4	1282248.593	Planning Agent	Selecting sources for Earthquake
5	1282248.687	Planning Agent	Selecting sources for Nuclear Test
6	1282248.872	Planning Agent	Plan created by the planner. Returning it to Br...
7	1282249.125	Broker Agent	Received plan from planner
8	1282250.095	CorrelationAgent	Executing IScape
9	1282257.906	TestSitesDB Resource Agent	Queried TestSitesDB
10	1282258.093	SignificantEarthquakesDB Resource Agent	Queried SignificantEarthquakesDB
11	1282258.171	Correlation Agent	Computed Union
12	1282258.156	NuclearTestsDB Resource Agent	Queried NuclearTestsDB
13	1282258.515	Correlation Agent	Evaluated Join
14	1282258.687	CorrelationAgent	SelectNodeProcessor done processing
15	1282258.734	Correlation Agent	Computed Union
16	1282259.109	Correlation Agent	Evaluated functions on a relation.
17	1282259.187	CorrelationAgent	SelectNodeProcessor done processing
18	1282259.781	Correlation Agent	Evaluated Relationship on a set of relations
19	1282259.828	Correlation Agent	Evaluated projection on a relation
20	1282259.937	Broker Agent	Received IScape results from Correlation Ag...
21	1282300.828	User Agent	Returning final result to client

Figure 10.6 Iscape Processing Monitor.

- The detailed log messages generated by various agents describe in considerable detail exactly what is going on during processing. The IPM is therefore extremely useful as a high-level debugging tool.

6. Related Work

SIMS [AKS96], TSIMMIS [CMH+94], Information Manifold [Levy et al., 1996], OBSERVER [Mena et al., 2000] and InfoSleuth [Bayardo, 1997] are some of the efforts to integrate information from multiple heterogeneous sources for querying. Most other systems focus on retrieving and integrating "data" from multiple sources and not on the "learning, exploring and understanding" aspects. The following are the features of InfoQuilt that are not supported by any other systems:

- A framework to study, analyze and learn about domains and inter-domain relationships.
- Functions and simulations to post-process the data retrieved from the resources thereby adding value to the data
- Complex relationships and constraints representing the semantic correlation of information across multiple domains that cannot be expressed using relational and logical operators can be modeled
- Powerful semantic query interface (IScapes)
- Additionally, in comparing the general approach of information integration to the other systems we identify the following noteworthy differences.

The mediator in SIMS [Arens, Knoblock and Shen, 1996] system is specialized to a single application domain, which is source dependent and query dependent. An application domain in SIMS models a single hierarchy of classes. It does not support inter-ontological relationships. SIMS uses Loom [Gre90], a member of the KL-ONE family knowledge representation systems to model domains as well as describe information sources. They do not consider use of local completeness information about sources and support only one binding pattern per web resource.

OBSERVER [Mena et al., 2000] uses ontologies to describe information sources and inter-ontology relationships like synonyms, hyponyms and hypernyms across terms in different ontologies to be able to translate a query specified using some ontology that the user selected into another query that uses related ontologies describing relevant information. This approach of using relationships to achieve interoperability between the sources is limited to basic relationships.

TSIMMIS [Chawathe et al., 1994],[Garcia-Molina, 1995] uses a mediator-based architecture [Wiederhold, 1993] with a query-centric approach. It uses Mediator Specification Language (MSL) to define mediators. The specification encodes how the system should use the available resources. The mediators are then generated automatically from these specifications. Since the MSL definitions need to be created manually, adding or removing information sources is not easy as it requires updating them after determining how the sources should be used to answer the queries and then recompiling them. It has a set of pre-defined query templates that it knows to answer. User queries are then answered by relating them to the existing query templates. The types of queries that can be answered using this approach are limited compared to our system.

Information Manifold (IM) [Levy et al., 1996] uses an approach similar to ours in that the user creates a *world view*, a collection of virtual relations and classes. The world view however does not capture semantics of the domains as InfoQuilt can using domain rules and FDs. Information sources are described to the system as a query over the relations in the world view. Locally complete resources cannot be modeled precisely. IM uses capability records to capture query capability limitations of sources. These records specify, among others, a set of input parameters and minimum and maximum number of inputs allowed. The system then arbitrarily selects a subset of the set of input parameters with at least the minimum number of allowed parameters in the set. The subset selected is arbitrary. Therefore, the capability records cannot precisely specify the binding patterns.

Semantic network is a related concept in terms of modeling knowledge and relationships [Mylopoulu et al., 1990]. Unlike information integration systems, the main purpose of semantic network is to create a logical, orderly and aesthetically consistent relationship of pages (page is an elemental unit of semantic network). The relationships modeled are hierarchical or similarity based are can be used to understand semantics involved within and across domains, but are limited in terms of their ability to model the complex semantic relationships like the "cause" or "affects" relationships.

Conclusions

This chapter discussed our approach for knowledge discovery on the evolving Semantic Web. It involves modeling knowledge in terms of complex relationships among Web-accessible semantically related but otherwise heterogeneous information. Iscape, a representation used for this comprises of ontologies, inter-ontological relationships, information sources and complex operations including functions and simulations. The chapter described the use of IScapes to construct and deploy complex information requests. Of particular interest was the use of inter-ontological relationships and functions to answer those requests. Simulations can also be integrated with the system to perform post-query analysis on the result. Iscape also form the basis for knowledge discovery, through the ability to study and explore new (complex) relationships.

The IScapes, inter-ontological relationships, support for functions and knowledge discovery are the distinguishing features of InfoQuilt. The system can easily adapt to new sources of information and is very scalable. Further, the system has been implemented and it makes use of query planning and optimization with a multi-threaded execution to exploit the parallelism in the plans [Patel & Sheth, 2001].

Some of the improvements to the system planned are as follows. The system will be enhanced to allow the use of recursive query plans, which would further expand its query capability. It needs to make use of inductive learning to infer rules and use them appropriately, which can speed up query processing [Hsu & Crai, 1994]. In using the current framework to support simulations, further investigation might be necessary to support simulations with more complex interactions.

References

[Arens, Hsu and Knoblock, 1996] Y. Arens, C. Hsu and C. A. Knoblock. Query processing in the SIMS information mediator. In Austin Tate, editor, Advanced Planning Technology. The AAAI Press, Menlo Park, CA, 1996.

[Arens, Knoblock and Shen, 1996] Y. Arens, C. A. Knoblock, and W. Shen. Query reformulation for dynamic information integration. Journal of Intelligent Information Systems, Vol. 6, pp. 99-130, 1996.

[Bayardo, 1997] R. J. Bayardo Jr., W. Bohrer, R. Brice, et al. InfoSleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Environments. In SIGMOD-97, pp. 195-206, Tucson, AZ, USA, May 1997.

[Bertram, 1998] C. Bertram. InfoQuilt: Semantic Correlation of Heterogeneous Distributed Assets. Masters Thesis, Computer Science Department, University of Georgia, 1998.

[Cla] Clarke’s Urban Growth Model, Project Gigalopolis, Department of Geography, University of California, Santa Barbara. http://www.ncgia.ucsb.edu/projects/gig/ncgia.html

[Chawathe, 1994] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS Project: Integration of Heterogeneous Information Sources. Proceedings of 10th Anniversary Meeting of the Information Processing Society of Japan, pp. 7-18, Tokyo, Japan, 1994.

[Duschka, 1997] O. M. Duschka. Query Planning and Optimization in Information Integration. Ph. D. Thesis, Computer Science Department, Stanford University, 1997.

[Fensel & Musen, 2001] D., Eds., The Semantic Web: A Brain for Humankind, Special issue of IEEE Intelligent Systems on Semantic Web Technology, March-april 2001.

[Garcia-Molina, 1995] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. In Proceedings of NGITS (Next Generation Information Technologies and Systems), 1995.

[Guntamadugu, 2000] M. Guntamadugu. MÉTIS: Automating Metabase Creation from Multiple Heterogeneous Sources. Masters Thesis, Computer Science Department, University of Georgia, 2000

[Hsu & Crai, 1994] Chun-Nan Hsu and Craig A. Knoblock. Rule Induction for Semantic Query Optimization. Proceedings on the 11th International Conference on Machine Learning, San Mateo, CA, 1994

[Kashyap & Sheth, 1996] V. Kashyap, A. Sheth. Schematic and Semantic Similarities between Database Objects: A Context-based Approach - in the VLDB Journal 5 (4).

[Kashyap & Sheth, 1997] V. Kashyap, A. Sheth. Semantic Heterogeneity in Global Information Systems: The Role of Metadata, Context and Ontologies. M. Papzoglou, and G. Schlageter, (Eds.), Academic Press,1997

[Kashyap & Sheth, 2000] V. Kashyap. A. Sheth. Information Brokering Across Heterogeneous Digital Data – A Metadata-based Approach. Kluwer Academic Publishers, 2000.

[Laksminarayanan, 2000] Laksminarayanan S. Achieving Semantic Inter-operability in Digital Libraries by use of Inter-Ontological Relations. Masters Thesis, Computer Science Department, University of Georgia, 2000.

[Berners-Lee, Hendler & Lassila, 2001] Time Berners-Lee, James Hendler and Ora Lassila. The Semantic Web published as an article in Scientific American. http://www.sciam.com/2001/0501issue/0501berners-lee.html

[Levy et al., 1996] A. Y. Levy, A. Rajaraman, J. J. Ordille. Querying heterogeneous information sources using source descriptions. Proceedings of the 22nd International Conference on Very Large Databases VLDB-96, Bombay, India, September 1996.

[Mylopoulu et al, 1990] J. Mylopoulus, A. Borgida, M. Jarke, M. Koubarakis. Telos: Representing knowledge about information systems. ACM transaction on information systems, 1990.

[Mena et al., 2000] E. Mena, A. Illarramendi, V. Kashyap, A. P. Sheth. OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. International Journal on Distributed and Parallel Databases, Vol. 8, No. 2, pp. 223-271, April 2000.

[Ont] Ontology-based information exchange for knowledge management and electronic commerce. http://www.ontoweb.org.

[Palsena, 2000] N. Palsena. A collaborative Approach to learning Using Information Landscapes. Masters Thesis, Computer Science Department, University of Georgia, 2000.

[Patel & Sheth, 2001] S. Patel and A. Sheth. Planning And Optimizing Semantic Information Requests On Heterogeneous Information Sources Using Semantically Modeled Domain And Resource Characteristics, 2001. LSDIS Technical Report, University of Georgia, March 2001

[Sheth, 1996] A. Sheth, "Data Semantics: What, Where, and How?", in Data Semantics (IFIP Transactions), R. Meersman and L. Mark, Eds., Chapman and Hall, London, 1996, pp. 601-610.

[Sheth, 1999] A. Sheth. Changing focus on Interoperability: From System, Syntax, Structure to Semantics. In M. Goodchild, M. Egenhofer, R. Fegeas, and C. Kottman, editors, Interoperating Geographic Information Systems, Kluwer Academic Publishers, 1999.

[Shah & Sheth, 1998] K. Shah and A. Sheth. Logical Information Modeling of Web-accessible Heterogeneous Digital Assets: Proceedings of the Forum on Research and Technology Advances in Digital Libraries (ADL), Santa Barbara, CA, April, 1998.

[Wiederhold, 1993] G. Wiederhold. Mediators in the architecture of future information systems. IEEE Computer, 25(3), pp. 38-49.

[Wiederhold, 1997] G. Wiederhold. Value-added mediation in Large-Scale Information Systems. Database Application Semantics, Chapman and Hall, 1997.

[Whiteford, 1989] G. T. Whiteford. Earthquakes and Nuclear Testing: Dangerous Patterns and Trends. In proceedings of the 2nd Annual Conference on the United Nations and World Peace, Seattle, Washington, April 1989.

Footnotes

[1]Large Scale Distributed Information Systems (LSDIS) Lab, Department of Computer Science, University of Georgia, Athens, GA 30602 USA, http://lsdis.cs.uga.edu,Email: {sanjeev, amit, shuchi}@cs.uga.edu

[2]One of the incarnations of the InfoQuilt system, as applied to the geographic information as part of the NSF Digital Library initiative is the ADEPT-UGA system http://alexandria.uscb.edu/

[3] Use of ontologies, context and relationships are critical in defining information requests and in supporting semantics – see for example DS-6 proceedings, esp. [Wie97] and [She96]

[4] The information presented in this example is based on the findings of Gary T. Whiteford published in the paper "Earthquakes and Nuclear Testing: Dangerous Patterns and Trends" presented at the 2nd Annual Conference on the United Nations and World Peace in Seattle, Washington on April 14, 1989. It has been recognized as the most exhaustive study yet of the correlation between nuclear testing and earthquakes. [Whi89]

[5] The period of 1950-1989 implies the period 1950-1989, since the data presented here was published by Gary T. Whiteford in 1989.

11. SEmantic portAL — The SEAL approach

Alexander Maedche[1,3], Steffen Staab[1,2], Nenad Stojanovic[1], Rudi Studer[1,2,3], and York Sure[1,2]

1. Introduction

The widely-agreed core idea of the Semantic Web is the delivery of data on a semantic basis. Intuitively the delivery of semantically apprehended data should help with establishing a higher quality of communication between the information provider and the consumer. How this intuition may be put into practice is the topic of this paper. We discuss means to further communication on a semantic basis. For this one needs a theory of communication that links results from semiotics, linguistics, and philosophy into actual information technology. We here consider *ontologies* as a sound semantic basis that is used to define the meaning of terms and hence to support intelligent access, *e.g.* by semantic querying [Decker et al., 1999] or dynamic hypertext views [Staab et al., 2000]. Thus, ontologies constitute the foundation of our SEAL (Semantic portal) approach. The origin of SEAL lie in Ontobroker [Decker et al., 1999], which was conceived for semantic search of knowledge on the Web and also used for sharing knowledge on the Web [Benjamins & Fensel, 1998]. It then developed into an overarching framework for search and presentation offering access at a portal site [Staab et al., 2000]. This concept was then transferred to further applications [Angele et al., 2000], [Staab & Maedche, 2001], [Sure, Maedche, & Staab, 2000] and is currently extended into a commercial solution *(cf.* http://www.time2research.de).

Here, we describe how we have applied SEAL to a real-world case study, *viz.* the AIFB web site. By the history of SEAL and related projects, we have distilled a methodology for construction of ontology-based knowledge systems that has been applied for the construction of the AIFB web site (Section 3). Following the description of this methodology and the experiences we made with its application to the AIFB site, we describe the SEAL core modules and its overall architecture (Section 4). Thereafter, we go into several technical details that are important for human and machine access to a semantic portal.

In particular, we describe a general approach for semantic ranking (Section 5). The motivation for semantic ranking is that even with accurate semantic access, one will often find too much information. Underlying semantic structures, *e.g.* topic hierarchies, give an indication of what should be ranked higher on a list of results. Then, we tackle the issue of semantic personalization (Section 6). The principal idea is that underlying semantics may be very useful for presenting personalized views, because they allow for content-based views onto the web site. Finally, we present mechanisms to deliver and collect machine-understandable data (Section 7). They extend previous means for better digestion of web site data by software agents. Before we conclude, we give a short survey of related work.

2. Ontology and knowledge base

For our AIFB intranet, we explicitly model relevant aspects of the domain in order to allow for a more concise communication between agents, *viz.* within the group of software agents, between software and human agents, and – last not least – between different human agents. In particular, we describe a way of modeling an ontology that we consider appropriate for supporting communication between human and software agents.

2.1. Ontologies for communication

Research in ontology has its roots in philosophy dealing with the nature and organisation of being. In computer science, the term ontology refers to an engineering artifact, constituted by a specific vocabulary used to describe a particular model of the world, plus a set of explicit assumptions regarding the intended meaning of the words in the vocabulary. Both, vocabulary and assumptions, serve human and software agents to reach common conclusions when communicating.

Reference and meaning. The general context of communication (with or without ontology) is described by the meaning triangle [Odgen & Richards, 1993]. The meaning triangle defines the interaction between symbols or words, concepts and things of the world *(cf.* Figure 11.1).

The meaning triangle illustrates the fact that although words cannot completely capture the essence of a reference (= concept) or of a referent (= thing), there is a correspondence between them.

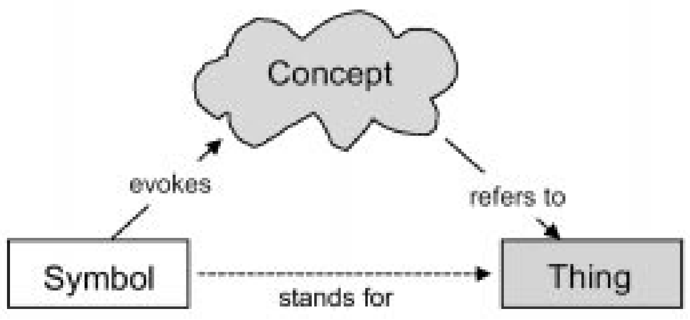


Figure 11.1 The Meaning Triangle.

The relationship between a word and a thing is indirect. The correct linkage can only be accomplished when an interpreter processes the word invoking a corresponding concept and establishing the proper linkage between his concept and the appropriate thing in the world.

Logics. An ontology is a general logical theory constituted by a vocabulary and a set of statements about a domain of interest in some logic language. The logical theory specifies relations between signs and it apprehends relations with a semantics that restricts the set of possible interpretations of the signs. Thus, the ontology reduces the number of mappings from signs to things in the world that an interpreter who is committed to the ontology can perform – in the ideal case each sign from the vocabulary eventually stands for exactly one thing in the world.

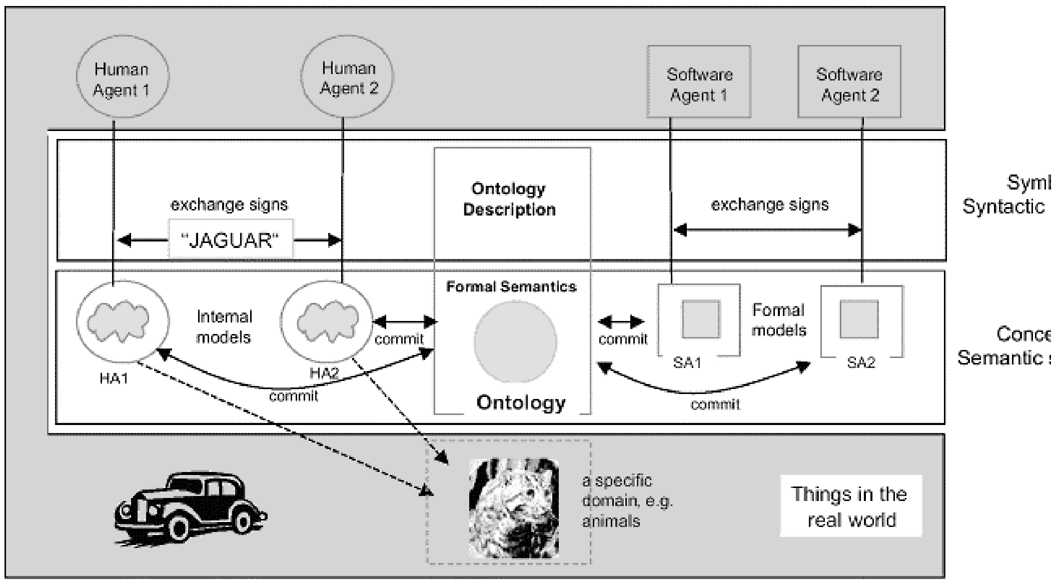


Figure 11.2 Communication between human and/or software agents.

Figure 11.2 depicts the overall setting for communication between human and software agents. We mainly distinguish three layers: First of all, we deal with things that exist in the real world, including in this example human and software agents, cars, and animals. Secondly, we deal with symbols and syntactic structures that are exchanged. Thirdly, we analyze models with their specific semantic structures.

Let us first consider the left side of Figure 11.2 without assuming a commitment to a given ontology.

Two human agents HA1 and HA2 exchange a specific sign, *e.g.* a word like “jaguar”. Given their own internal model each of them will associate the sign to his own concept referring to possibly two completely different existing things in the world, *e.g.* the animal *vs.* the car. The same holds for software agents: They may exchange statements based on a common syntax, however, they may have different formal models with differing interpretations.

We consider the scenario that both human agents commit to a specific ontology that deals *e.g.* with a specific domain, *e.g.* animals. The chance that they both refer to the same thing in the world increases considerably. The same holds for the software agents SA1 and SA2. They have actual knowledge and they use the ontology to have a common semantic basis. When agent SA1 uses the term “jaguar”, the other agent SA2 may use the ontology just mentioned as background knowledge and rule out incorrect references, *e.g.* ones that let “jaguar” stand for the car. Human and software agents use their concepts and their inference processes, respectively, in order to narrow down the choice of referents *(e.g.*, because animals do not have wheels, but cars have).

A new model for ontologies. Subsequently, we define our notion of ontology. However, in contrast to most other research about ontology languages it is not our purpose to invent a new logic language or to redescribe an old one. Rather what we specify is a way of *modelling* an ontology that inherently considers the special role of signs (mostly strings in current ontology-based systems) and references.

Our motivation is based on the conflict that ontologies are for human and software agents, but logical theories are mostly for mathematicians and inference engines. Formal semantics for ontologies is a *sine qua no*n. In fact, we build our applications on a well-understood logical framework, *viz.* F-Logic [Kifer, Lausen, & Wu, 1995] . However, in addition to the benefits of logical rigor, user and developer of an ontology-based system profit from ontology structures that allow to elucidate possible misunderstandings.

For instance, one might specify that the sign “jaguar” refers to the union of the set of all animals that are jaguars and the set of all cars that are jaguars. Alternatively, one may describe that “jaguar” is a sign that may either refer to a concept “animal-jaguar” or to a concept “car-jaguar”. We prefer the second way. In conjunction with appropriate GUI modules *(cf.* Sections 4ff) one may avoid presentations of ‘funny symbols’ to the user like “animal-jaguar”, while avoiding ‘funny inference’ such as may arise from artificial concepts like ‘animal-jaguar’ ‘car-jaguar’).

2.2. Ontology *vs.* knowledge base

Concerning the general setting just sketched, the term ontology is defined – more or less – as some piece of formal knowledge. However, there are several properties that warrant the distinction of knowledge contained in the ontology *vs.* knowledge contained in the so-called *knowledge bas*e, which are summarised in Table 11.1.

Table 11.1 Distinguishing ontology and knowledge base

	Ontology	Knowledge base
Set of logic statements	Yes	Yes
Theory	general theory	theory of particular circumstances
Statements are mostly	intensional	extensional
Construction	set up once	continuous change
Description logics	T-Box	A-Box

The ontology constitutes a general logical theory, while the knowledge base describes particular circumstances. In the ontology one tries to capture the general conceptual structures of a domain of interest, while in the knowledge base one aims at the specification of the given state of affairs. Thus, the ontology is (mostly) constituted by *intensional* logical definitions, while the knowledge base comprises (mostly) the *extensional* parts. The theory in the ontology is one which is mostly developed during the set up (and maintenance) of an ontology-based system, while the facts in the knowledge base may be constantly changing. In description logics, the ontology part is mostly described in the T-Box and the knowledge base in the A-Box. However, our current experience is that it is not always possible to distinguish the ontology from the knowledge base by the logical statements that are made. In the conclusion we will briefly mention some of the problems referring to some examples of following sections.

The distinctions (“general” *vs.* “specific”, “intensional” *vs.* “extensional”, “set up once” *vs.* “continuous change”) indicate that for purposes of development, maintenance, and good design of the software system it is reasonable to distinguish between ontology and knowledge base. Also, they describe a rough shape of where to put which parts of a logical theory constraining the intended semantic models that facilitate the referencing task for human and software agents. However, the reader should note that none of these distinctions draw a clear cut borderline between ontology and knowledge base in general. Rather, it is typical that in a few percent of cases it depends on the domain, the view of the modeler, and the experience of the modeler, whether she decides to put particular entities and relations into the ontology or into the knowledge base.

Both following definitions of ontology and knowledge base specify constraints on the way an ontology (or a knowledge base) should be modeled *in a particular logical language* like F-Logic or OIL:

Definition 1 (Ontology) An ontology is a sign system O := (L, F, G, C, H, R, A), which consists of

A *lexicon:* The lexicon contains a set of signs (lexical entries) for concepts, L^C, and a set of signs for relations, L^R. Their union is the lexicon $L := L^C \cup L^R$..

Two *reference functions* F, G, with F: $2^{Lc} \rightarrow L^C$ and G:$2^{Lr} \rightarrow 2^R$. F and G, link sets of lexical entries {Li} $\subset$ *L* to the set of concepts and relations they refer to, respectively, in the given ontology. In general, one lexical entry may refer to several concepts or relations and one concept or relation may be refered to by several lexical entries. Their inverses are F^{-1} and. G^{-1} .

In order to map easily back and forth and because there is a n to m mapping between lexicon and concepts/relations, F and G are defined on sets rather than on single objects.

A set of *concepts*: About each C $\in \mathcal{C}$ exists at least one statement in the ontology, viz. its *embedding in the taxonomy.*

A *taxonomy* H: Concepts are taxonomically related by the irreflexive, acyclic, transitive relation H,(H $\subset \mathcal{C} * \mathcal{C}$). H(C1, C2) means that is C1 a subconcept of C2.

A set of binary *relations* R: R denotes a set of binary relations[4] They specify pairs of domain and ranges (D, R) with D, R $\in$ C. The functions d and r applied to a binary relation Q yield the corresponding domain and range concepts D and R , respectively.

A set of *ontology axioms*, A.

The reader may note that the structure we propose is very similar to the WordNet model described by Miller [Miller, 1995]. WordNet has been conceived as a mixed linguistic / psychological model about how people associate words with their meaning. Like WordNet, we allow that one word may have several meanings and one concept (synset) may be represented by several words. However, we allow for a seamless integration into logical languages like OIL or F-Logic by providing very simple means for definition of relations and for knowledge bases.

We define a knowledge base as a collection of object descriptions that refer to a given ontology.

Definition 2 (Knowledge Base) We define a knowledge base as a 7- tupel

KB:= (L, J, I, W, S, A, O), that consists of

- a *lexicon* containing a set of signs for instances, L.
- A *reference function* J with $J : 2^L \rightarrow 2^I$.. J links sets of lexical entries $\{L_i\} \subset L$ to the set of instances they correspond to.

Thereby, names may be multiply used, e.g. “Athens” may be used for “Athens, Georgia” or for “Athens, Greece”.

- a set of *instances I*. About each $I_k \in I$, $k = 1,\dots,l$ exists at least one statement in the knowledge base, viz. a membership to a concept C from the ontology O.

- A *membership function* W with W : $2^I \rightarrow 2^C$. W assigns sets of instances to the sets of concepts they are members of.
- *Instantiated relations*, S, are described, viz. S $\subseteq$ {(x, y, z)x$\in$ I, y$\in$ R, z $\in$ I}.
- A set of knowledge base axioms, A.
- A reference to an ontology O.

Overall the decision to model some relevant part of the domain in the ontology *vs.* in the knowledge base is often based on gradual distinctions and driven by the needs of the application. Concerning the technical issue it is sometimes even useful to let the lexicon of knowledge base and ontology overlap, *e.g.* to use a concept name to refer to a particular instance in a particular context. In fact researchers in natural language have tackled the question how the reference function can be dynamically extended given an ontology, a context, a knowledge base and a particular sentence.

3. Ontology engineering

The conceptual backbone of our SEAL approach is the ontology. For our intranet, we had to model the concepts relevant in this setting. As SEAL has been maturing, we have developed a methodology for setting up ontology-based knowledge systems which we sketch here. Its extended description can be found in [Staab et al., 2001]. We also describe some experiences made during the ontology development.

3.1. Methodology for ontology engineering

Until a few years ago the building of ontologies was done in a rather *ad hoc* fashion. Meanwhile there have been some few, but seminal proposals for guiding the ontology development process (*e.g.* [Uschold & Gruninger, 1996], [Gomez-Perez, 1996], [Guarino & Welty, 2000]). For instance Guarino & Welty [Guarino & Welty, 2000] give formal guidelines for constructing a consistent and reusable ontology. Another approach, the Methontology framework [Gomez-Perez, 1996], includes the identification of the ontology development process, and stages through which an ontology passes during its lifetime. In contrast to these methodologies, which mostly restrict their attention within the ontology itself, our approach (*cf.* Figure 11.3) focuses on the application-driven development of ontologies

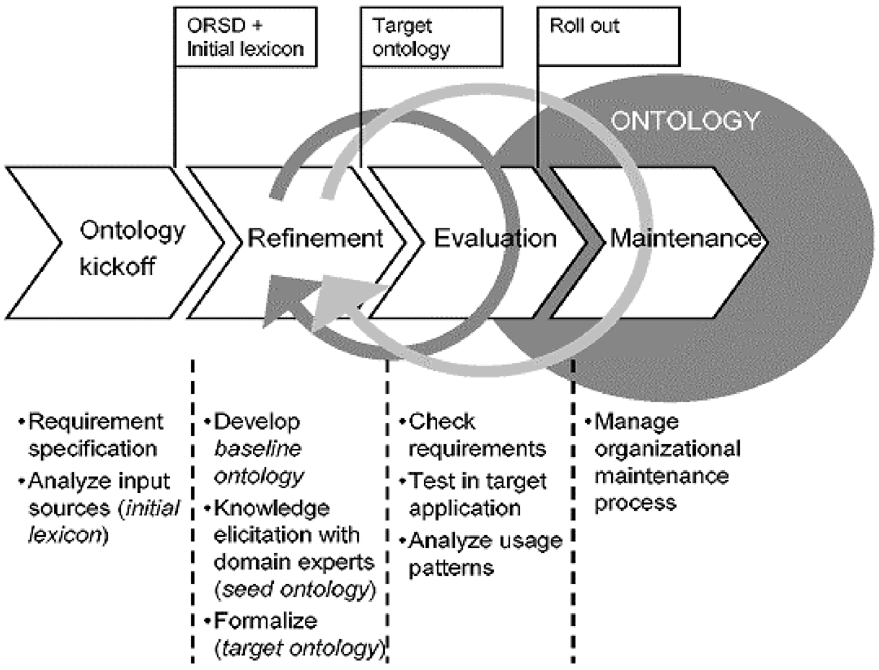

ORSD +
Initial lexicon
Target ontology
Roll out
ONTOLOGY
Ontology kickoff
Refinement
Evaluation
Maintenance
•Requirement specification
•Analyze input sources (*initial lexicon*)
•Develop *baseline ontology*
•Knowledge elicitation with domain experts (*seed ontology*)
•Formalize (*target ontology*)
•Check requirements
•Test in target application
•Analyze usage patterns
•Manage organizational maintenance process

Figure 11.3 Ontology Development.

Kickoff phase for ontology development. The result of the kickoff phase is an ontology requirements specification document (ORSD) describing what an ontology should support, sketching the planned area of the ontology application and listing, *e.g.*, valuable input sources for the gathering of the baseline taxonomy in the refinement phase. Analysis of these input sources delivers an “initial lexicon” containing relevant lexical entries. In general, the ORSD should guide an ontology engineer to decide about inclusion and exclusion of lexical entries, their linkings to concepts / relations and the hierarchical structure of the ontology. In this early stage one should look for already developed and potentially reusable ontologies.

Refinement phase. The goal of the refinement phase is to produce a mature and application-oriented “target ontology” according to the specification given by the kickoff phase. It is divided into several subphases. First, the initial lexicon is linked to corresponding concepts / relations and the concepts are ordered in a taxonomy to form a “baseline ontology”. Second, a knowledge elicitation process with domain experts based on the initial input from the baseline ontology is performed to develop a “seed ontology”. There, the initial baseline ontology is modified and / or extended and axioms are added on top. Third, a formalization phase where the seed ontology is transferred into the target ontology which is expressed in formal representation languages like F-Logic [Kifer, Lausen, & Wu, 1995] , OIL [Decker et al., 2000] or Conceptual Graphs [Sowa, 1992]. During the formalization phase the ontology engineer has to draw the line between ontology and knowledge base (*cf.* Section 2.2).

The usage of potentially reusable ontologies (identified during the kickoff phase) may improve the speed and quality of the development during the whole refinement phase. These ontologies might *e.g.* give useful hints for modeling decisions.

Evaluation phase. The evaluation phase serves as a proof for the usefulness of developed ontologies and their associated software environment. In a first step, the ontology engineer checks, whether the target ontology suffices the ontology requirements specification document. In a second step, the ontology is populated by adding instances to the knowledge base and tested in the target application environment – again, requirements from the ORSD serve as a base for evaluation. Feedback from beta users may be a valuable input for further refinement of the ontology. A valuable input may be the usage patterns of the ontology. The prototype system has to track the ways users navigate or search for concepts and relations. With such a “semantic logfile” (*cf.* Section 6.2) we may trace what areas of the ontology are often “used” and others which were not navigated. This phase is closely linked to the refinement phase and an ontology engineer may need to perform several cycles until the target ontology reaches the envisaged level – the “roll out” of the target ontology finishes the evaluation phase.

Maintenance phase. In the real world things are changing–and so do the specifications for ontologies. To reflect these changes ontologies have to be maintained frequently like other parts of software, too. We stress that the maintenance of ontologies is primarily an organizational process. There must be strict rules for the update-delete-insert processes within ontologies. We recommend that the ontology engineer gathers changes to the ontology and initiates the switch-over to a new version of the ontology after thoroughly testing possible effects to the application, *viz.* performing additional cyclic refinement and evaluation phases. Similar to the refinement phase, feedback from users and analysis of usage patterns may be a valuable input for identifying the changes needed. Maintenance should accompany ontologies as long as they are on duty.

3.2. Experience with ontology development

The methodology describes in general how to develop ontologies. We now describe experiences made while performing the described steps and include some aspects which were not covered by the methodology.

Kickoff phase for ontology development. Setting up requirements for the AIFB ontology we had to deal mainly with modeling the research topics done by different groups of our institute, teaching related topics and last but not least personal information about members of our institute.

We took ourselves as an “input source” and collected a large set of lexical entries for research topics, teaching related topics and personal information. By the sheer nature of these lexical entries, the ontology developers were not able to come up with all relevant lexical entries by themselves. Rather it was necessary to go through several steps with domain experts (*viz.* our colleagues) in the refinement phase.

Refinement phase. We started to develop a baseline taxonomy that contained a heterarchy of research topics identified during the kickoff phase. An important result for us was to recognize that categorization was not based on an isA-taxonomy, but on a much weaker HASSUBTOPIC relationship. We then switched to the second subphase of the refinement phase, *viz.* the development of the seed ontology through knowledge elicitation with our colleagues. There we needed three steps.

In the first step, lexical entries were collected by all members from the institute. Though we had already given the possibility to provide a rough categorization, the categories modeled by nonknowledge engineers were not oriented towards a model of the world, but rather toward the way people worked in their daily routine. Thus, their categorization reflected a particular rather than a shared view onto the domain. A lesson learned from this was that people need an idea about the nature of ontologies to make sound modeling suggestions. It was very helpful to show existing prototypes of ontology-based systems to the domain experts.

In the second step, we worked towards a common understanding of the categorization and the derivation of implicit knowledge, such as “someone who works in logic also works in theoretical computer science ” and inverseness of relations, *e.g.* “an author has a publication” is inverse to “a publication is written by an author”.

In the third step, we mapped the gathered lexical entries to concepts and relations and organized them at a middle level. Naturally, this level involved the introduction of more generic concepts that people would usually not use when characterizing their work (such as “optimization”), but it would also include “politically desired concepts”, because one own’s ontology exhibits one’s view onto the world. Thus, the ontology may become a political issue!

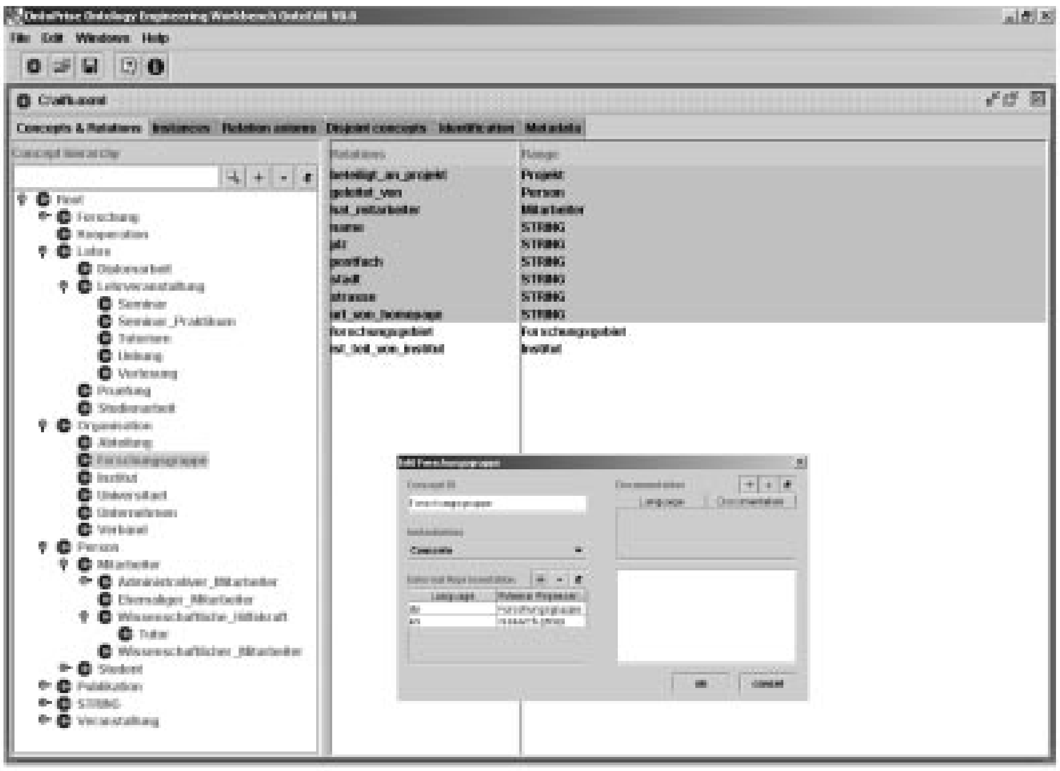
Modeling during early stages of the refinement phase was done with pen and paper, but soon we took advantage of our ontology environment OntoEdit (*cf.* Figure 11.4) that supports graphical ontology engineering at an epistemological level as well as formalization of the ontology. Our underlying inference engine (*cf.* Section 4) is based on F-Logic that we therefore chose as representation language for the target ontology. Like mentioned before, formalization is a non-trivial process where the ontology engineer has to draw the line between ontology and knowledge base. Our final decisions were much disputed. In the conclusion we will mention some of the intricacies that arise from excerpts of our ontology/knowledge base such as the ones given in Section 5.

Evaluation phase. After all we found that participation by users in the construction of the ontology was very good and met the previously defined requirements, as people were very interested to see their work adequately represented. Some people even took the time to learn about OntoEdit. However, the practical problem we had was that our environment does not yet support an ontology management module for cooperative ontology engineering.

We embedded the ontology in it’s version 1.0 into our application and enabled semantic logfiles (*cf.* Section 6.2) to track the usage of the ontology. On top of that we are collecting feedback from our users – basically colleagues and students from our institute. Currently we are still running the ontology version 1.0, but we expect maintenance to be a relevant topic soon.

4. SEAL infrastructure and core modules

The aim of our intranet application is the presentation of information to human and software agents taking advantage of semantic structures. In this section, we first elaborate on the general architecture for SEAL (SEmantic PortAL), before we explain functionalities of its core modules.



Relations	Range
beteiligt_an_projekt	Projekt
geleitet_von	Person
hat_mitarbeiter	Mitarbeiter
name	STRING
plz	STRING
postfach	STRING
stadt	STRING
strasse	STRING
url_von_homepage	STRING
forschungsgebiet	Forschungsgebiet
ist_teil_von_institut	Institut

Figure 11.4 OntoEdit.

4.1. Architecture

The overall architecture and environment of SEAL is depicted in Figure 11.5:

The *backbone* of the system consists of the *knowledge warehouse*, *i.e.* the data repository, and the *Ontobroker* system, *i.e.* the principal inferencing mechanism. The latter functions as a kind of middleware run-time system, possibly mediating between different information sources when the environment becomes more complex than it is now.

At the front end one may distinguish between three types of *agents*: *software agents*, *community users* and *general users*.

All three of them communicate with the system through the *web server*. The three different types of agents correspond to three primary modes of interaction with the system. First, remote applications (*e.g.* software agents) may process information stored at the portal over the internet. For this purpose, the *RDF generator* presents RDF facts through the web server. Software gents with *RDF crawlers* may collect the facts and, thus, have direct access to semantic knowledge stored at the web site.

Second, community users and general users can access information contained at the web site. Two forms of accessing are supported: navigating through the portal by exploiting hyperlink structure of documents and searching for information by posting queries. The hyperlink structure is partially given by the portal builder, but it may be extended with the help of the *navigation* module. The navigation modules exploits inferencing capabilities of the inference engine in order to construct conceptual hyperlink structures. Searching and querying is performed via the *query* module. In addition, the user can personalise the search interface using the *semantic personalization* preprocessing module and/or rank retrieved results according to semantic similarity (done by the postprocessing module for *semantic ranking*). Queries also take advantage of the Ontobroker inferencing.

Third, only community users can provide data. Typical information they contribute includes personal data, information about research areas, publications, activities and other research information.

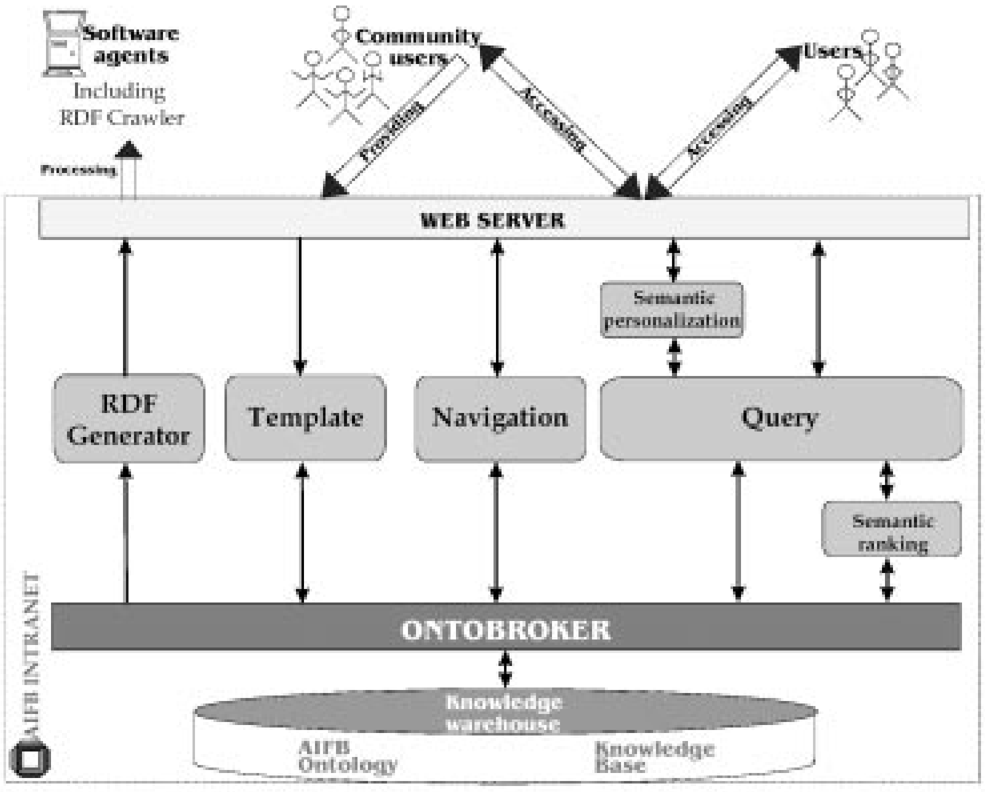


Figure 11.5 AIFB Intranet - System architecture.

For each type of information they contribute there is (at least) one concept in the ontology. Retrieving parts of the ontology, the *template* module may semi-automatically produce suitable HTML forms for data input. The community users fill in these forms and the template modules stores the data in the knowledge warehouse.

4.2. Core modules

The core modules have been extensively described in [Staab et al., 2000]. In order to give the reader a compact overview we here shortly survey their function. In the remainder of the paper we delve deeper into those aspects that have been added or considerably extended recently, *viz.* semantic ranking (Section 5), semantic personalization (Section 6), and semantic access by software agents (Section 7).

Ontobroker. The Ontobroker system [Fensel et al., 1998] is a deductive, object-oriented database system operating either in main memory or on a relational database (via JDBC). It provides compilers for different languages to describe ontologies, rules and facts. Beside other usage, in this architecture it is also used as an inference engine (server). It reads input files containing the knowledge base and the ontology, evaluates incoming queries, and returns the results derived from the combination of ontology, knowledge base and query.

The possibility to derive additional factual knowledge from given facts and background knowledge considerably facilitates the life of the knowledge providers and the knowledge seekers. For instance, one may specify that if a person belongs to a research group of institute

AIFB, he also belongs to AIFB. Thus, it is unnecessary to specify the membership to his research group *and* to AIFB. Conversely, the information seeker does not have to take care of inconsistent assignments, *e.g.* ones that specify membership to an AIFB research group, but that have erroneously left out the membership to AIFB.

Knowledge warehouse. The knowledge warehouse [Staab et al., 2000] serves as repository for data represented in the form of F-Logic statements. It hosts the ontology, as well as the data proper. From the point of view of inferencing (Ontobroker) the difference is negligible, but from the point of view of maintaining the system the difference between ontology definition and its instantiation is useful. The knowledge warehouse is organised around a relational database, where facts and concepts are stored in a reified format. It states relations and concepts as first-order objects and it is therefore very flexible with regard to changes and amendments of the ontology.

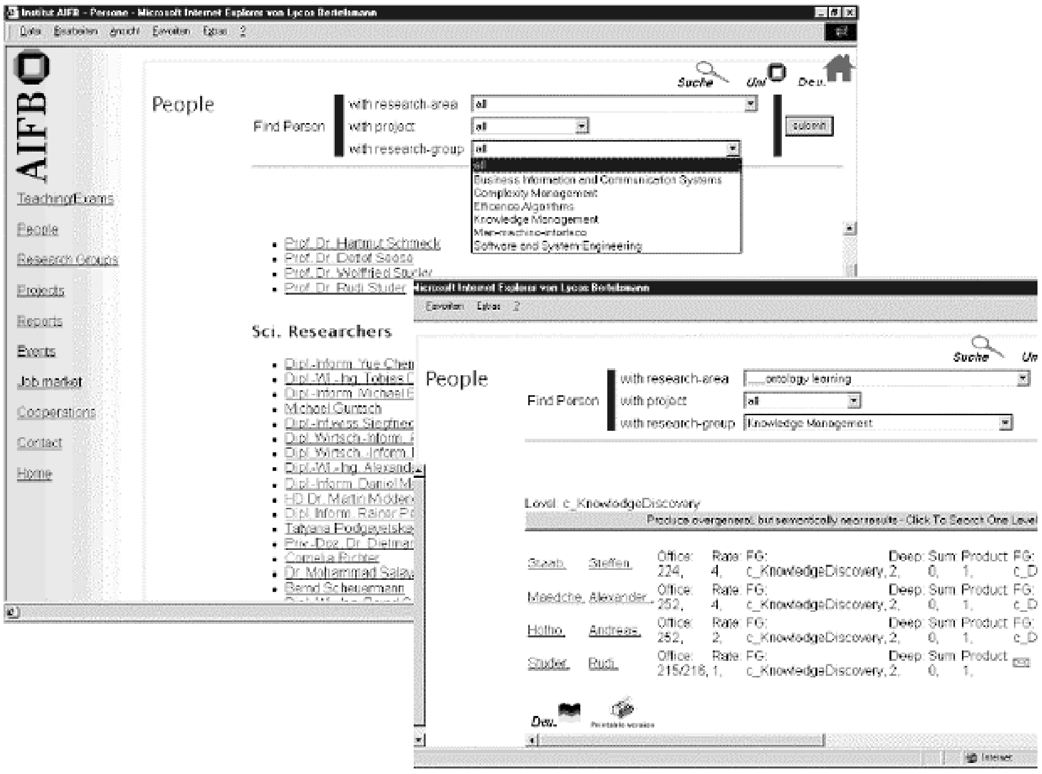
Navigation module. Beside the hierarchical, tree-based hyperlink structure which corresponds to hierarchical decomposition of domain, the navigation module enables complex graph-based semantic hyperlinking, based on ontological relations between concepts (nodes) in the domain. The conceptual approach to hyperlinking is based on the assumption that semantic relevant hyperlinks from a web page correspond to conceptual relations, such as memberOf or hasPart, or to attributes, like has Name. Thus, instances in the knowledge base may be presented by automatically generating links to all related instances. For example, on personal web pages (*cf.* Figure 11.7) there are hyperlinks to web pages that describe the corresponding research groups, research areas and project web pages.

Query module. The query module puts an easy-to-use interface on the query capabilities of the F-Logic query interface of Ontobroker. The portal builder models web pages that serve particular query needs, such as querying for projects or querying for people. For this purpose, selection lists that restrict query possibilities are offered to the user. The selection lists are compiled using knowledge from the ontology and/or the knowledge base. For instance, the query interface for persons allows to search for people according to research groups they are members of. The list of research groups is dynamically filled by an F-Logic query and presented to the user for easy choice by a drop-down list (*cf.* snapshot in Figure 11.6).

Even simpler, one may apprehend a hyperlink with an F-Logic query that is dynamically evaluated when the link is hit. More complex, one may construct an is A, a has Part, or a has Subtopic tree, from which query events are triggered when particular nodes in the tree are navigated.

Template module. In order to facilitate the contribution of information by community users, the template module generates an HTML form for each concept that a user may instantiate. For instance, in the AIFB intranet there is an input template (*cf.* Figure 11.7, upper left) generated from the concept definition of person (*cf.* Figure 11.7, lower left). The data is later on used by the navigation module to produce the corresponding person web page (*cf.* Figure 11.7, right hand side).

In order to reduce the data required for input, the portal builder specifies which attributes and relations are derived from other templates. For example, in our case the portal builder has specified that project membership is defined in the project template. The co-ordinator of a project enters information about which persons are participants of the project and this information is used when generating the person web page taking advantage of a corresponding F-Logic rule for inverse relationships. Hence, it is unnecessary to input this information in the person template.



People

Find Person
- with research-area: all
- with project: all
- with research-group: all

all
Business Information and Communication Systems
Complexity Management
Efficient Algorithms
Knowledge Management
Man-machine-interface
Software and System Engineering

- Prof. Dr. Hartmut Schmeck
- Prof. Dr. Detlef Seese
- Prof. Dr. Wolffried Stucky
- Prof. Dr. Rudi Studer

Sci. Researchers

- Dipl.-Inform. Yue Chen
- Dipl.-Wi.-Ing. Tobias
- Dipl.-Inform. Michael
- Michael Gurtsch

People

Find Person
- with research-area: ontology learning
- with project: all
- with research-group: Knowledge Management

Level: c_KnowledgeDiscovery

Produce overgeneral, but semantically near results - Click To Search One Level

Staab,	Steffen,	Office: 224,	Rate: 4,	FG: c_KnowledgeDiscovery,	Deep: 2,	Sum: 0,	Product: 1,	FG: c_D
Maedche,	Alexander,	Office: 252,	Rate: 4,	FG: c_KnowledgeDiscovery,	Deep: 2,	Sum: 0,	Product: 1,	FG: c_D
Hotho,	Andreas,	Office: 252,	Rate: 2,	FG: c_KnowledgeDiscovery,	Deep: 2,	Sum: 0,	Product: 1,	FG: c_D
Studer,	Rudi,	Office: 215/216,	Rate: 1,	FG: c_KnowledgeDiscovery,	Deep: 2,	Sum: 0,	Product: 1,	

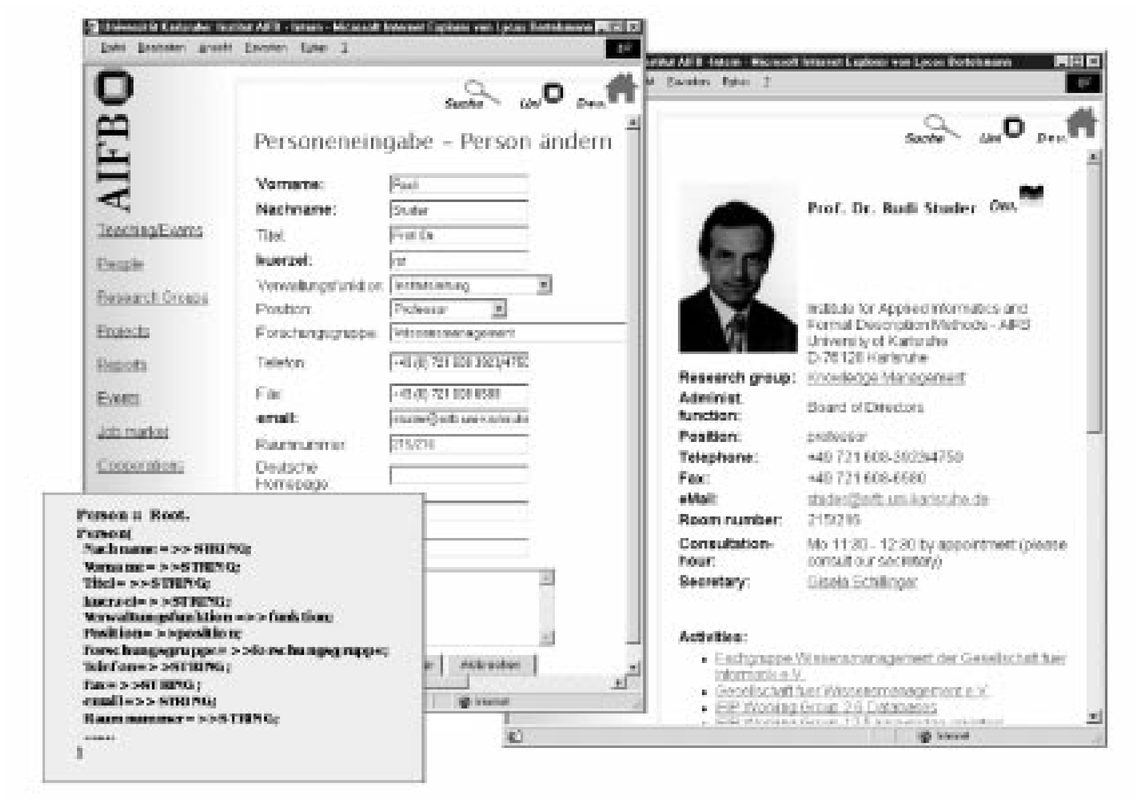
Figure 11.6 Searching form based on definition of concept Person.

Ontology lexicon. The different modules described here make extensive use of the lexicon component of the ontology. The most prevalent use is the distinction between English and German (realized for presentation, though not for the template module, yet). In the future we envision that one may produce more adaptive web sites making use of the explicit lexicon. For instance, we will be able to produce short descriptions when the context is sufficiently narrow, *e.g.* working with ambiguous acronyms like ASP [5] or SEAL [6].

5. Semantic ranking

This section describes the architecture component “Semantic ranking” which has been developed in the context of our application. First, we will introduce and motivate the requirement for a ranking approach with a small example we are facing. Second, we will show how the problem of semantic ranking may be reduced to the comparison of two knowledge bases. Query results are reinterpreted as “query knowledge bases” and their similarity to the original knowledge base without axioms yields the basis for semantic ranking. Thereby, we reduce our notion of similarity between two knowledge bases to the similarity of concept pairs [Maedche & Staab, 2000].

Let us assume the following ontology:



Personeneingabe – Person ändern

Vorname:	Rudi
Nachname:	Studer
Titel:	Prof. Dr.
kuerzel:	rst
Verwaltungsfunktion:	Institutsleitung
Position:	Professor
Forschungsgruppe:	Wissensmanagement
Telefon:	+49 (0) 721 608 3923/4750
Fax:	+49 (0) 721 608 6580
email:	studer@aifb.uni-karlsruhe.de
Raumnummer:	215/216
Deutsche Homepage	

```
Person :: Root.
Person[
  Nachname =>> STRING;
  Vorname =>> STRING;
  Titel =>> STRING;
  kuerzel =>> STRING;
  Verwaltungsfunktion =>> funktion;
  Position =>> position;
  Forschungsgruppe =>> forschungsgruppe;
  Telefon =>> STRING;
  Fax =>> STRING;
  email =>> STRING;
  Raumnummer =>> STRING;
  .....
]
```

Prof. Dr. Rudi Studer

Institute for Applied Informatics and Formal Description Methods - AIFB
University of Karlsruhe
D-76128 Karlsruhe

Research group:	Knowledge Management
Administ. function:	Board of Directors
Position:	professor
Telephone:	+49 721 608-3923/4750
Fax:	+49 721 608-6580
eMail:	studer@aifb.uni-karlsruhe.de
Room number:	215/216
Consultation-hour:	Mo 11:30 - 12:30 by appointment (please consult our secretary)
Secretary:	Gisela Schillinger

Activities:

- Fachgruppe Wissensmanagement der Gesellschaft fuer Informatik e.V.
- Gesellschaft fuer Wissensmanagement e.V.
- FIP Working Group 2.6 Databases

Figure 11.7 Templates generated from concept definitions

1: Person :: Object [worksIn ⇒ Project].

2: Project :: Object [worksIn ⇒ Topic].

3: Person :: Object [worksIn ⇒ Topic].

To give an intuition of the semantic of the F-Logic statements, in line 1 one finds a concept definition for a Person being an Object with a relation *worksIn*. The range of the relation for this Person is restricted to Project.

Let us further assume the following knowledge base:

4 : KnowledgeManagement: Topic.

5 : KnowledgeDiscovery: Topic [subtopicOf $\rightarrow$ KnowledgeManagement].

6 : Gerd : Person [worksIn $\rightarrow$ OntoWise].

7 : OntoWise : Project[hasTopic $\rightarrow$ KnowledgeManagement].

8 : Andreas : Person[worksIn $\rightarrow$ TelekomProject].

9 : TelekomProject : Project[hasTopic $\rightarrow$ KnowledgeDiscovery].

10: FORALL X, Y, Z Z[hasTopic $\rightarrow$ Y] $\leftarrow$ X[subtopicOf $\rightarrow$ Y] and [hasTopic $\rightarrow$ X].

Definitions of instances in the knowledge base are syntactically very similar to the concept definition in F-logic. In line 5 the instance KnowledgeDiscovery of the concept *Topic* is defined. Furthermore, the relation *subtopicOf* is instantiated between KnowledgeDiscovery and

KnowledgeManagement. Similarly in line 6, it is stated that Gerd is a Person working in OntoWise.

Now, an F-Logic query may ask for all people who work in a knowledge management project by:

$$\text{FORALL } Y, Z \leftarrow Y[\text{worksIn} \twoheadrightarrow Z] \text{ and } Z : \mathit{Project}[\text{hasTopic} \twoheadrightarrow \text{KnowledgeManagement}]$$

which may result in the tuples M_1^T : = (Gerd, OntoWise) and M_2^T : = (Andreas, TelekomProject). Obviously, both answers are correct with regard to the given knowledge base and ontology, but the question is, what would be a plausible ranking for the correct answers. This ranking should be produced from a given query without assuming any modification of the query.

5.1. Reinterpreting queries

Our principal consideration builds on the definition of semantic similarity that we have first described in [Maedche & Staab, 2000]. There, we have developed a measure for the similarity of two knowledge bases. Here, our basic idea is to reinterprete possible query results as a “query knowledge base” and compute its similarity to the original knowledge base while abstracting from semantic inferences. The result of an F-Logic query may be re-interpreted as a *query knowledge base* (*QKB*) by the following approach. An F-Logic query is of the form or can be rewritten into the form [7]:

$$\text{FORALL } \overline{X} \leftarrow \overline{P}(\overline{X}, \overline{k}), \qquad (4)$$

With *X* being a vector of variables *(X1,…,X n), k* being a vector of constants, and *P* being a vector of conjoined predicates. The result of a query is a two-dimensional matrix M of size *m* x *n,* with n being the number of result tuples and m being the length X of and, hence, the length of the result tuples. Hence, in our example above *X:= (Y, Z), k:=* (‘‘knowledge management’’), *P:=(P1, P2), P1(a, b, c)* : = a [worksIn → b], *P:=(P1, P2), P2(a, b, c)* : = b[hasTopic → c] and

$$M := (M_1, M_2) = \begin{pmatrix} \texttt{Gerd} & \texttt{Andreas} \\ \texttt{OntoWise} & \texttt{TelekomProjekt} \end{pmatrix}. \qquad (5)$$

Now, we may define the query knowledge base *i (QKBi)* by

$$QKB_i := \overline{P}(M_i, \overline{k}). \qquad (6)$$

The similarity measure between the query knowledge base and the given knowledge base may then be computed in analogy to [Maedche & Staab, 2000]. An adaptation and simplification of the measures described there is given in the following together with an example.

5.2. Similarity of knowledge bases

The similarity between two objects (concepts and or instances) may be computed by considering their relative place in a common hierarchy *H*. *H* may, but need not be a taxonomy H. For instance, in our example from above we have a categorization of research topics, which is not a taxonomy! Our principal measures are based on the cotopies of the correspondin objects as defined by a given hierarchy *H*, *e.g.* an ISA hierarchy H , an part-whole hierarchy, or a categorization of topics. Here, we use the *upwards cotopy* (UC) defined as follows:

$$\mathrm{UC}(O_i, H) := \{O_j | H(O_i, O_j) \vee O_j = O_i\} \qquad (7)$$

UC is overloaded in order to allow for a set of object *M* as input instead of only single objects, *viz.*

$$\mathrm{UC}(M,H) := \bigcup_{O_i \in M} \{O_j | H(O_i, O_j) \vee O_j = O_i\} \tag{8}$$

Based on the definition of the upwards cotopy (UC) the object match (OM) is defined by:

$$\mathrm{OM}(O_1, O_2, H) := \frac{|\mathrm{UC}(O_1, H) \cap \mathrm{UC}(O_2, H)|}{|\mathrm{UC}(O_1, H) \cup \mathrm{UC}(O_2, H)|}. \tag{9}$$

Basically, OM reaches 1 when two concepts coincide (number of intersections of the respective upwards cotopies and number of unions of the respective cotopies is equal); it degrades to the extent to which the discrepancy between intersections and unions increases (a OM between concepts that do not share common superconcepts yields value 0).

Example. We here give a small example for computing UC and OM based on a given categorization of objects H. Figure 11.8 depicts the example scenario.

The upwards cotopy UC (knowledge discovery, *H*) is given by UC (knowledge discovery, *H*) = {knowledge discovery; knowledge management}. The upwards cotopy UC(optimization);*H*) is computed by UC(optimization); *H*)) = {optimization}.

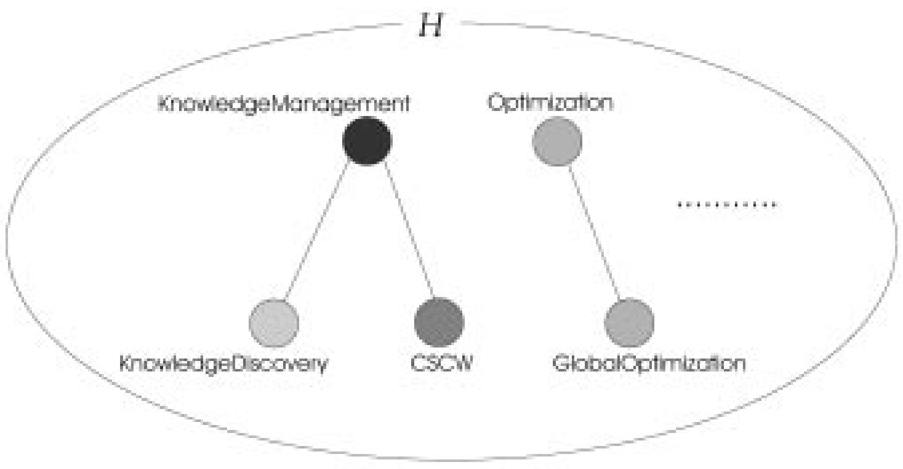


Figure 11.8 Example for computing UC and OM

Computing the object match OM between KnowledgeManagement and Optimization results in 0, the object match between KnowledgeDiscovery and CSCW computes to 1/3.

The match introduced above may easily be generalized to relations using a relation hierarchy *Hr*. Thus, the predicate match (PM) for two n-ary predicate *P1, P2* is defined by a mean value. Thereby, we use the geometric mean in order to reflect the intuition that if the similarity of one of the components approaches the overall similarity between two predicates should approach — which need not be the case for the arithmetic mean:

$$\mathrm{PM}(P_1(I_1,\ldots,I_n),P_2(J_1,\ldots,J_n)) := \sqrt[n+1]{\mathrm{OM}(P_1,P_2,\mathcal{H}_R)\cdot\mathrm{OM}(I_1,J_1,H)\cdot\ldots\cdot\mathrm{OM}(I_n,J_n,H)}. \qquad (10)$$

This result may be averaged over an array of predicates. We here simply give the formula for our actual needs, where a query knowledge base is compared against a given knowledge base KB:

$$Simil(QKB_i, KB) = Simil(\overline{P}(M_i,\overline{k}), KB) := \frac{1}{|\overline{P}|} \sum_{P_j \in \overline{P}} \max_{Q(M_i,\overline{k}) \in KB.S} \mathrm{PM}(P_j(M_i,\overline{k}), Q(M_i,\overline{k})). \quad (11)$$

For instance, comparing the two result tuples from our example above with the given knowledge base:

```
Gerd[WORKSIN ->> OntoWise].
OntoWise[HASTOPIC ->> KnowledgeManagement].                (12)
```

First, M1^T := (Gerd OntoWise). Then, we have the query knowledge base (*QKB1*):

and its relevant counterpart predicates in the given knowledge base (*KB*) are:

```
Gerd[WORKSIN ->> OntoWise].
OntoWise[HASTOPIC ->> KnowledgeManagement].                (13)
```

T

his is a perfect fit. Therefore *Simil(QKB1, KB)* computes to 1.

Second, M_2^T := (Andreas TelekomProject.) Then, we have the query knowledge base *(QKB2):*

```
Andreas[WORKSIN ->> TelekomProject].
TelekomProject[HASTOPIC ->> KnowledgeManagement].          (14)
```

and its relevant counterpart predicates in the given knowledge base *(KB)* are:

```
Andreas[WORKSIN ->> TelekomProject].
TelekomProject[HASTOPIC ->> KnowledgeDiscovery].           (15)
```

Hence, the similarity of the first predicates indicates a perfect fit and evaluates to 1, but the congruency of TelekomProject[hasTopic $\rightarrow$ KnowledgeManagement] with TelekomProject[hasTopic $\rightarrow$ KnowledgeManagement] measures less than 1. The instance match of KnowledgeDiscovery and

KnowledgeManagement returns 1/2 in the given topic hierarchy. Therefore, the predicate match returns (1* 1* 1/2)^1/3 ≈ 0, 79. Thus, overall ranking of the second result is based on 1/2 *(1+0,79) = 0,895

Remarks on semantic ranking. The reader may note some basic properties of the ranking: *(i)* similarity of knowledge bases is an asymmetric measure, *(ii)* the ontology defines a conceptual structure useful for defining similarity, *(iii)* the core concept for evaluating semantic similarity is cotopy defined by a dedicated hierarchy. The actual computation of similarity depends on which conceptual structures (*e.g.* hierarchies like taxonomy, part-whole hierarchies, or topic hierarchies) are selected for evaluating conceptual nearness. Thus, similarity of knowledge bases depends on the view selected for the similarity measure.

Ranking of semantic queries using underlying ontological structures is an important means in Order to allow users a more specific view onto the underlying knowledge base. The method that we propose is based on a few basic principles:

- Reinterprete the combination of query and results as query knowledge bases that may be compared with the explicitly given information.
- Give a measure for comparing two knowledge bases, thus allowing rankings of query results.

Thus, we may improve the interface to the underlying structures without changing the basic architecture. Of course, the reader should be aware that our measure may produce some rankings for results that are hardly comparable. For instance, results may differ slightly because of imbalances in a given hierarchy or due to rather random differences of depth of branches. In this case, ranking may perhaps produce results that are not better than unranked ones –but the results will not be any worse either.

6. Semantic personalization

Personalization of web sites might be enabled on different levels like *e.g.* so-called ”Check-Box Personalization“ and ”Preference-based Personalization“ (*cf.* [FAQ]). Both rely on the interaction between users and the underlying systems and aim at providing person-specific services and content to users. But they are differing the way systems are tracking a user’s interests to personalize web sites.

Check-box personalization offers configuration possibilities to users, allowing them to select from a given set of services and contents the ones they are looking for. Content is presented based on the check boxes marked by users. Preference-based personalization keeps track of users access patterns in logfiles. *E.g.* web usage mining tries to make sense of the web usage data created by surfers [Srivastava et al., 2000].

Considering check-box personalization, common systems rely on the selection of services and specific content areas and for preference-based personalization on monitoring of user-activities based on the sequence of clicked URLs. Our portal differs from common systems in offering semantic access. All users who access the web site use the underlying ontology for navigating, browsing and querying. To address check-box personalization users may store personalized *semantic bookmarks*. They rely on the semantic structure provided by the underlying ontology and like common bookmarks they facilitate access to regulary needed information.

While accessing the semantic web site, users leave footprints in form of clicked hyperlinks, but they also leave a trace of concepts and relations. Every query is composed from the ontology and (almost) every navigation follows underlying ontological structures. To enable preference-based personalization, we upgrade our logfiles with semantics to *semantic logfiles*. The user habits may then be semantically tracked, analyzed and used for personalization, general optimization of the web site and especially evaluation and maintenance of the ontology .

6.1. Semantic bookmarks

The AIFB mainly targets students and researchers. Both groups typically have different foci. Students tend to look for teaching-related topics and researchers tend to look for research-related topics. To satisfy their needs, we provide links to both areas from the starting page of our web site. But users sometimes need to have persistent pointers to specific information. Navigating and browsing through our web site as well as posting queries relies on the underlying ontology, therefore we have introduced *semantic bookmarks*.

Because many navigation facilities of our web site like *e.g.* hyperlinks and listboxes are created on the fly by our inference engine, semantic bookmarks basically contain predefined[8] query formulas. Upon selection bookmarks send their queries to the inference engine that reproduces the information shown to the user during his bookmarking process. Due to the fact that bookmarks contain query formulas they always produce most recent *i.e.* updated results. Semantic bookmarks are modeled in the ontology (*cf.* (16) *Bookmark*), their instantiations are stored in the knowledge warehouse (*cf.* (16) *Bookmark 1*):

```
Bookmark[                              Bookmark1:Bookmark[
QUERY ⇒⇒ STRING;                       QUERY →→ "FORALL X,N ← X:Person[lastName →→ N].";
NAME ⇒⇒ STRING;                        NAME →→ "List of all persons";
STYLESHEET ⇒⇒ PersonStylesheet;        STYLESHEET →→ PersonStylesheet1;
START ⇒⇒ BOOLEAN;                      START →→ FALSE;
OWNER ⇒⇒ User].                        OWNER →→ User1].
                                                                                  (16)
```

In addition to the *QUERY* formula, bookmarks contain several options for personalization. First, users may give a specific NAME for their bookmarks to describe the functionality. Second, they may choose a STYLESHEET from a given set of stylesheets for result presentation. Finally, semantic bookmarks might be marked as a START bookmark, so it will be executed as soon as users enter the starting page of our web site. This allows users to have quickest possible access to often needed informations. Every user may execute, edit and delete his own semantic bookmarks – identified by the relation OWNER.

In order to deliver personalized information the system needs to recognize users. We implemented a very simple ontology-based user administration allowing users to be recognized by their user-id. Therefore our ontology was expanded by user-specific concepts and relations (like User, ID, NAME etc.). Our system generates for each user an identifying id which is stored together with all relevant user information in our knowledge warehouse. For convenience, user-ids are also stored locally in cookies so that users are recognized automatically by the id stored on their computer. Possible pitfalls of such a simple approach are *e.g.* multi-user access from single computers or single-user access from multiple computers.

6.2. Semantic logfiles

Agents interact with our system through a web server (*cf.* Figure 11.5) who logs every request (*viz.* http-requests) into logfiles. Queries are processed to our inference engine through embedding query formulas into hyperlinks. Therefore all semantics are contained within logged hyperlinks.

Typically web servers track the following items: *Cookie*, *IP number*, *Timestamp*, *Request*. We take advantage of the information given. Cookies help to identify authenticated users and IP numbers in combination with timestamps help to distinguish non-authenticated users. Finally, the request string contains all concepts and relation of the ontology a user is interested in — due to the nature[9] of our ontology-enabled querying, navigation and browsing.

Currently we are working on transforming these logfiles into appropriate formats and to apply data mining methods and tools (like association rules). We are interested in answering, *e.g.*, the following questions:

- Does a single authenticated user have a special interest in a certain part of the ontology?
- Are there user groups, having similar interests?
- Which parts of the ontology are relevant to users, which ones are not?

Answering these questions will help us to optimize our system in iterative steps, *e.g.*, to enable ranked index lists providing each user with personalized shortcuts. Especially the last question is important for evaluation and maintenance of the ontology (*cf.* Section 3). To keep an ontology updated one might want to expand highly accessed parts, shrink rarely accessed parts and finally delete unaccessed parts. These topics are still ongoing research.

7. RDF outside – from a semantic web site to the Semantic Web

In the preceding sections we have described the development and the underlying techniques of the AIFB semantic web site. Having developed the core application we decided that RDF-capable software agents should be able to understand the content of application. Therefore, we have built an automatic RDF GENERATOR that dynamically generates RDF statements on each of the static and dynamic pages of the semantic knowledge portal. Our current AIFB intranet application is "Semantic Web-ized" using RDF facts instantiated and defined according to the underlying AIFB ontology. On top of this generated and formally represented metadata, there is the RDF CRAWLER, a tool that gathers interconnected fragments of RDF from the internet.

7.1. RDF GENERATOR – an example

The RDFMAKER established in the ONTOBROKER framework (*cf.* [Decker et al., 1999]) was a starting point for building the RDF GENERATOR. The idea of RDFMAKER was, that from ONTOBROKER'S internal data base, RDF statements are generated.

RDF GENERATOR follows a similar approach and extends the principal ideas. In a first step it generates an RDF(S)-based ontology that is stored on a specific XML namespace, *e.g.* in our concrete application http://ontobroker.semanticweb.org/ontologies/aifb-onto-2001-01-01.rdfs. Additionally, it queries the knowledge warehouse. Data, *e.g.* for a person, is checked for consistency, and, if possible, completed by applying the given F-Logic rules. We here give a short example of what type of data may be generated and stored on a specific homepage of a researcher:

```
<rdf:RDF
xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:aifb = "http://ontobroker.semanticweb.org/aifb-2001-01-01.rdfs#">
<aifb:PhDStudent rdf:ID="person:ama">
<aifb:name>Alexander Maedche</aifb:name>
<aifb:email>ama@aifb.uni-karlsruhe.de</aifb:email>
<aifb:phone>+49-(0)721-608 6558</aifb:phone>
<aifb:fax>+49-(0)721-608 6580</aifb:fax>
<aifb:homepage>http://www.aifb.uni-karlsruhe.de/WBS/ama</aifb:homepage>
<aifb:supervisor
rdf:resource = "http://www.aifb.uni-karlsruhe.de/Staff/studer.html#person:rst"/>
</aifb:PhDStudent>
</rdf:RDF>
```

RDF GENERATOR is a configurable tool, in some cases one may want to use inferences to generate materialized, complete RDF descriptions on a home page, in other cases one may want to generate only ground facts of RDF. Therefore, RDF GENERATOR allows to switch axioms on and off in order to adopt the generation of results to varying needs.

7.2. RDF CRAWLER

The RDF CRAWLER [10] is a tool which downloads interconnected fragments of RDF from the internet and builds a knowledge base from this data. Building an external knowledge base for the whole AIFB (its researcher, its projects, its publications, . . .) becomes easy using the RDF CRAWLER and machine-processable RDF data currently defined on AIFB's web. We here shortly describe the underlying techniques of our RDF CRAWLER and the process of building a knowledge base. In general, RDF data may appear in Web documents in several ways. We distinguish between pure RDF (files that have an extension like "*.rdf"), RDF embedded in HTML and RDF embedded in XML. Our RDF CRAWLER uses RDF-API[11] that can deal with different embeddings of RDF described above.

One problem of crawling is the applied filtering mechanism: Baseline crawlers are typically restricted by a given depth value. Recently several new research work on so-called *focused crawling* has been published (*e.g. cf.* [Chakrabarti, van den Berg, & Dom, 1999]). In their approach, they use a set of predefined documents associated with topics in a Yahoo like taxonomy to built a focused crawler. Two hypertext mining algorithms constitute the core of their approach. A classifier evaluates the relevance of a hypertext document with respect to the focus topics and a distiller identifies hypertext nodes that are good access points to many relevant pages within a few links. In contrast, our approach uses ontological background knowledge to judge the relevance of each page. If a page is highly relevant, the crawler may follow the links on the particular web site. If RDF data is available on a page, we judge relevance with respect to the quantity and quality of available data and by the existing URI's.

Example: Erdoes numbers. As mentioned above we here give a small example of a nice application that may be easily built using RDF metadata taken from AIFB using the RDF CRAWLER. The so-called *Erdoes numbers* have been a part of the folklore of mathematicians throughout the world for many years[12].

Scientific papers are frequently published with co-authors. Based on information about collaboration one may compute the Erdoes number (denoted PE *(R)*) for a researcher *R.* In the AIFB web site the RDF-based metadata allows for computing estimates of Paul Erdoes numbers of AIFB members.

The numbers are defined recursively:

1. *PE(R)* =0, if R is Paul Erdoes
2. *PE(R) = min{PE(R1) + 1}* else, where *R1* varies over the set of all researchers who have collaborated with R, *i.e.* have written a scientific paper together.

To put this into work, we need lists of publications annotated with RDF facts. The lists may be automatically generated by the RDF Generator. Based on the RDF facts one may crawl relevant information into a central knowledge base and compute these numbers from the data.

8. Related work

This section positions our work in the context of existing web portals and also relates our work to other basic methods and tools that are or could be deployed for the construction of community web portals, especially to related work in the areas of personalization and semantic ranking of query results.

Related work on Knowledge Portals. One of the well-established web portals on the web is Yahoo[13]. In contrast to our approach Yahoo only utilizes a very light-weight ontology that solely consists of categories arranged in a hierarchical manner. Yahoo offers keyword search (local to a selected topic or global) in addition to hierarchical navigation, but is only able to retrieve complete documents, *i.e.* it is not able to answer queries concerning the contents of documents, not to mention to combine facts being found in different documents or to include facts that could be derived through ontological axioms. Personalization is limited to check-box personalization. We get rid of these shortcomings since our portal is built upon a rich ontology enabling the portal to give integrated answers to queries. Furthermore, our semantic personalization features provide more flexible means for adapting the portal to the specific needs of its users.

A portal that is specialized for a scientific community has been built by the Math-Net project [Dalitz, Gr¨otschel, & Lügger, 1997]. At http://www.math-net.de/ the portal for the (German) mathematics community is installed that makes distributed information from several mathematical departments available. This information is accompanied by meta-data according to the Dublin Core[14] Standard [Weibel et al., 1998]. The Dublin Core element “Subject” is used to classify resources as conferences, as research groups, as preprints etc. A finer classification (*e.g.* via attributes) is not possible except for instances of the publication category. Here the common MSC-Classification[15] is used that resembles a light-weight ontology of the field of mathematics. With respect to our approach Math-Net lacks a rich ontology that could enhance the quality of search results (esp. via inferencing), and the smooth connection to the Semantic Web world that is provided by our RDF generator.

The Ontobroker project [Decker et al., 1999] lays the technological foundations for the AIFB portal. On top of Ontobroker the portal has been built and organizational structures for developing and maintaining it have been established. Therefore, we compare our system against approaches that are similar to Ontobroker.

The approach closest to Ontobroker is SHOE [Heflin & Hendler, 2000]. In SHOE, HTML pages are annotated via ontologies to support information retrieval based on semantic information. Besides the use of ontologies and the annotation of web pages the underlying philosophy of both systems differs significantly: SHOE uses description logic as its basic representation formalism, but it offers only very limited inferencing capabilities. Ontobroker relies on Frame-Logic and supports complex inferencing for query answering. Furthermore, the SHOE search tool neither provides means for a semantic ranking of query results nor for a semantic personalization feature. A more detailed comparison to other portal approaches and underlying methods may be found in [Staab et al., 2000].

Related work on Personalization. Personalization is a feature of portals that attracts more and more interest, both from a research as well as from a commercial point of view. In contrast to our semantic personalization approach that exploits the conceptualization as offered by the ontology and the inferencing capabilities as provided by our ontobroker component, preference-based personalization approaches typically rely on some notion of web mining exploiting the content of hypertext documents (web content mining), analyzing the hypertext links structure (web structure mining) or server/browser logs (web usage mining) [Kosala & Blockeel, 2000]. *I.e.*, the organization and presentation of the web site might be optimized over time based on the analysis of such logfiles. While users are accessing the web sites, their habits are compared to previous users’ behaviors to offer them personalized content like *e.g.* ranked index-lists (*cf.* [Perkowitz & Etzioni, 1999]).

From a marketing point of view the analysis of customer behaviour might be used to relate the interaction of customers with the web site to aspects that are important for customer relationship management. *E.g.*, Easyminer [Anand et al., 2000] is a web intelligence tool that allows to analyze the customer behaviour with respect to aspects like the clicks-to-close value, *i.e.* how easily customers can find the information they are interested in and how this value can be improved through personalization. However, in contrast to our semantic personalization approach the data mining algorithms used in Easyminer are analyzing the syntactical link structures.

Related work on Semantic Similarity. Since our semantic ranking is based on the comparison of the query knowledge base with the given ontology and knowledge base, we relate our work to the comparison of ontological structures and knowledge bases (covering the same domain) and to measuring the similarity between concepts in a hierarchy. Although there has been a long discussion in the literature about evaluating knowledge-bases [Menzis, 1998], we have not found any discussion about comparing two knowledge bases covering the same domain that corresponds to our semantic ranking approach. Similarity measures for ontological structures have been investigated in areas like cognitive science, databases or knowledge engineering (*cf. e.g.*, [Rada et al., 1989], [Resnik, 1995] , [Richardson, Smeaton, & Murphy, 1994], [Hovy, 1998]). However, all these approaches are restricted to similarity measures between lexical entries, concepts, and template slots within one ontology.

Closest to our measure of similarity is work in the NLP community, named semantic similarity [Resnik, 1995] which refers to similarity between two concepts in a isA-taxonomy such as the WordNet or CYC upper ontology. Our approach differs in two main aspect from this notion of similarity: Firstly, our similarity measure is applicable to a hierarchy which may, but not need be a taxonomy and secondly it is taking into account not only commonalties but also differences between the items being compared, expressing both in semantic-cotopy terms. This second property enables the measuring of self-similarity and subclass-relationship similarity, which are crucial for comparing results derived from the inferencing processes, that are executed in the background.

Conceptually, instead of measuring similarity between isolated terms (words), that does not take into account the relationship among word senses that matters, we measure similarity between “words in context”, by measuring similarity between Object-Attribute-Value pairs, where each term corresponds to a concept in the ontology. This enables us to exploit the ontological background knowledge (axioms and relations between concepts) in measuring the similarity, which expands our approach to a methodology for comparing knowledge bases.

From our point of view, our community portal system is rather unique with respect to the collection of methods used and the functionality provided. We have extended our community portal approach that provides flexible means for providing, integrating and accessing information [Staab et al., 2000] by semantic personalization features, semantic ranking of generated answers and a smooth integration with the evolving Semantic Web. All these methods are integrated into one uniform system environment, the SEAL framework.

Conclusions

In this paper we have shown our comprehensive approach SEAL for building semantic portals. In particular, we have focused on three issues.

First, we have considered the ontological foundation of SEAL and the methodological aspects of building ontologies for knowledge systems. There, we have made the experience that there are many big open issues that have hardly been dealt with so far. In particular, the step of formalizing the ontology raises very principal problems. The issue of where to put relevant concepts, *viz.* Into the ontology *vs.* into the knowledge base, is an important one that deeply affects organization and application. However, there exist no corresponding methodological guidelines to base the decision upon so far. For instance, we have given the example ontology and knowledge base in (1) and (2). Using description logics terminology, we have equated the ontology with the “T-Box” and we have put the topic hierachy into the knowledge base (“A-Box”). An alternative could have been to formalize the topic hierarchy as an isA-hierarchy, which however it isn’t and put it into the T-Box. We believe that both alternatives exhibit an internal fault, *viz.* the ontology should not be equated with the T-Box, but rather should its scope be independent from an actual formalization with particular logical statements. Its scope should to a large extent depend on soft issues, like “Who updates a concept?” and “How often does a concept change?” such as already indicated in Table 1.

Second, we have described the general architecture of the SEAL approach, which is also used for our real-world case study, the AIFB web site. The architecture integrates a number of components that we have also used in other applications, like Ontobroker, navigation or query module.

Third, we have extended our semantic modules to include a larger diversity of intelligent means for accessing the web site, *viz.* semantic ranking, personalization and machine access by crawling. Thus, we have shown a comprehensive approach to meet many of the challenges put forth in [Perkowitz & Etzioni, 1997].

For the future, we see a number of new important topics appearing on the horizon. For instance, we consider approaches for ontology learning [Maedche & Staab, 2001] in order to semi-automatically adapt to changes in the world and to facilitate the engineering of ontologies. Currently, we work on providing intelligent means for providing semantic information, *i.e.* we elaborate on a semantic annotation framework that balances between manual provisioning from legacy texts (*e.g.* web pages) and information extraction [Staab, Maedche, & Handschuh, 2001]. Given a particular conceptualization, we envision that one wants to be able to use a multitude of different inference engine taking advantage of different inferencing capabilities (temporal, nonmonotonic, high scalability, etc.). Then, however, one needs means to change from one representation paradigm to the next [Staab, Erdmann, & Maedche, 2001] .

Finally, we envision that once semantic web sites are widely available, their automatic exploitation may be brought to new levels. Semantic web mining considers the level of mining web site structures, web site content, and web site usage on a semantic rather than at a syntactic level yielding new possibilities, *e.g.* for intelligent navigation, personalization, or summarization, to name but a few objectives for semantic web sites [Hotho & Stumme, 2001].

Acknowledgements.

The research presented in this paper would not have been possible without our colleagues and students at the Institute AIFB, University of Karlsruhe, and Ontoprise GmbH. We thank Jürgen Angele, Kalvis Apsitis (now: RITI Riga Information Technology Institute), Nils Braeunlich, Stefan Decker (now: Stanford University), Michael Erdmann, Dieter Fensel (now: VU Amsterdam), Siegfried Handschuh, Andreas Hotho, Mika Maier-Collin, Daniel Oberle, and Hans-Peter Schnurr. Research for this paper was partially financed by Ontoprise GmbH, Karlsruhe, Germany, by US Air Force in the DARPA DAML project “OntoAgents”, by EU in the IST-1999-10132 project “On-To-Knowledge” and by BMBF in the project “GETESS” (01IN901C0).

References

[Anand et al., 2000] S.S. Anand, M. Baumgarten, A. Buechner, and M. Mulvenna. Gaining insights into web customers using web intelligence. In Proceedings of ECAI’2000, pages 681–685, Berlin, Germany, August 2000.

[Angele et al., 2000] J. Angele, H.-P. Schnurr, S. Staab, and R. Studer. The times they are a-changin’ —the corporate history analyzer. In D. Mahling and U. Reimer, editors, Proceedings of the Third International Conference on Practical Aspects of Knowledge Management. Basel, Switzerland, October 30-31, 2000, 2000. http://www.research.swisslife.ch/pakm2000/.

[Benjamins & Fensel, 1998] V. Richard Benjamins and Dieter Fensel. Community is knowledge! (KA). In Proceedings of the 11th Workshop on Knowledge Acquisition, Modeling, and Management (KAW ’98), Banff, Canada, April 1998, 1998.

[Chakrabarti, van den Berg, & Dom, 1999] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: a new approach to topicspecific web resource discovery. In Proceedings of WWW-8, 1999.

[Dalitz, Gr¨otschel, & Lügger, 1997] W. Dalitz, M. Gr¨otschel, and J. Lügger. Information Services for Mathematics in the Internet (Math-Net). In A. Sydow, editor, Proceedings of the 15th IMACS World Congress on Scientific Computation: Modelling and Applied Mathematics, volume 4 of Artificial Intelligence and Computer Science, pages 773–778. Wissenschaft und Technik Verlag, 1997.

[Decker et al., 1999] S. Decker, M. Erdmann, D. Fensel, and R. Studer. Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information. In R. Meersman et al., editors, Database Semantics: Semantic Issues in Multimedia Systems, pages 351–369. Kluwer Academic Publisher, 1999.

[Decker et al., 2000] S. Decker, D. Fensel, F. van Harmelen, I. Horrocks, S. Melnik, M. Klein, and J. Broekstra. Knowledge representation on the web. In Proceedings of the 2000 International Workshop on Description Logics (DL2000), Aachen, Germany, 2000.

[Fensel et al., 1998] D. Fensel, S. Decker, M. Erdmann, and R. Studer. Ontobroker: The Very High Idea. In Proceedings of the 11th International Flairs Conference (FLAIRS-98), Sanibel Island, Florida, May, 1998.

[Gomez-Perez, 1996] A. Gomez-Perez. A framework to verify knowledge sharing technology. Expert Systems with Application, 11(4):519–529, 1996.

[Guarino & Welty, 2000] N. Guarino and C. Welty. Identity, unity, and individuality: Towards a formal toolkit for ontological analysis. Proceedings of ECAI-2000, August 2000. available from http://www.ladseb.pd.cnr.it/infor/ontology/Papers/OntologyPapers.html.

[Heflin & Hendler, 2000] J. Heflin and J. Hendler. Searching the web with shoe. In Artificial Intelligence for Web Search. Papers from the AAAI Workshop. WS-00-01, pages 35–40. AAAI Press, 2000.

[Hotho & Stumme, 2001] A. Hotho and G. Stumme, editors. Semantic Web Mining — Workshop at ECML-2001 / PKDD- 2001, Freiburg, Germany, 2001.

[Hovy, 1998] E. Hovy. Combining and standardizing large-scale, practical ontologies for machine translation and other uses. In Proc. of the First Int. Conf. on Language Resources and Evaluation (LREC), 1998.

[Kifer, Lausen, & Wu, 1995] M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. Journal of the ACM, 42:741–843, 1995.

[Kosala & Blockeel, 2000] R. Kosala and H. Blockeel. Web mining reseach: A survey. SIGKDD Explorations, 2(1):1–15, 2000.

[Maedche & Staab, 2000] A. Maedche and S. Staab. Discovering conceptual relations from text. In Proceedings of ECAI- 2000. IOS Press, Amsterdam, 2000.

[Maedche & Staab, 2001] A. Maedche and S. Staab. Ontology learning for the semantic web. IEEE Intelligent Systems, 16(2), 2001.

[Menzis, 1998] T.J. Menzis. Knowledge maintenance: The state of the art. The Knowledge Engineering Review, 10(2), 1998.

[Miller, 1995] G. Miller. Wordnet: A lexical database for English. CACM, 38(11):39–41, 1995.

[Odgen & Richards, 1993] C.K. Odgen and I.A. Richards. The Meaning of Meaning: A Study of the Influence of Language upon Thought and of the Science of Symbolism. Routledge & Kegan Paul Ltd., London, 10 edition, 1923.

[Perkowitz & Etzioni, 1997] M. Perkowitz and O. Etzioni. Adaptive web sites: an AI challenge. In Proceedings of the 15th International Joint Conference on AI (IJCAI-97), pages 16–23, Nagoya, Japan, August 23-29 1997.

[Perkowitz & Etzioni, 1999] M. Perkowitz and O. Etzioni. Adaptive web sites: Conceptual cluster mining. In Proceedings of the 16th International Joint Conference on AI (IJCAI-99), pages 264–269, 1999.

[FAQ] personalization.com. Frequently asked questions: Are there different types of personalization? http://www.personalization.com/basics/faq/faq2.asp observed at Feb 14, 2001, 2001.

[Rada et al., 1989] R. Rada, H. Mili, E. Bicknell, and M. Blettner. Development and application of a metric on semantic nets. IEEE Transactions on Systems, Man, and Cybernetics, 19(1), 1989.

[Resnik, 1995] P. Resnik. Knowledge maintenance: The state of the art. In Proceedings of IJCAI-95, pages 448–453, Montreal, Canada, 1995.

[Richardson, Smeaton, & Murphy, 1994] R. Richardson, A. F. Smeaton, and J. Murphy. Using wordnet as knowledge base for measuring semantic similarity between words. Technical Report CA-1294, Dublin City University, School of Computer Applications, 1994.

[Sowa, 1992] J.F. Sowa. Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley, 1992.

[Srivastava et al., 2000] J. Srivastava, R. Cooley, M. Deshpande, and P. Tan. Web usage mining: Discovery and applications of usage patterns from web data. SIGKDD Explorations, 1(2):12–23, 2000.

[Staab et al., 2000] S. Staab, J. Angele, S. Decker, M. Erdmann, A. Hotho, A. Maedche, H.-P. Schnurr, R. Studer, and Y. Sure. Semantic community web portals. Proc. of WWW9 / Computer Networks, 33(1-6):473–491, 2000.

[Staab, Erdmann, & Maedche, 2001] S. Staab, M. Erdmann, and A. Maedche. Semantic patterns. Technical report, Institute AIFB, University of Karlsruhe, 2001.

[Staab & Maedche, 2001] S. Staab and A. Maedche. Knowledge portals — ontologies at work. AI Magazine, 21(2), Summer 2001.

[Staab, Maedche, & Handschuh, 2001] S. Staab, A. Maedche, and S. Handschuh. An annotation framework for the semantic web. In Proceedings of the First Workshop on Multimedia Annotation, Tokyo, Japan, January 30-31, 2001, 2001.

[Staab et al., 2001] S. Staab, H.-P. Schnurr, R. Studer, and Y. Sure. Knowledge processes and ontologies. IEEE Intelligent Systems, 16(1), 2001.

[Sure, Maedche, & Staab, 2000] Y. Sure, A. Maedche, and S. Staab. Leveraging corporate skill knowledge - From ProPer to OntoProper. In D. Mahling and U. Reimer, editors, Proceedings of the Third International Conference on Practical Aspects of Knowledge Management. Basel, Switzerland, October 30-31, 2000, 2000. http://www.research.swisslife.ch/pakm2000/.

[Uschold & Gruninger, 1996] M. Uschold and M. Gruninger. Ontologies: Principles, methods and applications. Knowledge Sharing and Review, 11(2), June 1996.

[Weibel et al., 1998] S. Weibel, J. Kunze, C. Lagoze, and M. Wolf. Dublin Core Metadata for Resource Discovery. Number 2413 in IETF. The Internet Society, September 1998.

Footnotes

[1] Institute AIFB, University of Karlsruhe, D-76128 Karlsruhe, Germany, http://www.aifb.uni-karlsruhe.de/WBS, mailto: {ama, sst, nst, rst, ysu}@aifb.uni-karlsruhe.de

[2] Ontoprise GmbH, Haid-und-Neu Straße 7, 76131 Karlsruhe, Germany, http://www.ontoprise.de

[3] ZI Research Center for Information Technologies, Haid-und-Neu Straße 10-14, 76131 Karlsruhe, Germany, http://www.fzi.de/wim

[4] ere at the conceptual level, we do not distinguish between relations and attributes.

[5] ctive server pages *vs.* active service providers.

[6] “SouthEast Asian Linguistics Conference” *vs.* “Conference on Simulated Evolution and Learning” *vs.* “Society for Evolutionary Analysis in Law” *vs.* “Society for Effective Affective Learning” *vs.* some other dozens —several of which are indeed relevant in our institute.

[7] Negation requires special treatment.

[8] Experts might store any possible query into bookmarks by manually editing the queries.

[9] Our system is based on java servlets and uses URL-encoded queries.

[10] RDF CRAWLER is freely available for download at http://ontobroker.semanticweb.org/rdfcrawler.

[11] RDF-API is freely available at http://www-db.stanford.edu/˜melnik/rdf/api.html.

[12] The interested reader may have a look at http://www.oakland.edu/˜grossman/erdoshp.html for an overall project overview.

[13] http://www.yahoo.com

[14] http://www.purl.org/dc

[15] *cf.* Mathematical Subject Classification; http://www.ams.org/msc/

III. Dynamic aspect

12. Semantic Gadgets: Ubiquitous Computing Meets the Semantic Web

Ora Lassila[1] & Mark Adler[1]

1. Introduction

Ubiquitous computing is an emerging paradigm of personal computing, characterized by the shift from dedicated computing machinery (that requires the user’s attention e.g., PCs) to pervasive computing capabilities embedded in our everyday environments [Weiser, 1991], [Weiser, 1993]. Characteristic to ubiquitous computing are small, handheld, wireless computing devices. The pervasiveness and the wireless nature of devices require network architectures to support automatic, *ad hoc* configuration [e.g., Huitema, 1998]. An additional reason for development of automatic configuration is that this technology is aimed at ordinary consumers.

A key technology of true ad hoc networks is *service discovery*, functionality by which "services" (i.e., functions offered by various devices on an *ad hoc* network) can be described, advertised, and discovered by others. Several frameworks and formalisms for service discovery and capability description have already emerged. Examples of these include Sun’s "Jini" and Microsoft’s Universal Plug and Play (UPnP) [e.g., Richard, 2000] as means of describing services and invoking them, as well as World Wide Web Consortium’s (W3C)

Composite Capability/Preference Profile (CC/PP) [Reynolds et al., 2000] as a means of describing device characteristics.

All of the current service discovery and capability description mechanisms are based on *ad hoc* representation schemes and rely heavily on standardization (i.e., on *a priori* identification of all those things one would want to communicate or discuss). In addition, "Jini" relies on the Java object system and instance serialization, and UPnP uses XML and its own flavor of HTTP.

UPnP, for example, aims to render a user interface for some device (on another device) and facilitate the remote control by a human via the second device. In contrast, our approach aims to accomplish this task without the human in the loop. In other words, the ultimate goal is the discovery and utilization of services by other automated systems without human guidance or intervention, thus enabling the automatic formation of *device coalitions* through this mechanism. We call these devices, capable of semantic discovery and coalition formation, *semantic gadgets*.

2. About Representation

Various Artificial Intelligence technologies, particularly from the area of Knowledge Representation (KR) will be useful and necessary to build "richer" discovery mechanisms. Given that we are dealing with networked devices and distributed software systems, the notion of the *semantic web* [e.g., Fensel, 2000] fits well in this context. Technically speaking, by semantic web we mean the emerging knowledge representation formalism DAML (DARPA Agent Markup Language, [Hendler & McGuinness, 2000]) and its foundation, W3C's RDF (Resource Description Framework, [Lassila, 1998], [Lassila & Swick, 1998], [Brickley & Guha, 1999]). In broad terms, the semantic web encompasses efforts to populate the Web with content which has formal semantics; thus the semantic web will enable automated agents to reason about Web content, and produce an intelligent response to unforeseen situations. Our vision is to overlay the semantic web on a ubiquitous computing environment, making it possible to represent and interlink devices, their capabilities, and the functionality they offer.

RDF is the W3C's standard for metadata, but it extends the traditional notion by allowing any "resource" on the WWW to be described. Because RDF uses Uniform Resource Identifiers (URIs) for addressing, it can even describe things that are not on the Web as long as they have a URI. RDF serves as the foundation for DAML, an emerging, more expressive KR language, suitable for building ontologies. We foresee that RDF will describe services and device functionality for the purposes of discovery. Just as the success of the deployment of the semantic web will largely depend on whether useful ontologies will emerge, so will discovery services benefit from mechanisms that allow shared agreements about vocabularies for knowledge representation. Why is this important? Sharing vocabularies and models allows automated interoperability; given a base ontology shared by two agents, each agent can extend the base ontology while achieving *partial understanding*. A base ontology is analogous to OOP systems, where a base class defines "common" functionality.

To simply describe devices and functionality, frame-based representation, with its close kinship to more widely accepted technologies such as object-oriented modeling and programming, may be the right paradigm of KR [Lassila & McGuinness, 2001]. The concept of a *frame* was proposed in the 1970's [Minsky, 1975], and *frame systems* subsequently gained ground as basic tools for representing knowledge [Fikes & Kehler, 1985], [Karp, 1992] ,[Chaudhri et al., 1998]. The idea of a frame system is simple: A frame represents an object or a concept. Attached to the frame is a collection of attributes (slots), potentially filled initially with 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. In the context of device description one could think of a device as a frame, and to represent its characteristics as slots (this, in fact, is the approach taken by CC/PP in its use of RDF).

3. Scenario: Semantic Gadget In a Museum

To illustrate how a semantic gadget (e.g., a next generation personal communicator, subsequently referred to as "SG") will work, and particularly how it will integrate and interoperate with other semantic devices, we have constructed a hypothetical scenario of gadget usage in an art museum instrumented with semantic devices and services.

In the car: As we approach the art museum, we check the dashboard map for parking options. Our museum membership entitles us to reduced parking fees at the two lots that are highlighted. A snow storm is predicted; knowing our dislike of cleaning snow off the car, the SG suggests parking in the garage instead of parking in the outside lot, even though parking in the garage means a longer walk to the entrance and paying a slightly higher fee. The gate lifts as we approach, and the dashboard map indicates an available parking spot on the third floor near the exit. We park, grab our mobile SG, and head for the stairs as our vehicle shuts off the lights and arms the security system. Our SG records the location of the car, and plots our route to the nearest museum entrance.

In the museum: As we enter the art museum, our SG negotiates our admission by supplying our membership number and confirming our reservation for the exhibit in the main gallery. Before leaving the lobby, the SG suggests that we check our coats because the temperature inside the museum is too warm to wear a coat, and the SG knows how much we dislike carrying our coats. A map on the SG guides us to the coatroom. The gallery is currently near capacity, so our SG recommends a detour through other galleries to see some recent impressionistic acquisitions. A map of the museum with the suggested path appears on the touch screen. As we follow the path, a more detailed view appears, with a flashing icon for the paintings that are of special interest. As we approach a painting, additional icons appear on the screen, indicating available information about the artist, the period, details about the museum's acquisition of the painting, and pointers to related art works. By clicking on the icons, we obtain the underlying information.

After a pleasant 15-minute diversion strolling through several galleries, we arrive at the main exhibition and enter without any delay. The first gallery seems crowded, but the SG suggests viewing the exhibit in reverse order because we prefer to avoid crowded viewing conditions. Using the Bluetooth-enabled headset, we listen to the museum's descriptions as we stroll through the exhibit. As we make our way through the exhibit again on the way to the exit, the SG reminds us of several displays that we missed the first time we walked through.

In the gift shop: Naturally, one cannot exit without passing through the gift shop. Ordinarily, we avoid this spending opportunity, but the SG flashes a message reminding us that we need a gift for my mother's birthday the following week. Based on her preference for writing notes long hand and her love of impressionistic art, the SG suggests purchasing a box of Monet stationery. The SG interacts with the cashier, obtains the member discount, charges the purchase to my credit card, and arranges to have the stationery gift-wrapped and shipped in time for Mom’s birthday.

Exiting the museum: As we leave the gift shop, the SG presents a list of suggested activities: It proposes (1) viewing additional art works that are highly correlated to the paintings that we spent the most time viewing; (2) attending a lecture associated with the exhibit; (3) eating at the museum restaurant; or (4) leaving the museum and going home. We choose the latter, indicating our choice verbally. The SG maps our path to exit and return to our car.

In the parking garage: As we approach the car, the SG unlocks the doors. We plug in the SG to recharge, and the display switches to the dashboard screen. Our SG handles the parking fee, including our member discount, and we are on our way home.

4. Semantic Discovery

Semantic gadget technology begins with the discovery of functionality and services. As mentioned before, a number of mechanisms for low-level service discovery have emerged (examples include Sun's "Jini" and Microsoft's Universal Plug and Play); these mechanisms attack the problem at a syntactic level, and rely heavily on the standardization of a predetermined set of functionality descriptions. Standardization, unfortunately, can only take us halfway toward our goal of intelligent automated behavior *vis-a-vis* discovery, as our ability to anticipate all possible future needs is limited. The semantic web offers the possibility to elevate the mechanisms of service discovery to a "semantic" level. Here a more sophisticated description of functionality is possible, and the shared understanding between the consumer and the service provider can be reached via the exchange of ontologies, which provide the necessary vocabulary for a dialogue.

Semantic discovery mechanisms will undoubtedly be layered on top of existing, lower-level services. These services involve *ad hoc* networking technologies and other mechanisms that are beyond the scope of this article. It is sufficient to state that physical devices, their functionality and the services they offer, will be abstracted as *software agents*. These agents will advertise services (if providers) and/or will query for services they need (if consumers). Both the advertisements and the queries will be abstract descriptions of functionality (in fact, there is no difference between the two, as Sycara has pointed out [Sycara et al., 1999]). Through a matchmaking process (either by the provider, by the consumer, or by a third party "matchmaker") we are able to associate compatible advertisements with queries. The match might be *perfect* (in which case a discovered service will exactly meet the need of the consumer) or *partial* (in which case the consumer might have to combine the service with some additional functionality; it can do this by itself, or – as we will demonstrate later – continue to discover the "missing pieces" and finally compose a service that meets its need).

The semantic web plays two key roles in the discovery process. First, semantic web techniques provide a rich mechanism for describing functionality. This is important in building and exploiting ontologies that describe the concepts and vocabulary needed to discuss functionality and services. Second, the semantic web provides a unifying layer of naming and distribution – that is, an addressing mechanism that can encompass virtual and physical entities, and a way for various pieces of the "puzzle" to reside on various servers, devices etc. For example, a device description can refer to an ontology elsewhere, and this ontology in turn can be a specialization or extension of another ontology, again somewhere else. As mentioned earlier, this *polymorphic* nature of ontologies and their extensions will allow broader interoperation through partial understanding and agreement.

5. Contracting the Use of Services

Once we have discovered and identified the service (or services) that we want to use, there are several rather "bureaucratic" issues to be dealt with. These include (but are not limited to):

- assuring security, privacy, and trust,
- compensating the service provider, and
- determining execution locus.

The semantic web promises to be extremely useful when it comes to matters of security, privacy and trust, as these are largely issues of representation. Generally, the semantic web rests heavily on a framework of trust partially constructed using digital signatures and other types of cryptographic certificates. Any agent can expose its reasoning about trust using the same mechanisms by which everything else is represented on the semantic web. These representations of reasoning (we could call them "proofs") might themselves be exchanged between agents as a means of more "compelling" communication (e.g., for access to restricted resources or services).

We anticipate services specific to assisting agents with their security, privacy, and trust issues (e.g., there might be a service that rates the "reputation" of other services: reliability, trustworthiness, or some other relevant metric). Again, these services themselves can be discovered and their use contracted.

Given a functional security and trust framework, we can introduce the notion of *payments*. This is required for the purpose of compensating providers for services rendered. Generally, we anticipate third-party services (which, again, are discoverable) to facilitate payments or other types of compensation. Note that we are not trying to imply that everything on the semantic web will cost something, but certainly some services will emerge that are not free.

Advertising (of products and services *for humans* this time) will no longer be as viable as a revenue model once automated agents take care of a large part of the information exchange, reasoning and service utilization. The interoperability of physical devices is an objective worth pursuing: the devices have typically already been purchased, and it is unlikely that a single company will be engaged in the manufacture of all the different types of devices that we anticipate will make up the web of semantic gadgets (for example, Nokia makes cellular phones and other wireless communication devices but it does not make thermostats; yet the consumer might benefit from a wireless device that communicates with a thermostat).

As an issue related to security, the determination of execution locus is an interesting issue. A particular service might be executed by the agent offering the service (the typical case), or it might involve *trusted mobile code* downloaded to the requester’s computational environment, or some combination thereof. We will assume that this issue is resolved and acknowledge that it is beyond the scope of this article.

6. Composition of Services

The discovery of services based on some description of a requirement of new functionality might result in a partial match [e.g., Sycara et al., 1999]. In this case the requester can attempt to provide the missing parts itself or continue the discovery process and identify other services. They can then be pieced together to form an aggregate service that fulfills the original requirement.

Composition of exact required functionality from partially matched discovered services should be viewed as a process of *goal-driven assembly*. For example, automated planning or configuration techniques could be used to achieve the assembly. In the context of ubiquitous computing and semantic gadgets, contracting the use of services should always be viewed as a goal-driven activity because of the "volatile" nature of ubiquitous computing environments (not only can any of the devices and services fail or be removed from the environment at any time, but new ones can be added to it; thus *opportunistic* exploitation of services might be beneficial).

The goal-driven approach takes information system interoperability beyond what mere standardization or simple interface sharing enables, since it is based on "deeper" descriptions of service functionality and can be performed *ad hoc* and on demand. In fact, the dynamically composed aggregate services are the semantic web’s virtual equivalent of "real-life" *value chains*: large quantities of information (i.e., "raw material") may be obtained, and at each step the value of information increases and its volume decreases.

Given that the services discovered represent actual (possibly physical) functionality of devices, aggregate services could be seen as *device coalitions*. Not only can individual devices extend their functionality, but the device coalitions form a type of "super-devices".

7. Museum Scenario Revisited: an Analysis

There are several technologies that are required to implement the ubiquitous computing functionality illustrated in the art museum scenario:

- *context awareness* (including location, outside using GPS, and some analogous system inside, as well as additional sensory data that might be relevant),
- *service discovery* (finding available service providers in a wireless network),
- requirements/preferences (making the user's desires known to other service providers),
- user interface design (touch screen, voice input, speech output, etc.),
- the ability to match requirements to services (including planning functions to compose services to meet the user's needs), and
- *machine learning* to improve performance over time, and adapt to better meet the user's needs.

Many of these technologies fall within the scope of this article. Generally, enabling all this functionality is a rich, shared, frame-based representation of the underlying information.

In the car: Within the confines of the car, various gadgets can establish a network connection through Bluetooth, or a wired connection such as a recharging port, so the SG can use the dashboard display, the radio speakers, onboard computing resources, sensors, etc. Today's cars already have GPS sensors, maps, and wireless connectivity. In this scenario, we combine these tools with the ability to access weather reports, and digital wallet features including museum membership information, user preferences, and the ability to communicate with devices outside the bounds of the car itself like the parking garage tollgate. Finally, recording the car’s location in the garage requires some location service inside the garage, since GPS is not effective inside a building. Beyond the connectivity issues, quite a bit of information must be exchanged to provide this functionality. Location information is of course one of the keys here, but so are personal preferences and tradeoffs among them, privacy of personal information, and knowledge of typical tasks, such as the steps involved in parking a car in a public garage, and shutting down network connections that are severed when the SG is removed from the car's environment.

In the museum: In this non-familiar environment, the SG must use its own user interfaces rather than rely on available peripherals. In addition, discovery of available services is of utmost importance. Here we see examples of services that include an automated box office, interactions with temperature sensors, crowd capacity services, interior location and map services, coat check services, and especially catalogs of interrelated knowledge about art works, and again user preferences combined with the monitoring of our progress through the exhibit. In a ubiquitous computing environment, there might be several possible implementations that could provide this functionality to the user. Privacy concerns may dictate that each person's own SG plays an active role in tracking each users progress, rather than allowing sensors in the room to capture one's position over time. The availability of required knowledge over network access is again critical to these functions.

In the gift shop: The shopping experience in the ubiquitous computing world has little to do with the fact that this gift shop is in the museum. Here the SG combines personal calendar information about birthdays, conventions about birthday gifts [Charniak, 1972], personal preference information (in this case for others, involving trust and requiring the system to deal with privacy issues), and discovery of descriptions of available purchase inventory.

It is an interesting separate problem to determine what the appropriate context is for the SG to trigger the gift reminder. Not every shopping opportunity is appropriate: for example, it would seem out of place for such a reminder to appear while shopping for groceries.

The act of purchasing the item involves the digital wallet again, this time including credit card charges as well as museum membership, and the availability of a member discount. An example of service composition is illustrated by the combination of additional services of gift-wrapping and shipping (which also requires address lookup capabilities).

Exiting the museum: One of the interesting aspects of this scenario is that of context. As we leave the gift shop, we are changing context. We have completed that scenario and are ready to proceed to the next one. It is important that the SG can determine these transition points as we move through our daily tasks.

At these transition points, there are multiple knowledge sources that influence the possible choices. There are additional activities at the museum, our current frame of reference, but there are opportunities outside of this context. Eating at the museum restaurant is an interesting combination that would satisfy the urge to eat, while leaving available additional museum activities after the meal.

In the parking garage: Returning to the car involves a reversal of the car security features that were enacted on our arrival. Docking the SG provides a means for recharging, but it could also be a convenient way to transfer information for backup or additional planning purposes.

Conclusions

We have presented how semantic web technologies can be used in the context of ubiquitous computing to enrich the capabilities of service discovery mechanisms. By abstracting device functionality as software agents, and then using semantic web technologies (e.g., distributed frame-based representation) to describe agent services, we can build a "semantic" discovery service capable of function beyond *a priori* standardization. Through the composition of partially matching (discovered) services into virtual value chains we are able to form device coalitions which opportunistically exploit a dynamically changing ubiquitous computing environment.

References

[Brickley & Guha, 2000] Dan Brickley & R.V.Guha: "Resource Description Framework (RDF) Schema Specification 1.0", W3C Candidate Recommendation 2000-03-27

[Charniak, 1972] Eugene Charniak: "Toward a Model of Children's Story Comprehension", MIT AI Lab Tech Report AI-TR-266, December 1972

[Chaudhri et al., 1998] Vinay Chaudhri, Adam Farquhar, Richard Fikes, Peter Karp & James Rice; "OKBC: A Programmatic Foundation for Knowledge Base Interoperability", Proc. AAAI 1998

[Fensel, 2000] Dieter Fensel (ed.): "The Semantic Web and its Languages", IEEE Intelligent Systems 15(6): 67-73 (November/December 2000)

[Fikes & Kehler, 1985] Richard Fikes & Tom Kehler: "The Role of Frame-Based Representation in Reasoning", CACM 28(9): 904-920 (1985)

[Hendler & McGuinness, 2000] James Hendler & Deborah L. McGuinness: "The DARPA Agent Markup Language", in [Fensel, 2000]

[Huitema, 1998] Christian Huitema: "Plug and Play", in IPv6: the New Internet Protocol, Prentice-Hall, Upper Saddle River (NJ), 1998

[Karp, 1992] Peter D. Karp: "The design space of frame knowledge representation systems", Technical Report 520, SRI International AI Center, 1992

[Lassila, 1998] Ora Lassila: "Web Metadata: A Matter of Semantics", IEEE Internet Computing 2(4): 30-37 (1998)

[Lassila & McGuinness, 2001] Ora Lassila & Deborah L. McGuinness: "The Role of Frame-Based Representation on the Semantic Web", Electronic Transactions on AI (to appear); available as a Stanford KSL technical report KSL-01-02

[Lassila & Swick, 1999] Ora Lassila & Ralph Swick: "Resource Description Framework (RDF) Model and Syntax Specification", W3C Recommendation 1999-02-22

[Minsky, 1975] Marvin Minsky: "A Framework for Representing Knowledge", in Patrick Henry Winston (ed.), The Psychology of Computer Vision, McGraw-Hill, New York (NY), 1975

[Reynolds et al., 2000] Franklin Reynolds, Chris Woodrow & Hidetaka Ohto: "Composite Capability/Preference Profiles (CC/PP): Structure", W3C Working Draft 2000-07-21

[Richard, 2000] Golden G. Richard III: "Service Advertisement and Discovery: Enabling Universal Device Cooperation", IEEE Internet Computing 4(5): (2000)

[Sycara et al., 1999] Katia Sycara, Matthias Klusch, Seth Widoff & Jianguo Lu: "Dynamic Service Matchmaking Among Agents in Open Information Environments", ACM SIGMOD Record 28(1): 47-53 (March 1999)

[Weiser, 1991] Mark Weiser: "The Computer for the Twenty-First Century", Scientific American, September 1991

[Weiser, 1993] Mark Weiser: "Some Computer Science Problems in Ubiquitous Computing", CACM, July 1993

Footnotes

[1] Nokia Research Center, 5 Wayside Road, Burlington MA 01803, USA

13. Static and Dynamic Semantics of the Web

Christopher Fry[1], Mike Plusch[2] and Henry Lieberman[3]

1. Introduction

The original perception of the Web by the vast majority of its early users was as a static repository of unstructured data. This was OK for browsing small sets of information by humans, but this static model is now breaking down as programs attempt to dynamically generate information, and as human browsing is increasingly assisted by intelligent agent programs.

The next phase of the Web, as represented in this book's movement towards the "Semantic Web", lies in encoding properties of and relationships between, objects represented by information stored on the Web. It is envisioned that authors of pages include this semantic information along with human-readable Web content, perhaps with some machine assistance in the encoding. Parsing unstructured natual language into machine-understandable concepts is not feasible in general, although some programs may be able to make partial sense out of Web content.

However, this semantics is primarily *declarative* semantics, semantics that changes only relatively slowly as Web pages are created, destroyed, or modified, typically by explicit, relatively coarse-grained human action.

Less concern has been given to *dynamic* semantics of the Web, which is equally important. Dynamic semantics have to do with the *creation* of content, actions which may be guided by

- User-initiated interface actions
- Time
- Users' personal profiles
- Data on a server

and other conditions.

Even less concern has been given to methods for cleanly integrating the static and dynamic aspects of the Web. In this paper, we discuss the need for dynamic semantics, and show how dynamic semantics will be enabled by and useful to the new generation of intelligent agent software that will increasingly inhabit the Web. Though there will be autonomous agents, the more interesting agents will cooperate interactively with humans, helping them to achieve their goals more efficiently than a user could on their own. We also present a new language, *Glue*, that fully integrates static and dynamic semantics, while keeping close to the XML framework that has made the Web successful.

Beyond that, we envision that Web end users and Web application developers will not be routinely writing code directly in Glue or any other formal semantic language, but instead avail themselves of Programming by Example [Lieberman, ed., 2001] and other interactive agent interfaces that will hide the details of the formal languages. But transition to these future interfaces will be greatly aided by a foundation that can cleanly integrate static and dynamic semantics.

2. Static Semantics

2.1. The Web's Link Structure Mimics a Semantic Net

A semantic net [Woods, 1975] is a network of concepts linked by relations. The Web is, of course, a network of pages, each containing text, pictures, other media types, and links to other Web pages. Though the Web has far less structure than typical AI semantic nets, the Web pages that constitute the nodes of the Web's network often do represent concepts and the links between them represent relations between those concepts. For example, my home page is the Web's representation of me, in a sense. The links leading off my home page - to my publication list, my e-mail address, courses that I teach, etc. - represent the relation of me to, e.g. the articles I have published.

The only problem is that these relations are expressed in human-understandable natural language and human-understandable pictures. Short of full natural language understanding, it is difficult for a computer program to automatically extract those concepts and relations in order to do query answering, inference and other tasks.

2.2. Should the Semantic Web Be Static?

The Semantic Web movement, represented in many of the articles in this book, and in Web standards initiatives such as RDF, DAML, and OIL, is an attempt to introduce common formal languages for expressing concepts and relations in a machine readable way. To leverage existing Web tools and emulate the Web's social success, such efforts strive to embed the descriptive information in pages similar to the way text, pictures and conventional media are already described using HTML and XML.

However, all this is basically limited to *static* semantics. The interaction paradigm is that knowledge descriptions are authored by a developer in the same way that HTML pages are presently authored by a Web designer, then "published" to the Web to make them available in the network. Sometimes, the descriptions may be automatically produced by a program from the results of user interaction in a manner similar to the way many WYSIWYG Web editors such as Macromedia Dreamweaver or Adobe GoLive produce HTML automatically from user editing operations.

In all of this, there is the underlying assumption that semantics

- Is *represented declaratively.* It is represented in passive data that is descriptive and can be retrieved, e.g. a Web page on a Web server, and

- *Changes relatively slowly.* The Web publishing events happen rarely relative to actions such as browsing a page or clicking a link.

Structured static semantics are a huge advance for the web since they will enable software agents to find and reason about a colossal volume of information, essentially turning mere data into knowledge. But the larger and more complex the knowledge, the less complete and useful static representations become.

3. Dynamic Semantics

In addition to the static semantics of Web pages, links, and Web markup, there is also what we call dynamic semantics. Dynamic semantics

- Is *represented procedurally*. It can be computed by programs running on the client or server side, based on immediate interactive user input. This computation can depend on the immediate *context* - including time, personal information about the user, user-initiated actions, etc.
- *Changes relatively rapidly*. A single user click can cause the semantics to be generated or to change, or it can be changed by the actions of programs continuously in real-time.

As the web matures, there are many ways in which static semantics are being augmented and supplanted by dynamic semantics. As a simple example, some URLs are not addresses of static pages stored on Web servers, but rather act as directives to the server to initiate some computation. CGI scripts are an example of this. The question-mark in the URL is a signal for the server to retrieve some named program and execute it, possibly with arguments. An Active Server Page queries a database and constructs a page on the fly. Even search engine results pages, and customized ads based on cookies are examples of dynamically created Web pages. Streaming audio, video and other media also make the Web more dynamic.

3.1. Transparency Between Static and Dynamic Semantics

The key is that, to the browser, the HTTP stream delivered by the procedure is identical to that which would have come from a statically stored page, so that the requester need not concern themselves with the question of whether that information was stored or computed.

That kind of transparence between static and dynamic data is an extremely important property that a system can have. It means that a system can always be extended by replacing some piece of formerly static data with a new, dynamically generated object, as long as the new object's behavior is compatible with the old. This property has long been appreciated in the communities of AI and HCI languages such as

Lisp, Prolog and Smalltalk, though it has been underappreciated in the communities of users of more conventional languages such as C and Java. In the knowledge representation community, this has long been studied under the name *procedural attachment*. Our aim is to extend the principle of equivalence of static and dynamic data as the Web moves toward encoding more semantics.

4. Sources of Dynamic Semantics

In addition to the examples of procedural Web data cited above, there are several other sources of dynamic semantics for the Web. The first is in the process of Web browsing itself.

4.1. Web Browsing Generates Dynamic Semantics

The process of the user navigating through Web pages might result in new objects and relations being created that need to be represented, both on the client's machine and also on the Web servers with which the client communicates. These may either be represented statically and stored explicitly, or produced dynamically as a result of user action. For example, personal information about the user, such as their current interests, might be communicated from the client to the server. Now, cookies are used as a very primitive means of client-to-server communication, but they can only communicate a single piece of information. Information like the user's current interests might be represented by a complex structure with dynamic contingencies.

4.2. Agents Generate Dynamic Semantics

An alternative to having Web page creators explicitly author meta-data, is to have the meta-data computed by agents from human-readable information. An active area of research is having agents "read" Web pages containing natural language and formatted text intended for humans, and compute meta-data by using machine learning to infer "wrappers" that describe the structure of the text [Kushmerick, Weld & Doorenbos, 1997]. This can be done in many cases without having to completely solve the natural language problem. The field of Information Extraction uses partial parsing to perform tasks like semi-automatic topic categorization and summarization. Examples are price-comparison shopbots that extract prices from on-line catalogs, or filters that remove advertisements. Users may even define these wrappers dynamically [Bauer, Dengler & Paul, 2001].

4.3. Web Editing Generates Dynamic Semantics

Finally, the evolution of the Web itself leads to dynamic semantics. As pages are modified, new pages added and old ones disappear, both Web clients and Web servers might want to track and represent how pages change, or keep history that may be dynamically accessible in different ways. This could be as simple as displaying the date of the last change to a page, or highlighting the parts that have changed.

5. Web Agents Make Use of Dynamic Semantics

A revolution that is currently underway is the growing popularity of intelligent agents on the Web. Agents are programs that act as assistants to the user in the interface. They can track user interests, explore the Web proactively, learn through interacting with the user, provide personalized data and services, and much more.

Examples are Letizia and Powerscout [Lieberman, Fry & Weitzman, 2001], which act as *reconnaissance agents*, building a profile of the users' interests by watching his or her Web browsing, and dynamically and incrementally crawling the Web or using traditional search engines as subsidiary tools to suggest related material in real time.

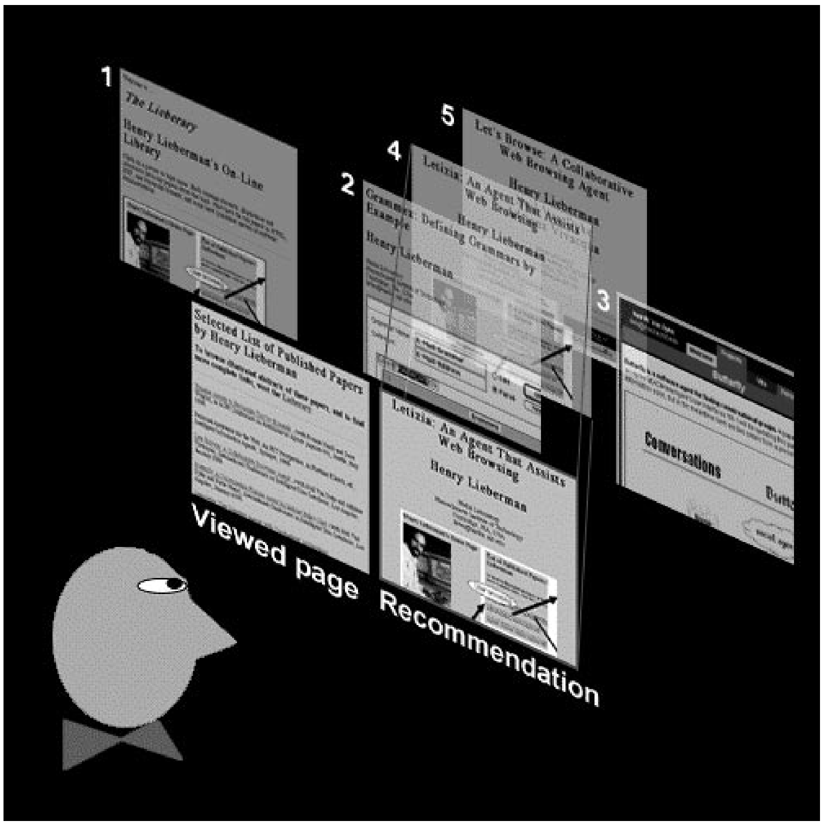


Figure 13.1 Letizia "spirals out" from the user’s current page, filtering pages through the user’s interest profile.

What sets these kinds of agents apart from the more traditional tools like search engines and cookie-personalized sites is their more dynamic nature. They are computing concepts and relations dynamically from the data stored and retrieved on the Web, procedures which are attached to that data, and also from the dynamic process of the user's interactive navigation through the Web sites.

6. Information-Retrieval and Theorem-Proving Perspectives

It is instructive to consider how the advent of the Web changed the perspective of information-seeking activities from the old "information retrieval" model, essentially a static model, to a newer model of dynamic, continuous and cooperative information navigation. As we embed more semantics into the Web, we need to make a similar shift in perspective from the old "theorem proving" model of inference, to a new model of dynamic and cooperative reasoning and problem solving.

Here's a caricature of what we consider the dominant paradigm of Information Retrieval field to have been prior to the advent of the Web. The old Information Retrieval model was that information was stored in a "database", essentially a large, static big bag of "records" and "fields" that changes only slowly [via "database updates"]. The user's only option for interfacing with the database is to throw a "query" at it, a statement of the user's interests in a formal query language like SQL. The user was expected to have in mind a Platonic "ideal document" that was described by the query. The job of the database was to return one or more documents in the database ordered by how well they satisfied the query. Today's search engines are the modern manifestation of this paradigm.

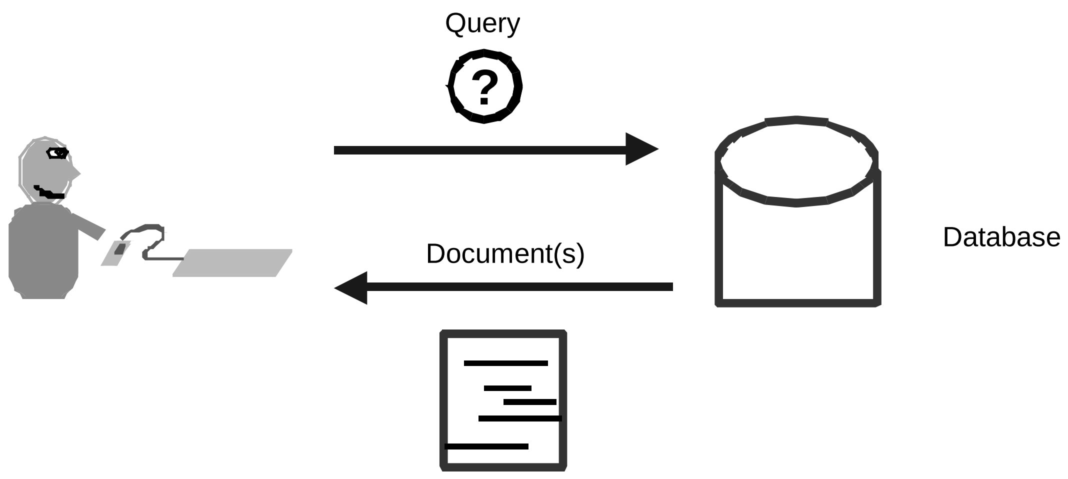


Figure 13.2 The traditional Information Retrieval paradigm.

What's wrong with this paradigm for the Web? Well…

- *It's difficult for users to express queries precisely.* In the old days of IR, users were expert librarians who formulated Boolean queries in formal languages. Now, ordinary users type one or two words to search engines. These don't express intent precisely
- *There may not be a "best document" in the Web.* Any query can potentially return an infinite number of documents. You can't tell which is "best" from a single query alone.
- *Query-response interaction is a "batch processing" view.* It is sequential - either you or the retrieval system are working, not both at the same time. Except for query refinement, each query and response is essentially an independent event, and unrelated to anything else you might be doing.

There are also many other critiques of the old IR paradigm, but that will suffice for the moment.

Agent software on the Web breaks all of these assumptions. Agents like Letizia and Powerscout track the history of user browsing and use it as a persistent context from which to infer user interests. Rather than globally ranking documents, they present context-dependent suggestions that can improve over time. Rather than use the query-response paradigm, they are always active, and deliver their recommendations in a continuous stream in real-time. They essentially integrate browsing and searching into a unified process.

The goal is not to find the best document, but to make the best use of the user's time. In short, they treat browsing as a continuous, co-operative activity between one or more humans and one or more software agents.

6.1. The theorem-proving paradigm

Adding semantics to the Web allows agents to do problem-solving by using traditional theorem-proving techniques to process assertions and descriptions stored on the Web. Again, though, we believe that the paradigm has to change to accommodate the dynamic and fluid nature of the Web. By analogy, we present a caricature of the old theorem-proving paradigm.

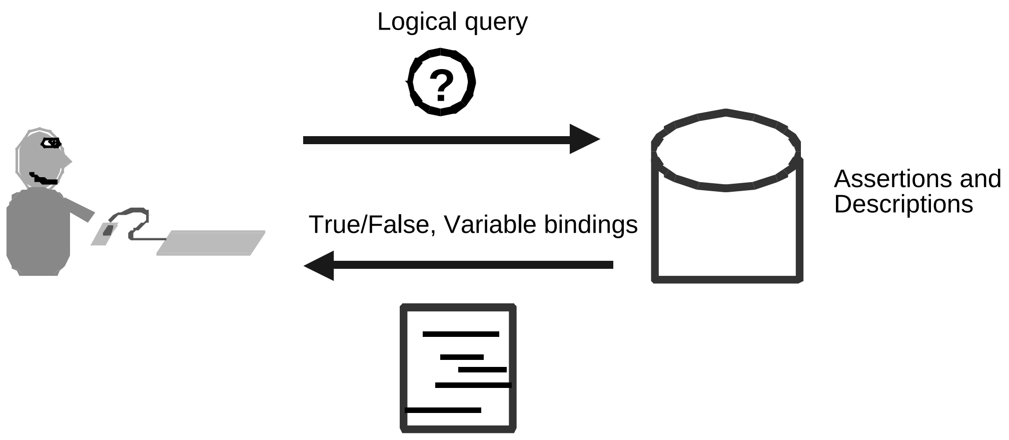


Figure 13.3 The traditional theorem-proving paradigm.

The old theorem-proving paradigm treated a knowledge base as a big bag of assertions and descriptions expressed in a logical language. Again, the way the user was assumed to interact with this was to issue a logical query in the form of an assertion to be proved true or false, possibly filling in the values of logical variables contained in the assertion. This would launch an inference procedure that would find "the answer".

What's wrong? Well…

- *It's difficult for users to express logical queries precisely.* We can't expect users of Web applications to be mathematicians. We need to have interfaces that interact with the user and help formulate queries before they can be sent to an inference system.
- *There may not be a "best answer".* How much is that plane flight? Well, when are you asking? Do you have a discount? Should we ask for a price on Priceline? Buy one on Ebay? How long are you willing to wait for the answer? Can you leave a

day earlier? … We need the ability to put procedural hooks in the answers.

- *Query-response interaction is a "batch processing" view.* Again, in the standard view, no opportunity for dynamic user input or user-agent cooperation is provided for.

Once again, the modern view should be that problem-solving and inference should be a cooperative venture between one or more humans and one or more computer agents. Interaction and parallelism should be available at any point in the process. After all, the expensive component is not the net connection, the client computer or even the servers that store the information, it's the user's *time*. This means that the components of the future Semantic Web should be able to integrate not only text, pictures, links, and semantic descriptions, but also dynamic and procedural objects. But how?

7. The Semantic Web shouldn't sit on the Tower of Babel

The approach to the Semantic Web advocated in this book via languages like DAML and OIL aims to leverage existing Web standards like XML and RDF, and add new facilities that allow expressing formal semantic information embedded in traditional HTML documents and enabling inference. This has the advantage that encoding semantics in the Web becomes an evolutionary and not disruptive step in the evolution of the Web. However, procedural information has not been covered by any of the existing proposals.

7.1. Not my department?

One approach is simply to say, as the current standards do, that it is out of the scope of Web standards to dictate how Web applications will be programmed. Fear of the divisiveness of arguments over programming languages in the computer science community has been motivating this abdication of responsibility for encoding of procedures.

But look where it's gotten us. Many of today's Web applications are programmed piecemeal in a bewildering array of programming languages, for example,

- Javascript
- Java
- Perl
- VBScript
- PHP

just to name a few. It is not uncommon for what appears to the end user as a single Web application to use, in total, 7 or 8 of these languages. The disadvantages of such multi-language environments should be obvious.

- *Difficulty of learning multiple programming languages.* Not only does it duplicate the effort multiple times of learning syntax and semantics, but there is no rational relationship between the languages, e.g. Perl has nothing to do with Java. Maddeningly, despite their names and superficial similarity, Javascript and Java contain major differences in their type and object systems. Having to learn two or more to get a task done is more than twice as hard as one language because the interface between the languages is always additional hair, usually poorly documented. You also must learn *when* to use one language or the other.

- *Difficulty of integrating Web data with multiple languages.* It is worthwhile to keep in mind that the majority of these languages were originally designed for a purpose that had nothing to do with the Web. Java was designed as a control language for small devices; Javascript was a scripting language for Netscape; VBScript as an extension of Basic for Microsoft desktop applications; Perl, for string manipulation via regular expressions. They were all pasted together in an unprincipled manner.
- *Necessity of conversion between different formats.* Conversion wastes computation time and storage and opens up the possibility for errors due to mismatches in data semantics.
- *Difficulty of debugging across multiple languages and systems.* If something goes wrong with a multi-language program, how do you tell who's at fault? None of these languages has anything approaching a decent debugger, even

for programs written solely in that language, so if more than one language is involved, it becomes a nightmare.

And, perhaps the worst problem as far as Web semantics is concerned, is that the plethora of languages prevents representing procedural semantics in any sort of principled way. We need consolidation of existing Web functionality.

8. We need another language like a hole in the head

New Web languages seem to be born monthly. Rather than introduce a new language for implementing some specialized functionality, we recognize that sooner or later that functionality will need general purpose utilities like conditional execution,

loops, functional abstraction, etc. When these are added on to special purpose languages late in the game, elegance is compromised by compatibility with the original specialized language and we end up with a mess.

Instead, we propose, as DAML/OIL does for static Web semantics, to gently extend the semantics of the widely accepted HTML for markup and XML for static data, to encode procedural semantics in Web pages. The extension is called Glue.

Glue is a language for the web that embodies the three primary functionalities needed for general purpose information manipulation into one unified language:

- *Markup*. Since Glue is a superset of HTML, it inherits all its capability.
- *Data*. Glue permits the description of persistent structured data on the web via XML like syntax yet having the capability of dynamically computing values that may contain self-referential interconnections.
- *Code*. Glue is a general purpose object oriented programming language that is, at its core, more flexible than Javascript, VBScript and Java.

Glue provides a way to define functions and call them in addition to defining and instantiating objects. The object system is a multiple-inheritance, prototype-based object system with annotations. Thus it is not merely a vanilla procedural/object oriented general purpose language grafted on to HTML and XML, but rather a language whose very core supports the intimate mixture of unstructured content, structured data and active values through a highly dynamic object system.

9. Is Procedural Attachment Rope to Hang Yourself?

Advocates of declarative representations always get nervous when someone advocates letting the user call a procedure from any point in a declarative representation. There is the danger that the code can have unpredictable effects, and that the reasoning systems cannot guarantee anything about the behavior of descriptions that contain embedded code. In this view, procedural attachment is giving the user rope to hang themselves.

To some extent, that's true. But we should recognize that many operations that Web applications will want to do will inevitably go beyond purely declarative representations. I/O operations, especially network I/O, user interface operations that interact in real time with the user, and side effects to shared data structures, often have this character. If we disallow procedures to be embedded in the declarative representation, we are merely pushing the procedural information outside of the declarative representation entirely. While this might gain some cleanliness in the declarative part, this doesn't alter whatever properties of unpredictability the entire system, procedures and data, has as a whole. And, after all, that's what we're really interested in.

9.1. Procedural Markup

A better approach is to allow procedures to be embedded in data, accepting the risk, but to also encourage the programmer to do what we might call *procedural markup*, analogous to markup for text. Markup for text [e.g. <b>] supplies additional declarative annotation that serves as advice to the renderer about how to display the text. Markup for procedures is to include with the embedded procedures a declarative representation of the result of the procedure, so that the reasoning system can operate on it. For example, a procedure that computes the price of an airline ticket might assert that the result is a positive number in US dollars.

Of course, the results of the reasoning will only be valid if the markup accurately describes the result of the procedure. Although it is impossible in general to predict the result of an arbitrary program, in some cases, automatic program verification might actually be able to either generate the assertions automatically or prove that the code actually satisfies the assertions. To enable this, it is best to keep the code close at hand to the places where it is used. And for debugging, it is invaluable to have a seamless interface that integrates procedures and data so the programmer can see if there is a mismatch between what the program produced and what he or she is expecting.

10. Glue's Provides a Smooth Path From Text to Code

Like HTML, Glue permits you to just write a paragraph of English text and call it a web page. Also like HTML, you can take another pass and add markup without having to rewrite your original text. You need only learn the HTML tags that you need to know.

However, should you require more dynamic behavior, with Glue you needn't learn a new scripting syntax such as Javascript, or worse, a whole new programming environment such as Java. You can simply insert Glue tags into your page using the same basic syntax that your markup is in. Plain text, markup, data, and code can be freely intermixed with no barriers between those functionalities and no "impedance mismatches". Glue tries hard to give the user the most functionality per learning effort, emphasizing economy and elegance at the expense of hairy and rarely used features.

10.1. Who's the user?

The target audience for Glue is the web site author who needs more than HTML can offer yet is intimidated (and rightly so) by multi-language programs. Although Glue can fulfill most of the needs of the general purpose programmer, someone who's spent years learning a difficult language like C or Java is hard to convince to give up on their investment even if they can exceed their previous productivity in a few weeks. The HTML author looking for more flexibility is an easier sell. And there are millions of them. But the suitability of Glue is not limited to toy programs. Glue is not simplified at the expense of being able to accommodate complex programming tasks.

11. Glue's Distinguishing Features

Glue has a unique approach to bridging the apparent gap between static and dynamic semantics by unifying what in other languages are objects and function calls. Yet it doesn't start off by throwing today's web author in cold water either.

- *Minimal differences between markup, data, and code.* We have already discussed this point.
- *Minimal differences between classes and instances.* Glue's object system uses prototypes instead of classes [Lieberman, 1986]. No need to learn separate operations for classes and instances since there's no distinction between a class and an instance. An object can be used "as an instance", and/or it can be "subclassed" with the new object inheriting the desired characteristics of its parent(s) while other characteristics are modified to differentiate the new object from its parent(s).
- The language Self [Ungar & Smith, 1987] had a simple and elegant prototype-based object system. Self's object system differs from Glue in that it was only single-inheritance and had a "copy down" semantics for inherited values rather than the "dynamic look-up" mechanism of Glue. The dynamic lookup of Glue makes object creation faster and uses less memory than "copy-down". It also preserves locality of reference for shared data which may have additional speed advantages. But if "copy-down" is desired, the "init" function for

object creation of a given parent object can choose to copy down desired fields.

- *Minimal difference between "instance variables" and "methods".* Each are stored as values of fields in an object. You can get the value of an instance variable or a method by just getting the value of a named field. Methods are implemented by objects just like everything else in the language. To call a method, you use the function calling syntax which looks like using an HTML tag.
- *Minimal differences between a function call and object creation.* The tag name refers to the parent object. The parameters after the tag name serve as either field initializers or function arguments depending on the value of the tag.
- *Minimal differences between aggregate data types.* Objects, vectors and hash tables are all represented by the same data structure. An object is a set of fields whose keys can be any object [interned strings, numbers, other objects]. When you're treating an object as a vector, a looping mechanism permits looping over all fields whose keys are integers. A "size" field makes finding out the length of the vector quick as well as facilitating "adding an element to the end of the vector".
- *No differences between the object system and the type system.* For example, "integer" can be thought of as a "type"

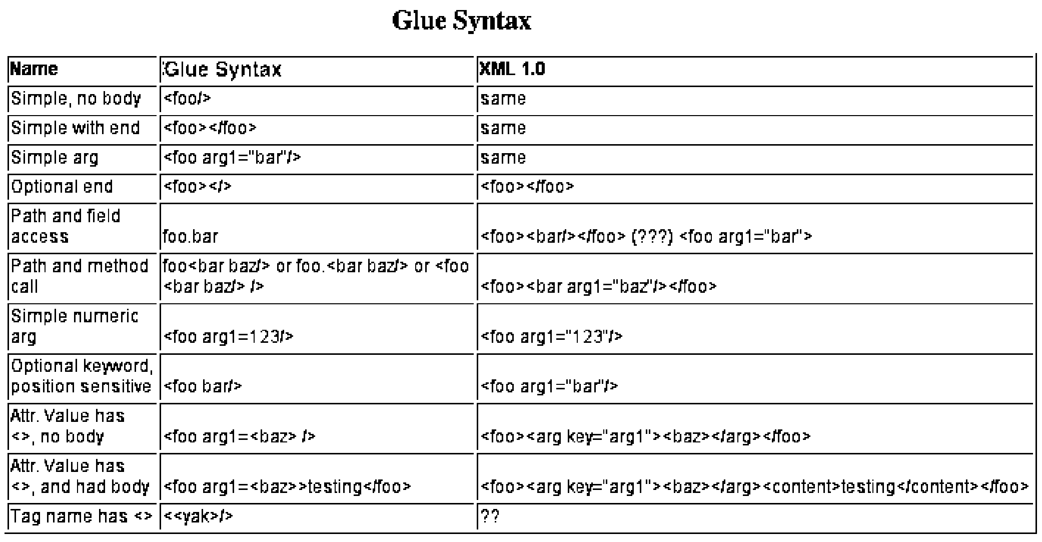
but its really just another object in the object tree that happens to be the parent object of all integers. You can reference all objects used as types via the path syntax, just like you can reference all fields of objects. In fact, since any object can become a parent, any object can effectively be a type.

- *Minimal differences between initialization and normal methods.* An *init* method can, in fact, return any type of object and should declare the returned type of object.
- *Minimal [but very significant] differences between local variables, fields, and subobjects* for creating, setting and referencing.
- *Minimal differences between a field and an annotation to that field.* Glue provides a way to add information ABOUT a field. Say you have a field named "color" in an object named "car". You might want to add a comment about the field such as "all colors besides black and white cost extra" or declare that the type of the field must be an integer. Annotations are the mechanism that Glue uses to associate information about a field with that field. Annotations of a field "foo" are indicated by another field with a naming convention of "foo@annotation-name". A looping mechanism makes it easy to find all annotations of a given field or to ignore all annotations of an object when you want to loop through an object's field values.

- *Glue provides an easy way to specify the "evaluation kind"* of each parameter in a method definition. This controls how each argument in a call to the method is interpreted. It can be treated as code and evaluated, which is the default. It can be treated as an expression to be parsed but not evaluated, which is especially good for "delayed evaluation" where a method takes code as an argument and can evaluate the code when it chooses. It can be treated as the string of its source code (and not even parsed), and hypertext, which treats text within angle brackets as code to evaluate and other text just as strings (especially convenient for implementing HTML tags as Glue objects or method calls).
- *Glue permits the parent of an object to be changed at runtime.* This is not a common situation, but makes it easy for a parent to "adopt" a child.
- *Glue allows optional keywords.* In HTML you usually use keywords to specify each attribute. XML requires the use of keywords. In Java and Javascript you can't use keywords to specify any argument. Glue permits you to use keywords when you want to be more explicit or pass arguments "out of order" and pass arguments by position (without keywords) when you want to be more concise. There is no extra overhead for declaring keywords when defining a method, they are simply the parameter's name.

- *Glue permits "Active Values" on object fields.* These facilitate simple constraint systems, and, along with other dynamic features, help to implement specification changes after most of the implementation is done by permitting field references and setters to turn into method calls without changing the referencing and setting code at all.

Glue is especially easy to write interactive programming environment tools for, due to its extensive introspection capabilities, evaluation kinds, elegant object system, uniform syntax, ultra-dynamic behavior and small size. Below, we present a summary of Glue syntax, for those interested in the details. An example of Glue code follows.



Glue Syntax

Name	Glue Syntax	XML 1.0
Simple, no body	<foo/>	same
Simple with end	<foo></foo>	same
Simple arg	<foo arg1="bar"/>	same
Optional end	<foo></>	<foo></foo>
Path and field access	foo.bar	<foo><bar/></foo> (???) <foo arg1="bar">
Path and method call	foo<bar baz/> or foo.<bar baz/> or <foo <bar baz/> />	<foo><bar arg1="baz"/></foo>
Simple numeric arg	<foo arg1=123/>	<foo arg1="123"/>
Optional keyword, position sensitive	<foo bar/>	<foo arg1="bar"/>
Attr. Value has <>, no body	<foo arg1=<baz> />	<foo><arg key="arg1"><baz></arg></foo>
Attr. Value has <>, and had body	<foo arg1=<baz>>testing</foo>	<foo><arg key="arg1"><baz></arg><content>testing</content></foo>
Tag name has <>	<<yak>/>	??

Figure 13.4 Glue syntax table.

12. A Glue Example

Glue permits arbitrary code intermixed with HTML. Here's a snippet of ordinary HTML:

```
<font size="3">Hello World</font>
```

With Glue we can stick code anywhere within this such as an attribute value:

```
<font size=1.<random 7/> > Hello World</font>
```

Glue really has two syntaxes, a pure XML syntax and one that permits some short cuts such as "by position arguments" which makes using attribute names unnecessary. Both syntaxes can be used in the same body of code.

Here's a more complex example of generating a catalogue for a store. First we set up our inventory "database". This might come from another file or another method call but for simplicity we enter it directly as a vector of vectors like so:

```
<set root.inventory
  <vector
   <vector "Flannel Top Sheet"  "17.95"/>
   <vector "Satin Pillow Case"  "11.45"/> /> />


<set page
     <p>For our everyday unbeatable price we have
     </p> />


<set page <p>For our everyday unbeatable price we have</p>
/>


root.inventory.<for_each>
```

```
            page.content.<push field_value.0/>
            page.content.<push " for the low low price of "/>
            page.content.<push field_value.1/>
            page.content.<push <br/> />
      </for_each>
```

At the end of running this program the "page" variable contains our page object. We can get the string of HTML like so:

```
      page.<to_html/>
```

Let's add some variability based on the day of the month:

```
      <set root.is_sale_day <date/>.day_of_month.<same 1/> />
```

Now is_sale_day is true if its the first day of the month and false otherwise.

```
      <set page <if root.is_sale_day <p>On Sale today we have</p>
                   true <p>For our everyday unbeatable price we have</p>
                </if> />
```

The heading to our page is now customized, so let's do the same for the body:

```
      root.items.<for_each>
            <set price
             field_value.1.<times <if> root.is_sale_day .75
                                   true 1
                                   </if>
                        />
```

```
          />
          page.content.<push field_value.0/>
                             " for the low low price of "/>
                             price
                             <br/>
                      />
     </for_each>
```

Above the price will be multiplied by .75 if we're on the first of the month.

13. Comparison with Java

Some of Glue's features provide fixes for inconsistencies and other unnecessary complexities in Java [Lemay & Cadenhead, 2001].

- Unlike Java, all Glue data types are full-fledged objects. There are no "ints" and "Integers", just "integers".
- Unlike Java, Glue has true multiple inheritance. There is no extra "interface" mechanism.
- Unlike Java, Glue supports both a prefix syntax and an infix syntax -automatically. Glue does not need precedence rules or parentheses.
- Unlike Java, the types of elements in vectors and hash tables can be declared. There is no need for casting when you extract an element from a vector or a hash table.

- Unlike Java, the types of fields and method return values need not be declared. They default to the most general type of object. However, type declarations are encouraged as they help declare the programmer's intent and can be used by programs to find inconsistencies in the programmer's intent as well as speedup execution.
- Unlike Java, the "init" method is just like a regular method. Its body returns a value [usually but not necessarily a subobject of the called object]. It is named and has a return type declaration just like a regular method.
- Unlike Java, instance variables can be created or removed within an object at run time.
- Unlike Java, you can add methods to a system class (or any other), extending its behavior without having to edit and recompile the original source. Glue gives the programmer more flexibility in modularity so that a method for a class need not reside in the same file as the class definition if that's what the programmer chooses.
- Unlike Java, Glue's evaluation kinds permit the easy construction of high-level code manipulation methods which in many cases reduce the complexity of packaging up and calling advanced functionality.
- Unlike Java, there is rarely a need to create a "singleton", ie a class with one instance object. That's because in Glue,

a "class" object can be treated just like an instance object. You can call its methods directly using the "class" as "this". There is no need to have a distinction between static variables and instance variables, nor static and instance methods. And each "instance" object in Glue can get its own special version of method, if the programmer so chooses, to distinguish its behavior from that of its sibling objects. "Static" is another concept you just don't have to know about in Glue.

- Compared to Java, Glue is much more consistent with itself. For example, Java has three different syntaxes for programmatically determining the length of an array, a vector and a string. Glue has one.

14. Comparison with Javascript

Glue has a lot in common with Javascript [Goodman, 2000]:

- Glue and Javascript can be embedded in HTML.
- Glue and Javascript are interpreted.
- Glue and Javascript both have "eval" though Javascript doesn't have a "parsed but not evaled" representation using for programs that manipulate code.
- Glue and Javascript objects can both have fields added dynamically, after an object is created.
- But there are a lot of differences as well:

- Glue can be "more" embedded than Javascript within HTML. Generally speaking, Javascript can only be attached to certain control attributes of Javascript tags. Glue permits active code just about anywhere within HTML.
- Glue's syntax is much more like HTML than Javascript's.
- The object system in Javascript is designed to be a "prototype" object system. Yet it is, at the very best, poorly documented. Glue's prototype object system is easier to use.
- Glue permits (but does not force) the declaration of types for arguments and object fields. Javascript does not permit type declarations.
- Glue has much more flexible argument definitions which can take default values and "evaluation kinds" in addition to names and types.
- Glue has a simple, concise syntax for referencing objects through long paths in the object hierarchy and interconnected field values.

Conclusion

We are now at a crossroads in the evolution of the Web. The Web has evolved from a relatively static collection of pages and links, to a dynamic, interactive interface to semantic information. We are at the verge of being able to create the Semantic Web, in terms of declaratively representing objects that are already human-readable on the Web. Next, we need to make it the Dynamic Semantic Web by encoding procedures in Web material as first-class objects.

We've presented an argument for dynamic semantics, along with a language, Glue, that integrates procedures seamlessly into Web pages, just as XML and DAML/OIL integrate descriptions. Some may think we place too much emphasis on the language. It is true that good environments can be built on top of mediocre languages, if you dedicate a great deal of effort. The current Web itself may be viewed an example of that. But great environments need a language that supports the easy construction of dynamic, interactive and introspective tools. The difference between good environments and great environments is tens or even hundreds of percents in speed of implementing reliable, maintainable programs. Increasing productivity of Web applications is the ultimate goal. We can do this not just by handing existing programmers more powerful tools, but by giving people who consider themselves non-programmers, as many of today's HTML authors do, the power to radically customize their computer.

References

[Bauer, Dengler & Paul, 2001] Mathias Bauer, Dietmar Dengler, and Gabriele Paul, Programming by Demonstration for Information Agents, in [Lieberman, ed. 01].

[Goodman, 2000] Danny Goodman's Javascript Handbook, IDG Books, 2000.

[Kushmerick, Weld & Doorenbos, 1997] Nicholas Kushmerick, Daniel S. Weld, Robert Doorenbos, Wrapper Induction for Information Extraction. Intl. Joint Conference on Artificial Intelligence (IJCAI), 1997.

[Lemay & Cadenhead, 2001] Laura Lemay and Rogers Cadenhead, Teach Yourself Java 2 in 21 Days, Sams Press, 2001.

[Lieberman, 1986] Henry Lieberman, Using Prototypical Objects to Implement Shared Behavior in Object-Oriented Systems , First Conference on Object-Oriented Programming Languages, Systems, and Applications [OOPSLA-86], ACM SigCHI, Portland, Oregon, September 1986.

[Lieberman, ed., 2001] Henry Lieberman, ed., Your Wish is My Command: Programming by Example, Morgan Kaufmann, San Francisco, 2001.

[Lieberman, Fry & Weitzman, 2001] Why Surf Alone? Exploring the Web with Reconnaissance Agents, Communications of the ACM, to appear, 2001.

[Ungar & Smith, 1987] Self: The Power of Simplicity. ACM Conference on Object-Oriented Programming Languages, Systems, and Applications [OOPSLA-87].

[Woods, 1975] Woods, W. (1975), What's in a Link: Foundations for Semantic Networks, in D.G. Bobrow & A. Collins (eds.), Representation and Understanding, Academic Press.

Footnotes

[1]Bowstreet, Inc., Lynnfield, MA, USA

[2]Glueworks, Wellesley, MA, USA

[3]Media Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA

14. Semantic Annotation for Web Content Adaptation

Masahiro Hori[1]

1. Introduction

Web contents are becoming accessible from a wide range of mobile devices such as cellular phones, pagers, and palmtop computers. Since these devices do not have the same rendering capabilities as desktop computers, it is necessary for Web contents to be adapted for transparent access from a variety of client agents. For example, a large full-color image may be reduced with regard to size and color depth, removing unimportant portions of the content. Such content adaptation is exploited for either an individual element or a set of consecutive elements in a Web document, and results in better presentation and faster delivery to the client device [Bickmore and Schilit 1997; Fox and Brewer 1996; Rousseau et al. 1999]. Transformation of information from one form to another is often called *transcoding*, and it is particularly called Web-content transcoding when the transformation is conducted along the Web transaction path. The Web-content transcoding is a way of adding value to Web contents, and facilitates re-purposing the Web contents. The transcoding is thus crucial for universal Web access under different conditions, which may depend on client capabilities, network connectivity, or user preferences.

Markup languages such as HTML embed annotations into documents. For example, an <ol> tag indicates the start of an ordered list, and a paragraph begins with a <p> tag. However, annotations can be external, residing in a file separate from the original document. It seems be impractical to incorporate the data about Web documents, namely, Web metadata into existing HTML documents, because of the difficulty of changing the established HTML specification to meet the application-specific annotation needs. Although making annotations external may require additional bookkeeping tasks, it has the substantial advantage of not requiring any modification of existing Web documents. More importantly, the external annotation facilitates re-purposing Web documents through the reuse of annotations that can be shared across Web documents. External annotation is thus a promising approach to facilitating the Web-content transcoding.

It is important to note here that the result of applying an annotation to a Web document depends on a transcoding policy. Annotations provide additional information about Web contents, so that an adaptation engine can make better decisions on the content re-purposing. The role of annotations is to provide explicit semantics that can be understood by a content adaptation engine. Although Web-content annotation or metadata has a variety of potential applications [Lassila 1998], they can be categorized into three types: discovery, qualification, and adaptation of Web contents. The primary focus of this article is on the Web-content adaptation, and would not necessarily address the other kinds of applications such as accurate searches of Web resources (i.e., discovery) and descriptions of users' preferences regarding privacy (i.e., qualification).

In pursuit of Web content re-purposing, transcoding systems that adopt the external annotation approach are emerging [Hori et al. 2000a; WTP Developer's Guide 2000; Nagao et al. 2001; Asakawa and Takagi 2001). In the remainder of this article, a framework of external annotation is introduced first, and a high-level overview of annotation-based transcoding is explained with emphasis on the idea of authoring-time transcoding. HTML-page splitting is then taken as an example of Web-content adaptation, and the results of applying the transcoding module to real-life Web contents are investigated. Finally, a preliminary empirical evaluation is presented in the case of the HTML-page splitting plug-in, and the annotation-based transcoding approach is compared with related works.

2. External Annotation Framework

A framework of external annotation prescribes a scheme for representing annotation files and a way of associating original documents with external annotations. The role of annotation is to characterize ways of content adaptation rather than to describe individual contents themselves. The basic ideas behind this annotation framework are twofold. One is that new elements and/or attributes should not be introduced into a document-type definition of documents to be annotated. The other is that annotations need to be created for arbitrary parts of annotated documents.

Annotations can be embedded into a Web document as an inline annotation. Inline annotations are often created as extra attribute of document elements. Most of existing HTML browsers ignore unknown attributes added to HTML elements, without being bothered by the proprietary inline annotations. Because of its simplicity, inline annotation has been often adopted as a way of associating annotation to HTML documents [Mea et al. 1996; Rousseau et al. 1999; Erdmann et al. 2000; Heflin and Hendler 2000]. An advantage of the inline annotation approach is the ease of annotation maintenance without the bookkeeping task of associating annotations with their target document. The inline approach, however, requires annotators to have document ownership, because annotated documents need to be modified whenever inline annotations are created or revised.

The external annotation approach, on the other hand, does not suffer from the issues related to the document ownership. Moreover, the most important point of the external annotation is that this approach facilitates the sharing and reuse of annotations across Web documents. Since an external annotation points to a portion of a Web document, the annotation can be shared across Web documents that have the same document fragment. Furthermore, mixing of contents and metadata would not be desirable according to the design consideration that separates content from presentation [Style Sheets 2000).

Figure 14.1 depicts paths from a Web document to different client devices. When a Web document is provided with an annotation document, a transcoding proxy may adapt the document on the basis of associated annotations upon receiving a request from a personal device (Figure 14.1 (b)). When a Web document can be rendered as in the case of HTML documents, the document must be viewable in a normal browser on a desktop computer even if it is associated with an annotation document (Figure 14.1 (a)).

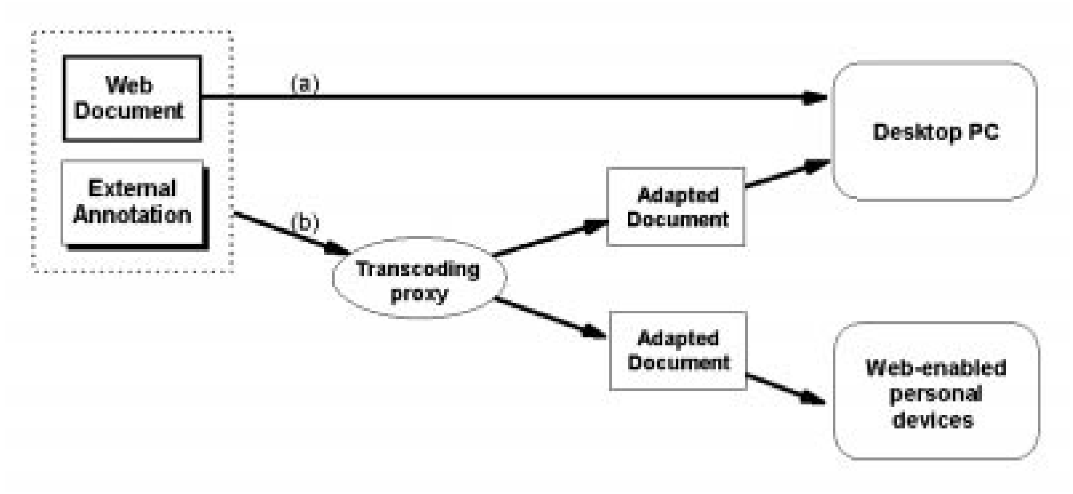
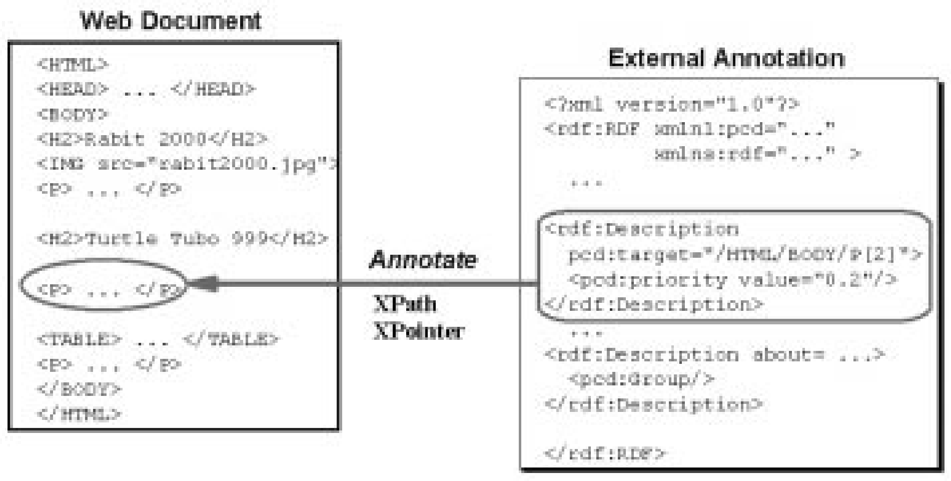


Figure 14.1 Overview of an annotation-based transcoding.

External annotation files contain metadata that addresses a part of a document to be annotated. [Xpath, 1999] and

[Xpointer, 2001] are used to associate annotated portions of a document with annotating descriptions. Figure 14.2 illustrates a way of associating an external description with a portion of an existing document. An annotation file refers to portions of an annotated document. A reference may point to a single element (e.g., an IMG element), or a range of elements (e.g., an H2 element and the following paragraphs). For example, /HTML/BODY/P[3] points to the third P element of the BODY element of the annotated document. If a target element has an id attribute, the attribute can be used for direct addressing without the need for a long path expression. While XPath expressions basically selects a node or subtree of a target document tree, XPointer could point to a sub-string text contents, or a range of consecutive nodes that may not necessarily constitute a subtree.



```
<HTML>
<HEAD> ... </HEAD>
<BODY>
<H2>Rabit 2000</H2>
<IMG src="rabit2000.jpg">
<P> ... </P>

<H2>Turtle Tubo 999</H2>

<P> ... </P>

<TABLE> ... </TABLE>
<P> ... </P>
</BODY>
</HTML>
```

```
<?xml version="1.0"?>
<rdf:RDF xmlnl:pcd="..."
         xmlns:rdf="..." >
  ...

<rdf:Description
  pcd:target="/HTML/BODY/P[2]">
  <pcd:priority value="0.2"/>
</rdf:Description>
  ...
<rdf:Description about= ...>
  <pcd:Group/>
</rdf:Description>

</rdf:RDF>
```

Figure 14.2 Framework of external annotation

As the fundamental syntax of annotation files, the Resource Description Framework [RDF, 1999] can be used. An annotation file, which is an XML document, therefore consists of a set of RDF descriptions. The RDF data model defines a simple model for describing relations among resources in terms of named properties and values. In addition, it is necessary to deal with user preferences and device capabilities for the content adaptation. Such information profiles can be described by using the specification of Composite Capability/Preference Profiles, which is often abbreviated as CC/PP [CCPP, 2000]. The CC/PP specifies client-side profiles can be delivered to a proxy server over HTTP [CCPP-exchange, 1999]. Furthermore, it is currently investigated ways of describing document profiles so that requirements for desired rendering can be clarified [XHTML-prof, 1999], and RDF is being employed for encoding the conformance profiles. Taking account of the situations of these standardization activities, it is reasonable for annotation vocabularies to be encoded in RDF, so that comprehensive content adaptation mechanisms can be pursued in accordance with open standards.

When annotation files are stored in a repository, an appropriate annotation file for a Web document needs to be selected dynamically from the repository either implicitly by means of a structural analysis of the subject document or explicitly by means of a reference contained in the subject document or some other association database. The explicit association, for example, can be done by using <link> tag [HTML, 1999] or [Xlink, 2000]. An annotation file can be associated with a single document file, but the relation is not limited to one-to-one. It is possible for multiple annotation files to be associated with a single document file, when each annotation file contains descriptions related to different portions of an annotated document. On the other hand, a single annotation file may contain an annotation description to be shared among multiple document files. This type of annotation would be useful when it is necessary to annotate common parts of Web documents, such as a page header, a company logo image, and a side bar menu.

3. Annotation-Based Transcoding System

Since content adaptation can be done on either a content server, a proxy, or a client terminal, an adaptation engine should not be forced to reside in any particular location. In order to resolve this limitation, a proxy-based approach has been adopted for content adaptation [Fox and Brewer 1996; Schickler et al. 1996; Barrett and Maglio 1998; WTP 2001; Britton et al. 2001]. Computational entities stay along the Web transaction path are called intermediaries [Barrett and Maglio 1999], and existing approaches to annotation systems confirm to a common abstract architecture based on intermediaries [Vasudevan and Palmer, 1999].

3.1. Transcoding Architecture

As shown in Figure 14.3, intermediaries are entities that reside along the path of information streams, and facilitate an approach to making ordinary information streams into smart streams that enhance the quality of communication [Barrett et al., 1999]. An intermediary processor or a transcoding proxy can operate on a document to be delivered, and transform the contents with reference to associated annotation files. From a computational perspective, the use of an intermediary architecture is an approach to providing pluggable services between a Web client and server [Thompson et al., 1999]. To put it another way, intermediaries provide special-purpose proxy servers for protocol extension, document caching, Web personalization, content adaptation, and so on [Barrett et al., 1998]. The intermediary-based approach is suitable for the realization of content adaptation by means of external annotation, because the intermediary transcoding modules do not require changing the existing HTTP-based information streams.

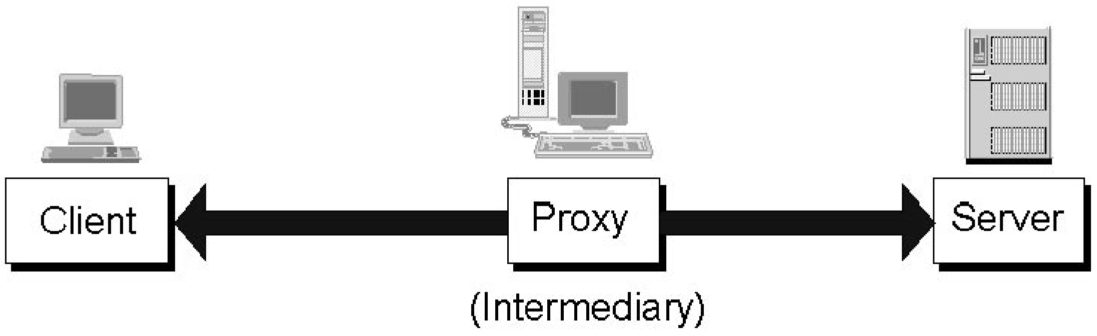


Figure 14.3 Intermediary between client and server.

3.2. Authoring-Time Transcoding

Creation and revision of annotations would not easy solely with a simple source tag editor. For example, addressing by XPath/XPointer may require to following a hierarchy of document elements from the root to a focal element for the creation of an annotation, and it is also necessary for annotator to navigate from an existing annotation to a portion of an annotated document designated by a corresponding XPath/XPointer expression. Furthermore, it is substantially helpful if the results of content adaptation can be verified through a previewer. The results of adaptation are difficult to anticipate when it transforms the structure of an original document, and often comes out with unexpected results due to inappropriate specification of the annotations. Therefore, transcoding needs to be done not only at the run time, but also at the design time on demand in the course of annotation authoring, so that the result of adaptation can be checked immediately.

The idea of authoring-time transcoding is depicted in Figure 14.4. An annotation tool consists of an editor for the creation and revision of annotations, and a viewer to show the result of applying the annotations to a target document. The editor is employed not only to edit annotation descriptions, but also to specify a portion of a target document to be annotated. When an annotator creates an annotation file from scratch, a typical scenario of the authoring-time transcoding would be as follows. A target document is opened in the editor (Figure 14.4 (1)), which allows an annotator to create annotations (Figure 14.4 (2)). The created annotations are saved as an annotation file, and then stored in an annotation repository or an HTTP server (Figure 14.4 (3)). When the viewer is invoked, a transcoding proxy is called over HTTP and the corresponding annotation is applied to a target document (Figure 14.4 (4)). An adapted document is then sent back to the viewer for the display (Figure 14.4 (5)). If the adaptation results were not desirable for the annotator, the annotation authoring can be continued to revising the annotations seamlessly in the same authoring environment.

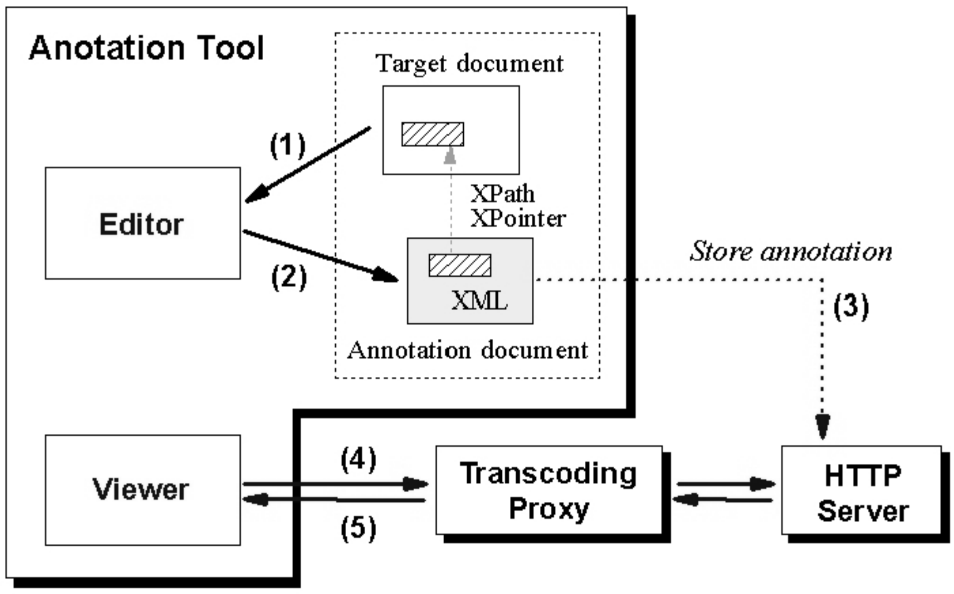


Figure 14.4 Authoring-time transcoding

The authoring-time transcoding would be crucial when annotations are employed for content adaptation, rather than discovery or qualification of contents, because the content adaptation often changes the structure of original documents as the results of transcoding. In most annotation tools for HTML annotation, it is assumed that a target HTML document is solely displayed on an HTML browser and can never be edited [Denoue and Vignollet, 2000, Erdmann et al., 2000, Nagao et al. 2001, Sakairi and Takagi 2001]. On the other hand, there is an annotation tool that is fully integrated with a WYSIWYG HTML editor [Hori et al., 2000b], and allows users to annotate an HTML document as well as editing the document to be annotated.

Configurations of annotation tools depend on the situations of annotation. The browser-based annotation tools are desirable when annotators are not allowed for editing target documents without the document ownership. On the other hand, an annotation tool based on a WYSIWYG editor is helpful, when annotators are responsible for not only the creation of annotations but also the editing of target documents. Regardless of the variety of emerging annotation tools, a significant limitation of the current annotation tools is the lack of extensibility, because the existing tools are developed solely for a particular annotation vocabulary, and provided with predefined views for the authoring. An annotation tool framework is currently being pursued in order to realize the flexibility in the annotation tool configuration [Abe and Hori, 2001].

4. HTML-Page Splitting for Small-Screen Devices

The annotation framework mentioned above prescribes a skeletal structure of annotation, without regard to the specifications of any annotation vocabulary and the behavior of any content adaptation module. As an example of the annotation-based transcoding, this section explains an annotation vocabulary [AWCT, 1999] and a content adaptation module [Hori et al., 2000a] that splits HTML pages into smaller fragments to be displayed on small-screen devices.

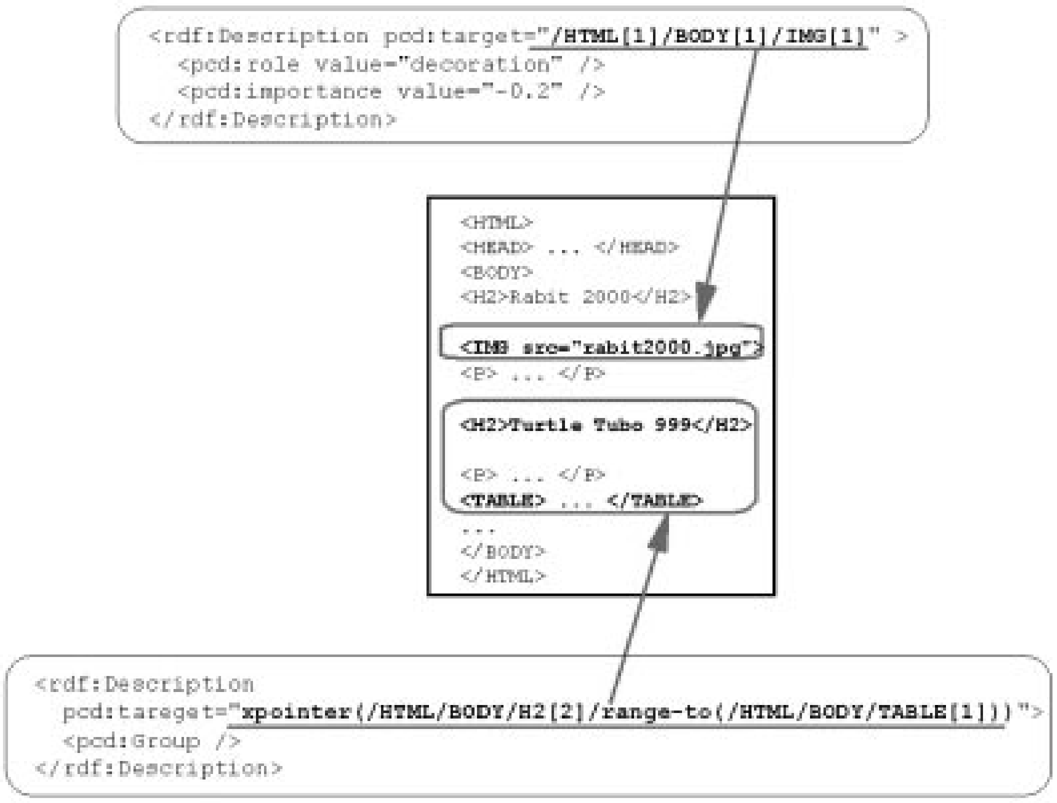
4.1. Annotation Vocabulary

Annotation vocabulary for HTML-page splitting needs to be specified for constraining the possibilities for decomposition, combination, and partial replacement of contents. The vocabulary includes three types of annotation: *alternatives*, *splitting hints* and *selection criteria*. A namespace [XML-names, 1999] prefix "pcd" is used for the transcoding vocabulary as well as "rdf" for the constructs of RDF. Further details on this annotation vocabulary can be found in another article [AWCT, 1999].

Alternatives Alternative representations of a document or any set of its elements can be provided. For example, a color image may have a grayscale image as an alternative for clients with monochrome displays. A transcoding proxy selects the one alternative that best suits the capabilities of the requested client device. Elements in the annotated document can then be altered either by replacement or by on-demand conversion.

Splitting hints. An HTML file, which can be shown as a single page on a normal desktop computer, may be divided into multiple pages on clients with smaller display screens. A *pcd:Group* element specifies a set of elements to be considered as a logical unit, and provides hints for determining appropriate page break points. For example, the annotation description in Figure 14.5(a) indicates that the range of elements from the second occurrence of an H2 element through the first occurrence of a TABLE element is annotated as a group.

(b) Role and importance annotation.



```
<rdf:Description pcd:target="/HTML[1]/BODY[1]/IMG[1]" >
  <pcd:role value="decoration" />
  <pcd:importance value="-0.2" />
</rdf:Description>
```

```
<HTML>
<HEAD> ... </HEAD>
<BODY>
<H2>Rabit 2000</H2>
<IMG src="rabit2000.jpg">
<P> ... </P>
<H2>Turtle Tubo 999</H2>

<P> ... </P>
<TABLE> ... </TABLE>
...
</BODY>
</HTML>
```

```
<rdf:Description
  pcd:taregat="xpointer(/HTML/BODY/H2[2]/range-to(/HTML/BODY/TABLE[1]))">
  <pcd:Group />
</rdf:Description>
```

(a) Group annotation for a page-breaking point.

Figure 14.5 Examples of annotation descriptions

Selection criteria An annotation may contain information to help a transcoding module select from several alternative representations the one that best suits the client device. A *pcd:role* element, for example, specifies the role of an annotated element. This role element is provided with a value attribute, which may be specified as proper content, side menu, or decoration, for example. A *pcd:importance* element specifies the priority of an annotated element relative to the other elements in the page. When the importance of an element is low, for example, it will be ignored or may be displayed in a smaller font. The importance value is a real number ranging from -1 (lowest priority) to 1 (highest priority). The default importance value is 0. For example, when an element is provided with a decoration role and a low importance value such as -0.2, the element may not be sent to a lightweight client. An example of such annotation description is given in Figure 14.5 (b).

4.2. Adaptation Engine

The HTML-page splitting module runs on an intermediary server called WBI [WBI, 2000]. WBI is a programmable processor for HTTP requests and responses. It receives an HTTP request from a client such as a Web browser, and produces an HTTP response to be returned to the client. Modules or plug-ins available at an intermediary processor controls the processing in between. WBI plug-in is constructed from three fundamental building blocks: *monitor*, *editor*, and *generator* [Barrett et al., 1998]. Monitors observe transactions without affecting them. Editors modify outgoing requests or incoming documents. Generators produce documents in response to requests.

An HTML-page splitting module was implemented as a WBI plug-in [Hori et al., 2000a]. It adapts requested documents in accordance with the capabilities of each client (Figure 14.6). Hints for the adaptation are expressed by using the annotation vocabulary mentioned above, and provided as external annotations. The execution sequence of the page splitting plug-in is briefly explained below.

Upon receipt of a request from a client device, an original HTML page is retrieved from a content server (Figure 14.6 (1)), and parsed into an internal data structure (Figure 14.6 (2)). The editor component of the plug-in then tries to find out the locations of annotation files. The editor component first check if a <link> tag is given in the HTML document, and tries to lookup a database of mapping from an URL of the HTML page to URLs of annotations. Annotation files are then retrieved from an annotation repository with reference to the URLs (Figure 14.6 (3)). The internal structure of the HTML page is elaborated incorporating the retrieved annotation descriptions. Note that if no URL of annotation files is found, the HTML page is returned as it is, and the session is terminated.

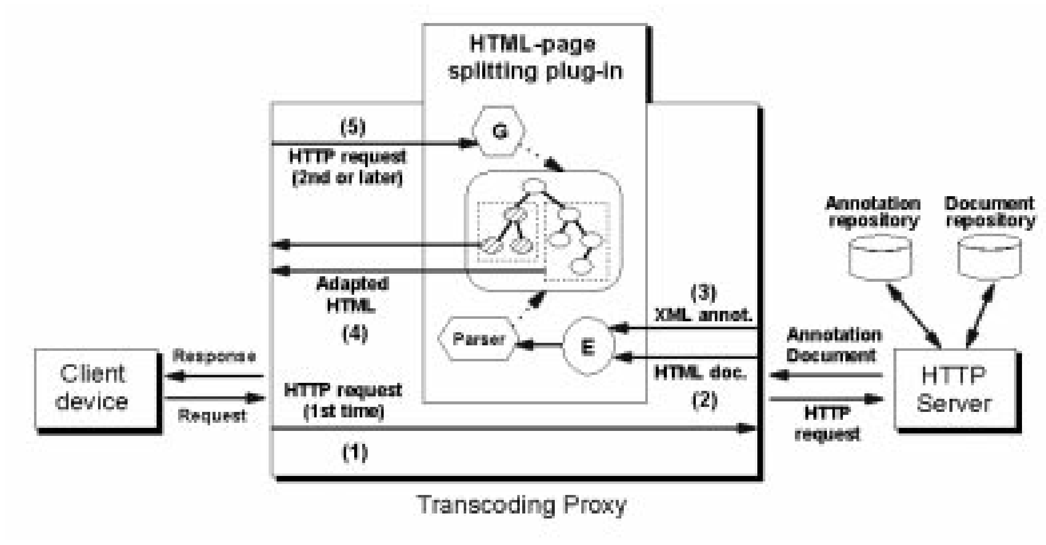
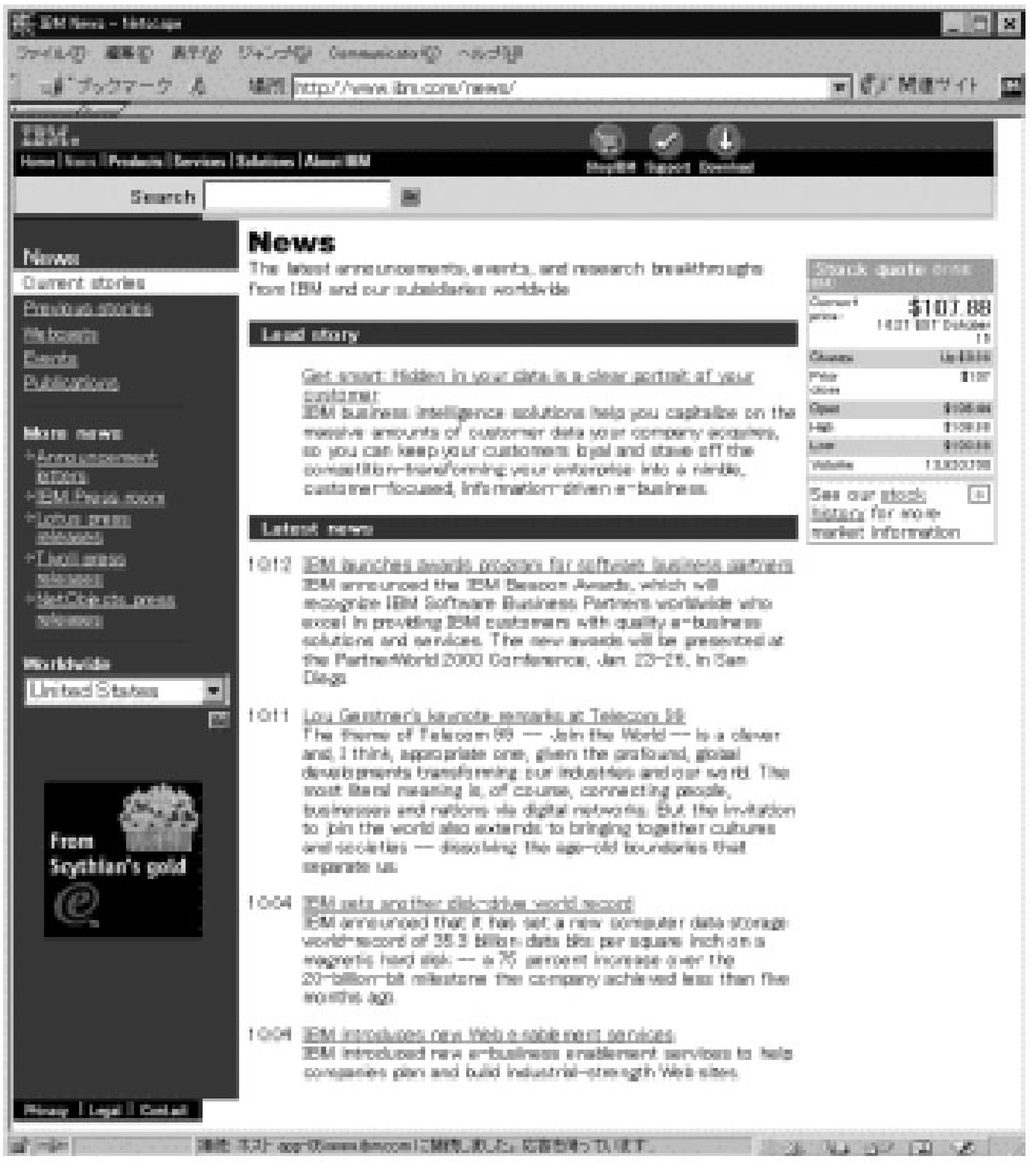


Figure 14.6 Annotation-based transcoding by a page splitting plug-in

Taking account of client profile data included in the HTML request header, the generator component selects a portion of the document structure. The generator then creates and returns an adapted page that is split off as a separate page from the original page (Figure 14.6 (4)). When another fragment of the original page is requested, the generator component of the plug-in is activated (Figure 14.6 (5)). The generator examines the requested URL, and determines which part of the original page is to be sent back. A split-off page is then created and returned to the client. Note here that each anchor element linking to another fragment of the same page must have an "href" attribute given with a special URL, which is created by the page splitting plug-in at run time. In the current implementation, such URLs include host names for the generator component and a session identifier for specifying the original page.

4.3. Application to real-life HTML pages

This section shows the results of applying the page splitting plug-in to real-life Web documents. The Web page used as an example is a news page from a corporate Web site (Figure 14.7). Use of tables for page layout is inappropriate not only as regards a clear distinction between style and content, but also as regards Web content accessibility. According to the accessibility guidelines [WCAG, 1999], content developers are encouraged to make contents navigable. In reality, however, there are a large number of HTML pages in which table elements are employed for layouting. The news page in Figure 14.7 consists of three tables stacked from top to bottom as depicted in the left of Figure 14.8. The top and middle tables correspond respectively to a header menu and a search form. The bottom table, which is labeled as "Layouter (3)" in Figure 14.8, is used for layouting.



IBM News - Netscape

http://www.ibm.com/news/

Home | News | Products | Services | Solutions | About IBM

Search

News

The latest announcements, events, and research breakthroughs from IBM and our subsidiaries worldwide

Lead story

Get smart: Hidden in your data is a clear portrait of your customer

IBM business intelligence solutions help you capitalize on the massive amounts of customer data your company acquires, so you can keep your customers loyal and stave off the competition—transforming your enterprise into a nimble, customer-focused, information-driven e-business.

Latest news

10/12 IBM launches awards program for software business partners
IBM announced the IBM Beacon Awards, which will recognize IBM Software Business Partners worldwide who excel in providing IBM customers with quality e-business solutions and services. The new awards will be presented at the PartnerWorld 2000 Conference, Jan. 23–26, in San Diego.

10/11 Lou Gerstner's keynote remarks at Telecom 99
The theme of Telecom 99 — Join the World — is a clever and, I think, appropriate one, given the profound, global developments transforming our industries and our world. The most literal meaning is, of course, connecting people, businesses and nations via digital networks. But the invitation to join the world also extends to bringing together cultures and societies — dissolving the age-old boundaries that separate us.

10/04 IBM sets another disk-drive world record
IBM announced that it has set a new computer data storage world-record of 35.3 billion data bits per square inch on a magnetic hard disk — a 75 percent increase over the 20-billion-bit milestone the company achieved less than five months ago.

10/04 IBM introduces new Web enablement services
IBM introduced new e-business enablement services to help companies plan and build industrial-strength Web sites.

Figure 14.7 Layout of a real-life news page.

Figure 14.8 illustrates how the news page will be fragmented in a small display. According to the header role of the top table in the original page, the same header appears in each of the split pages. The "[Side menu]" anchor in the center is created in accordance with the auxiliary role of the vertical side bar menu in the original page. In contrast, because the importance value is "-1.00," the search form table is omitted in every split page. The main news content then starts after the "[Side menu]" anchor, and allows users to access the primary content of the page directly.

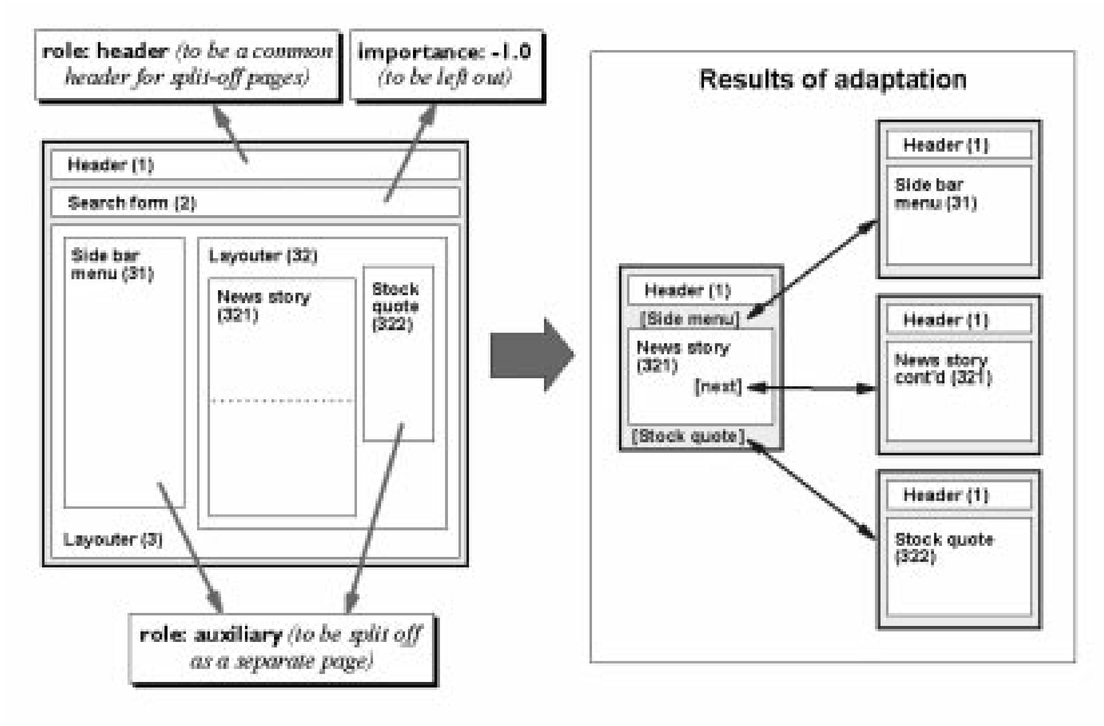
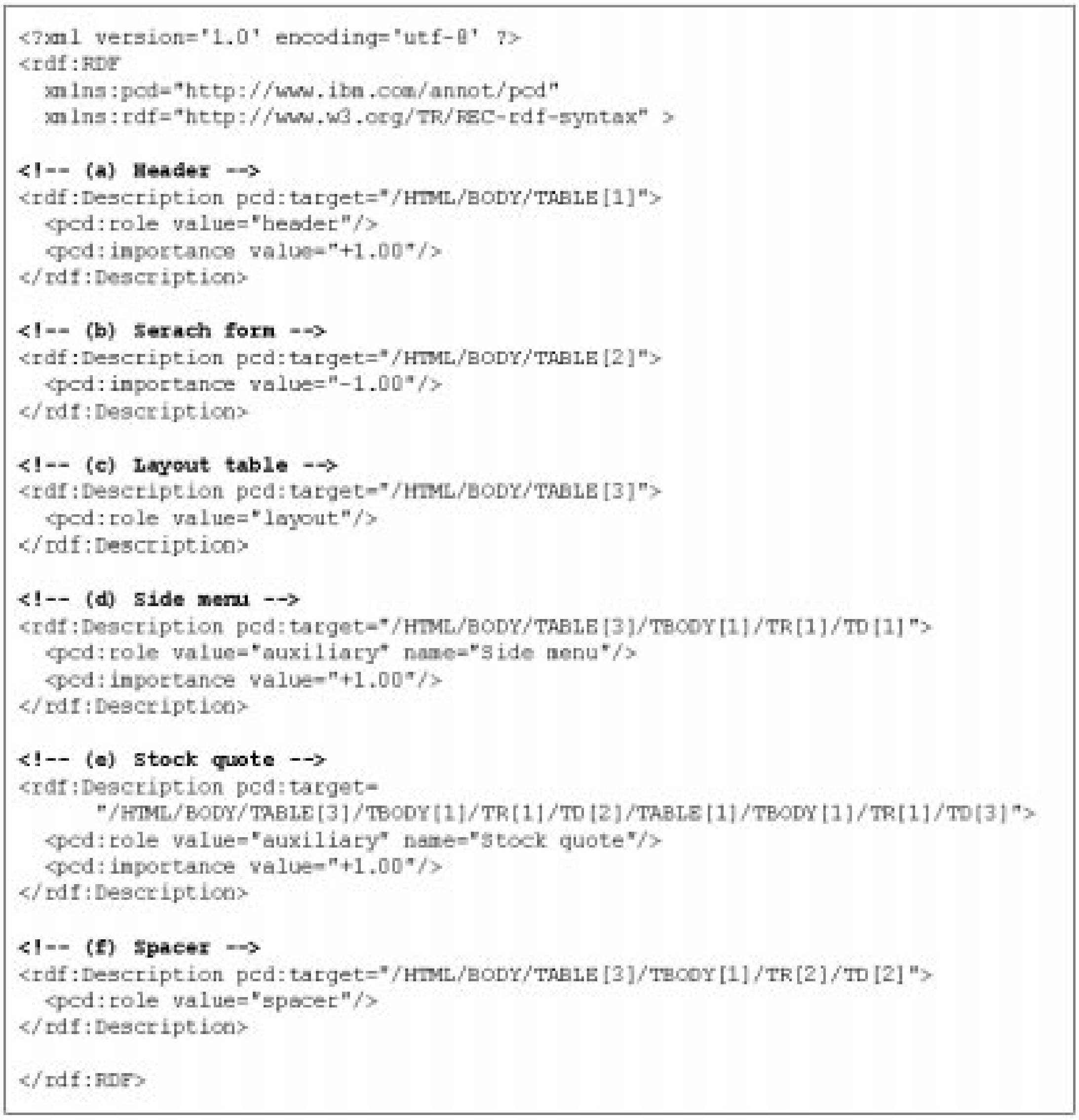


Figure 14.8 Annotation for fragmentation of an actual news page

Figure 4.9 shows annotations to be associated with the news page mentioned above. The annotation contains RDF descriptions specifying the roles of the tables. For example, the first description in the figure is about the top table, which is indicated by "Header (1)" in the left of Figure 14.8. According to the header role annotation, the page splitting plug-in adds the table element as a header of every split page. In addition, the importance value "+1.00" indicates that this table element should not be omitted in any case. The fourth description (Figure 4.9 (d)) concerns the left menu bar ("Side bar menu (31)" in Figure 14.8). Since the role of this element is annotated as auxiliary, this portion of the news page will be presented as a separate page upon receipt of a request from a small-screen device. On the other hand, the role of the bottom table in the news page ("Layouter (3)" in Figure 4.9) is annotated as layout (Figure 4.9 (c)). Therefore, the bottom table will not be retained in the display for small-screen devices. The last description (in Figure 14.9(f)) annotates a table cell that is used solely for embedding a space for adjusting the page layout. The spacer role given to the table cell helps the page splitting plug-in to decide removing the cell in small screen devices. In this way, annotations provide content semantics explicitly, which may not necessarily be given in the original documents.



```
<?xml version='1.0' encoding='utf-8' ?>
<rdf:RDF
  xmlns:pcd="http://www.ibm.com/annot/pcd"
  xmlns:rdf="http://www.w3.org/TR/REC-rdf-syntax" >

<!-- (a) Header -->
<rdf:Description pcd:target="/HTML/BODY/TABLE[1]">
  <pcd:role value="header"/>
  <pcd:importance value="+1.00"/>
</rdf:Description>

<!-- (b) Serach form -->
<rdf:Description pcd:target="/HTML/BODY/TABLE[2]">
  <pcd:importance value="-1.00"/>
</rdf:Description>

<!-- (c) Layout table -->
<rdf:Description pcd:target="/HTML/BODY/TABLE[3]">
  <pcd:role value="layout"/>
</rdf:Description>

<!-- (d) Side menu -->
<rdf:Description pcd:target="/HTML/BODY/TABLE[3]/TBODY[1]/TR[1]/TD[1]">
  <pcd:role value="auxiliary" name="Side menu"/>
  <pcd:importance value="+1.00"/>
</rdf:Description>

<!-- (e) Stock quote -->
<rdf:Description pcd:target=
      "/HTML/BODY/TABLE[3]/TBODY[1]/TR[1]/TD[2]/TABLE[1]/TBODY[1]/TR[1]/TD[3]">
  <pcd:role value="auxiliary" name="Stock quote"/>
  <pcd:importance value="+1.00"/>
</rdf:Description>

<!-- (f) Spacer -->
<rdf:Description pcd:target="/HTML/BODY/TABLE[3]/TBODY[1]/TR[2]/TD[2]">
  <pcd:role value="spacer"/>
</rdf:Description>

</rdf:RDF>
```

Figure 14.9 Annotations for splitting the news page.

Figure 4.10 shows the contents displayed on an HTML browser in a small screen device. Figure 4.10 (a) shows the news page without content adaptation, presenting only the top one-ninth of the original content. In contrast, Figure 4.10 (b) shows the result of adaptation by the page splitting plug-in. It is important to note here that the page splitting not only reduces the content to be delivered, but also places the primary content near the top of the fragmented page that is provides with navigational features. This result of adaptation follows the design guidelines for reducing scrolling during interaction with small screens: *placing navigational features (menu bars etc.) near the top of pages*, *placing key information at the top of pages*, and *reducing the amount of information on the page* [Jones et al,. 1999, p. 58].

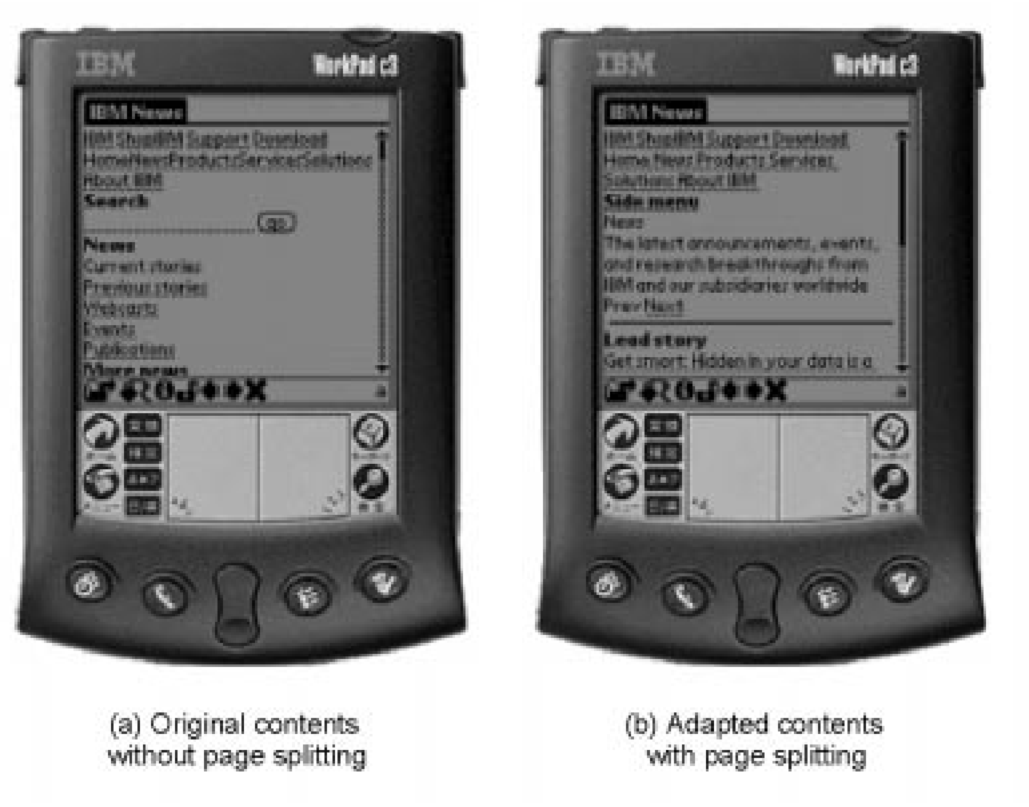

IBM
WorkPad c3
IBM News
IBM ShopIBM Support Download
HomeNewsProductsServicesSolutions
About IBM
Search
go
News
Current stories
Previous stories
Webcasts
Events
Publications
More news
(a) Original contents without page splitting
IBM
WorkPad c3
IBM News
IBM ShopIBM Support Download
Home News Products Services
Solutions About IBM
Side menu
News
The latest announcements, events, and research breakthroughs from IBM and our subsidiaries worldwide
Prev Next
Lead story
Get smart: Hidden in your data is a
(b) Adapted contents with page splitting

Figure 14.10 Comparison of display on a small-screen device

Small screens force users to employ frequent scrolling activities that may affect the accessibility of contents as well as the usability of devices. It has been reported that users with small screens were 50% less effective than those with large screens in completing retrieval tasks [Jones et al., 1999]. Therefore, page fragmentation based on the semantic annotation will be more appropriate than page transformation done by solely syntactic information, such as removing white spaces, shrinking or removing images, and so on. Semantic rearrangement is one of the critical limitations of the syntactic transformation approach. The navigational features achieved by this semantic annotation are noteworthy from the perspective of Web content accessibility. Important point here is that the semantic annotation help the page splitting plug-in make better decision on the content adaptation.

5. Discussion

5.1. Empirical Evaluation

Real-life Web documents cannot be the same for the long time, but are updated and changed occasionally. Therefore, the feasibility of the external annotation approach relies on the robustness of an addressing expression annotating a portion of a target document. Such addressing expressions, which are described with XPath in this article, may encounter two types of problems due to the updates of an annotating document. One is the case when an addressing expression points no document element to be annotated. Another case may happen when an expression is forced to point an incorrect element as the result of modification of the target document structure. The latter case would be difficult to detect, because the proper semantics of the original document are not represented explicitly. The lack of the explicit semantics is the motivation of the semantic annotations. The integrity of addressing location in the second type is an important issue that requires further research, and beyond the scope of this article. The first type of the addressing problem is taken into account of the evaluation here.

Robustness of the annotations for the news page (Figure 14.9) is investigated with regard to the gradual changes of the news page, considering the existence of pointing node in the target document tree. Figure 14.11 shows a portion of the document tree of the original news page, with indication of the nodes pointed by the six annotation descriptions in Figure 14.9. The minimum depth of the annotated nodes is 2 ((a), (b), and (c) in Figure 14.11), while the maximum depth of the annotated node is 9 (Figure 14.11(e)). The news page of the same URL has been saved as an HTML file once a day for 60 days. The depth of the saved document trees was 22, and did not changed throughout the 60 days.

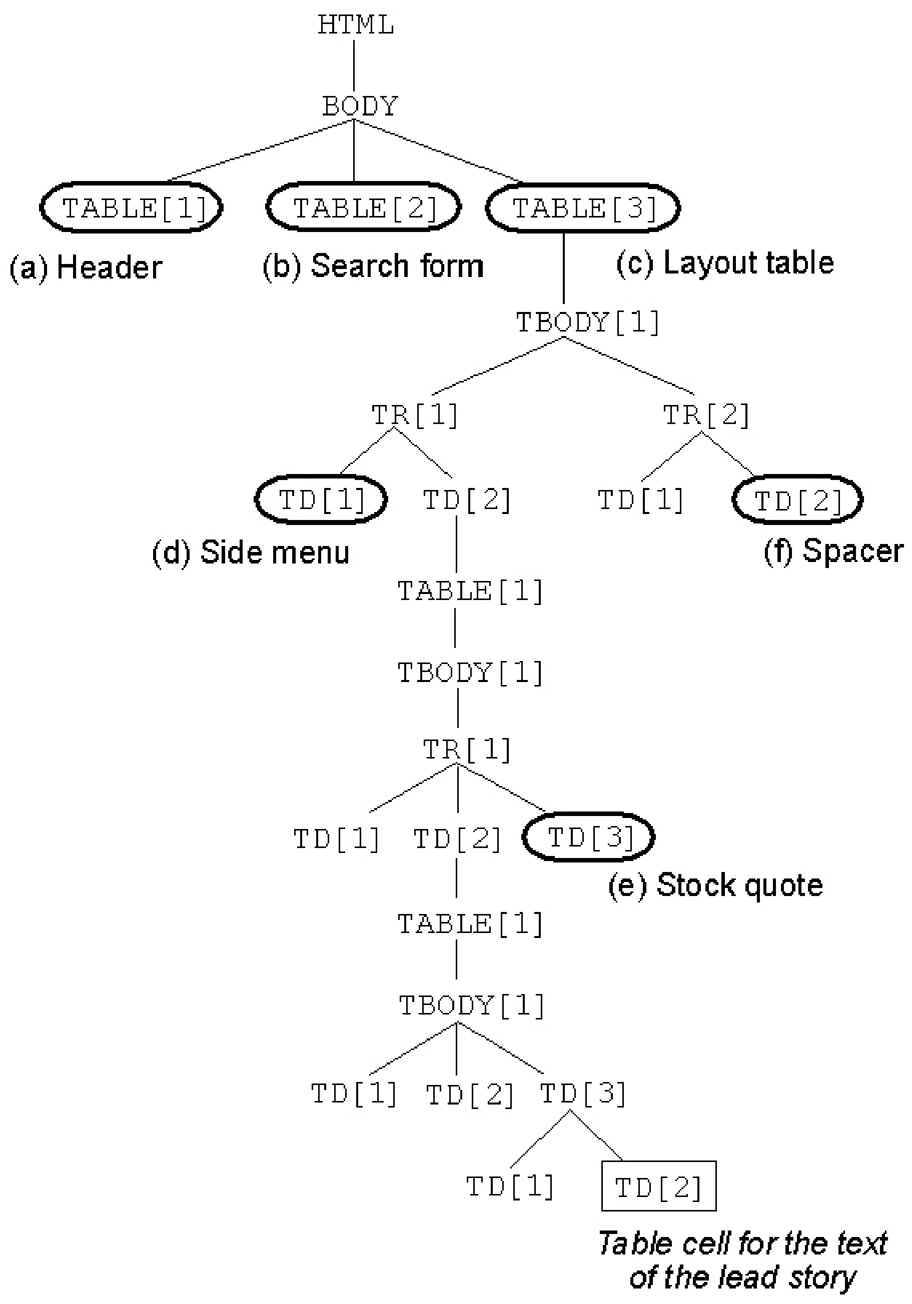


Figure 14.11 Annotated nodes in the news page

Regardless of the unchanged maximum depth of the document tree, the news page was actually changed during the period. Figure 14.12 shows the numbers of changed nodes in each of saved pages against the original news page. Changed nodes are counted by calculating the difference between two document trees [TreeDiff, 1999)]. The depth of changed nodes is distinguished in Figure 14.12, and the number of changed nodes was always more than 20 after the first 10 days. However, all the changes were done for the nodes whose depth is greater than or equal to 10, while the maximum depth of the annotated nodes was 9. Therefore, the node changes made no influence on the locations of annotated nodes during the observed period. This is merely a statement obtained from the observation of a particular Web document during the limited period, and can never be generalized to different situations. However, it is possible to draw an implication to a tactics for making the external annotations robust. The descriptions in Figure 14.9 are annotating auxiliary portions of the news page, such as header, side menu, and stock quote table. The auxiliary parts would not be subject to change so often as compared with the text contents of the news stories and their surrounding area. The text of the lead story in the original news page was actually located in the table cell, whose depth was 13, as indicated at the bottom of Figure 14.11. Furthermore, only one type of XPath expressions was used in the annotations in Figure 14.9, because that type of expressions allows to point at most one node regardless of the document structure. However, it is possible to provide other types of XPath expressions, taking account of the robustness of the addressing. This is also an important research issue that requires further investigation.

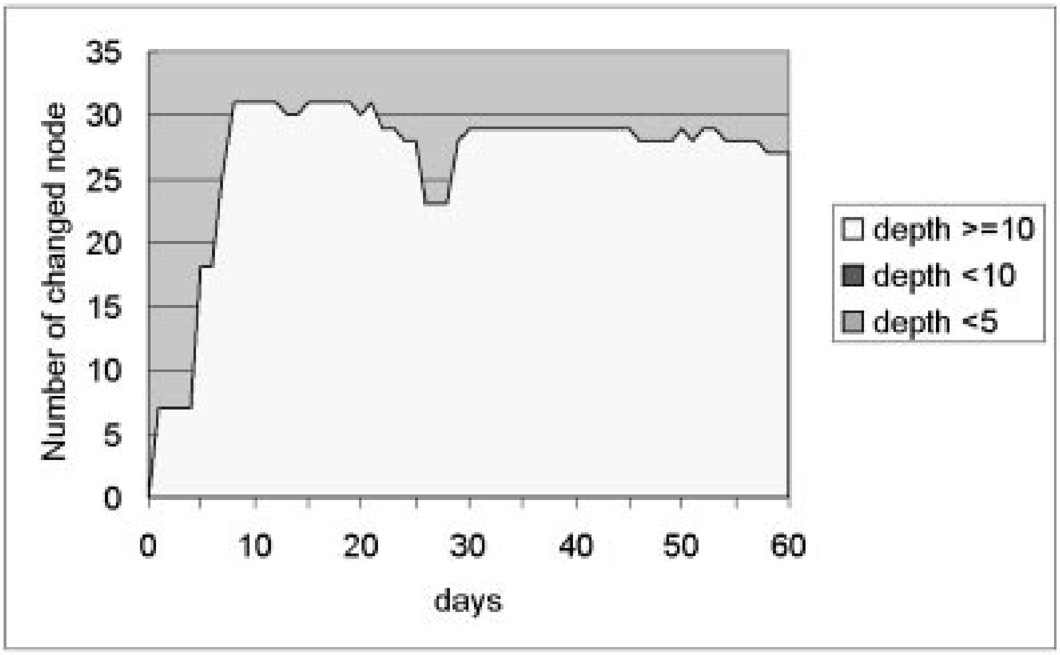


Figure 14.12 Number of changed nodes in the news page

5.2. Comparison with Related Work

Although annotation-based transcoding is an approach to realizing content adaptation, it is possible to think about the other approaches on the basis of the intermediary-based transcoding platform. One is to provide a custom-tailored transcoding module that runs without any external annotations. An automatic adaptation process has been proposed for device-independent access to Web contents [Bickmore et al., 1997]. A heuristic planner, which searches a document transformation space, selects the most promising state that occupies the smallest display. It is reported, however, that in the worst case the planner produces 80 versions of the document during the search process. If metadata or annotation is provided with the planner, the search space will be pruned more effectively. Note here that it is not an issue of whether metadata is embedded or external. The point is that metadata must be provided explicitly rather than implicitly in the adaptation algorithm.

Another approach is to use a general-purpose transformation engine such as an XSLT processor that employs externally provided transformation rules [XSLT, 1999]. XSLT rules can also be regarded as external annotations, because XSLT rules can be given as a separate style sheet externally and each rule entails an XPath expression that points to target nodes to be transformed. The advantage of the general-purpose transformation engine is the broadness of its applicability, while the HTML-page splitting plug-in, for example, is limited to the page-splitting task at hand. However, XSLT or XSL Transformation Language is more like a programming language rather than a declarative markup language. Therefore, authors of XSLT rules need to have in-depth understanding of the programming by XSLT, so that the authors can fully exploit the capability of the language.

In contrast, the advantage of a task-specific transcoding engine lies in the task-specific semantics that can be made explicit in the document definition of annotation vocabulary. To put it another way, adoption of task-specific transcoding approach is to trade the scope of applicability in the general-purpose transformation approach, for task-specific semantics articulated in the specification of an annotation vocabulary. In the case of the page splitting plug-in, roles such as header, auxiliary, and spacer supplement semantics that cannot be prescribed in the existing Web documents. In this sense, the importance of external annotation lies in the role of the mediating representation that articulates the semantics to be shared between human annotators and a content adaptation engine.

Conclusions

The external annotation framework presented in this article is applicable not only to transcoding for Web-enabled personal devices, but also to other cases in which content adaptation is desirable. For example, when HTML documents are translated into multiple target languages by means of a machine translation engine, linguistic annotations would be helpful for improving the translation accuracy [Nagao et al., 2001]. In other situations, content adaptation may be needed, so that user-side constraints can be taken into account. For example, text contents should be transcoded into audio content for those who cannot rely on the visual Web access [Asakawa et al., 2001].

Web-content transcoding allows Web documents to be reused for purposes that are not necessarily expected by the original content creators. Taking account of the situations of content re-purposing, Web documents are not always given with explicit semantics that are understandable by a content adaptation software. The benefit of annotation as supplementary semantics is obvious for HTML documents, because they are encoded with an presentation markup, namely, HTML for the display on Web browsers. However, XML documents can also be the target of external annotation. Platform-independent markup languages have been investigated for the abstract specifications of Web applications [Gellersen and Gaedke, 1999; Kitayama et al., 1999], XML user interface [Abrams et al., 1999], Web sites [Ceri et al., 2000], and online interaction [Xforms, 2001]. Such abstract specifications ultimately need to be specialized to be an platform-dependent representations. Web-content annotation is an approach promising for the specializations of abstract contents, and the external annotation framework presented in this article is applicable to XML documents as well as HTML documents, as long as they are represented as instances of a document object model [DOM, 1998]. Semantic annotation is thus a key to taking the full advantage of Web contents, by means of not only re-purposing the existing, legacy contents, but also elaborating platform-independent abstract content models for diverse needs of content use.

Acknowledgements

This chapter has grown out of the two papers. One is "Authoring tool for Web content transcoding" by M. Hori, K. Ono, G. Kondoh, and S. Singhal presented at Markup Technologies ’99, Philadelphia, PA, 7-9 December, 1999. Another paper is "Annotation-based Web content transcoding" by M. Hori, G. Kondoh, K. Ono, S. Hirose, and S. Singhal presented at the 9th International World Wide Web Conference, Amsterdam, Netherlands, 15-19 May, 2000. The author would like to thank the participants at the both conferences whose contributions have aided the development of this article.

References

[Abe & Hori, 2001] Abe, M. and Hori, M. (2001). Visual composition of XPath expressions for external metadata authoring. Research Report RT-0406, IBM Tokyo Research Laboratory.

[Abrams et al., 1999] Abrams, M., Phanouriou, C., Batongbacal, A. L., Williams, S. M., and Shuster, J. E. (1999). UIML: an appliance-independent XML user interface language. Proceedings of the 8th International World Wide Web Conference (WWW8), pp. 618-630, Toronto, Canada.

[Asakawa & Takagi, 2001] Asakawa, C. and Takagi, H. (2001). Transcoding system for non-visual Web access (2): annotation-based transcoding. Sixteenth International Conference on Technologies and Persons with Disabilities (CSUN2001).

[AWCT, 1999] Annotation of Web Content for Transcoding. W3C Note, http://www.w3.org/TR/annot/.

[Barret & Maglio, 1998] Barrett, R. and Maglio, P. P. (1998). Intermediaries: New places for producing and manipulating Web content. Proceedings of the 7th International World Wide Web Conference (WWW7), pp. 509-518, Brisbane, Australia.

[Barret & Maglio, 1999] Barrett, R. and Maglio, P. P. (1999). Intermediaries: An approach to manipulating information streams. IBM Systems Journal, Vol. 38, No. 4, pp. 629-641.

[Bickmore and Schilit 1997] Bickmore, T. W. and Schilit, B. N. (1997). Digestor: Device-independent access to the World Wide Web. Proceedings of the 6th International World Wide Web Conference (WWW6), pp. 1075-1082, Santa Clara, CA.

[Britton et al., 2001] Britton, K. H., Li, Y., Case, R., Seekamp, C., Citron, A., Topol, B., Floyd, R., and Tracey, K. (2001). Transcoding: extending e-business to new environments. IBM Systems Journal, Vol. 40, No. 1, pp. 153-178.

[Ceri et al., 2000] Ceri, S., Fraternali, P., and Bongio, A. (2000). Web modeling language (WebML): a modeling language for designing Web sites. Proceedings of the 9th International World Wide Web Conference (WWW9), pp. 137-157, Amsterdam, Netherlands.

[CCPP, 2000] Composite Capabilities/Preference Profiles: Requirements and Architecture. W3C Working Draft, http://www.w3.org/TR/CCPP-ra/.

[CCPP-exchange, 1999] CC/PP exchange protocol based on HTTP Extension Framework, W3C Note, http://www.w3.org/TR/NOTE-CCPPexchange.

[Denoue & Vignollet, 2000] Denoue, L. and Vignollet, L. (2000). An annotation tool for Web browsers and its applications to information retrieval. Proceedings of the 6th Conference on Conten-Based Multimedia Information Access (RIAO 2000), Paris, France.

[DOM, 1998] Document Object Model (DOM) Level 1 Specification Version 1.0. W3C Recommendation, http://www.w3.org/TR/REC-DOM-Level-1/.

[Erdmann et al., 2000] Erdmann, M., Maedche, A., Schnurr, H.-P., and Staab, S. (2000). From manual to semi-automatic semantic annotation: about ontology-based text annotation tools. Proceedings of the COLING 2000 Workshop on Semantic Annotation and Intelligent Content, Luxembourg.

[Fox and Brewer 1996] Fox, A. and Brewer, E. A. (1996). Reducing WWW latency and bandwidth requirements by real-time distillation. Proceedings of the 5th International World Wide Web Conference (WWW5), pp. 1445-1456, Paris, France.

[Gellersen & Gaedke, 1999] Gellersen, H.-W. and Gaedke, A. (1999). Object-oriented Web application development. IEEE Internet Computing, Vol. 3, No. 1, pp. 60-68.

[Heflin & Hendler, 2000] Heflin, J. and Hendler, J. (2000). Semantic interoperability on the Web. Proceedings of Extreme Markup Languages 2000, pp. 111-120.

[Hori et al., 2000a] Hori, M., Kondoh, G., Ono, K., Hirose, S., and Singhal, S. (2000a). Annotation-based Web content transcoding. Proceedings of the 9th International World Wide Web Conference (WWW9), pp. 197-211, Amsterdam, Netherlands.

[Hori et al., 2000b] Hori, M., Ono, K., Kondoh, G., and Singhal, S. (2000b). Authoring tool for Web content transcoding. Markup Languages: Theory & Practice, Vol. 2, No. 1, pp. 81-106.

[HTML, 1999] HTML 4.01 Specification. W3C Recommendation, http://www.w3.org/TR/html401/.

[Jones et al., 1999] Jones, M., Marsden, G., Nasir, N., Boone, K., and Buchanam, G. (1999). Improving Web interaction on small displays. Proceedings of the 8th International World Wide Web Conference (WWW8), pp. 51-59, Toronto, Canada.

[Kitayama et al., 1999] Kitayama, F., Hirose, S., Kondho, G., and Kuse, K. (1999). Design of a framework for dynamic content adaptation to Web-enabled terminals and enterprise applications. Proceedings of the Sixth Asia Pacific Engineering Conference (APSEC '99), pp. 72-79, Takamatsu, Japan.

[Lassila 1998] Lassila, O. (1998). Web metadata: a matter of semantics. IEEE Internet Computing, Vol. 2, No. 4, pp. 30-37.

[Mea et al., 1996] Mea, V. D., Beltrami, C. A., Roberto, V., and Brunato, D. (1996). HTML generation and semantic markup for telepathology. Proceedings of the 5th International World Wide Web Conference (WWW5), pp. 1085-1094, Paris, France.

[Nagao et al., 2001] Nagao, K., Shirai, Y., and Kevin, S. (2001). Semantic annotation and transcoding: making Web content more accessible. IEEE Multimedia, Vol.8, No.2 (to appear).

[RDF, 1999] Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, http://www.w3.org/TR/REC-rdf-syntax/.

[Rousseau et al. 1999] Rousseau, F., Macias, J. A., de Lima, J. V., and Duda, A. (1999). User adaptable multimedia presentations for the World Wide Web. Proceedings of the 8th International World Wide Web Conference (WWW8), pp. 195-212, Toronto, Canada.

[Sakairi & Takagi, 2001] Sakairi, T. and Takagi, H. (2001). An annotation editor for non-visual Web access. Proceedings of the 9th International Conference on Human-Computer Interaction (HCI International 2001), New Orleans, LA (to appear).

[Schickler et al., 1996] Schickler, M. A. , Mazer, M. S., and Brooks, C. (1996). Pan-Browser support for annotations and other meta-information on the World Wide Web. Proceedings of the 5th International World Wide Web Conference (WWW5), pp. 1063-1074, Paris, France.

[Style Sheets, 2000] Style Sheets Activity Statement. W3C User Interface Domain Activity Statement, http://www.w3.org/Style/Activity.

[Thompson et al., 1999] Thompson, C., Pazandak, P., Vasudevan, V., Manola, F., Palmer, M., Hansen, G., and Bannon, T. (1999). Intermediary architecture: interposing middleware object services between Web client and server. ACM Computing Surveys 31(2es) #14.

[TreeDiff, 1999] XML TreeDiff. alphaWorks, IBM Corp. http://www.alphaworks.ibm.com/tech/xmltreediff.

[Vasudevan & Palmer, 1999] Vasudevan, V. and Palmer, M. (1999). On Web annotations: promises and pitfalls of current Web infrastructure. Proceedings of the 32nd Hawaii International Conference on Systems Sciences, Maui, Hawaii.

[WCAG, 1999] Web Content Accessibility Guidelines 1.0. W3C Recommendation,http://www.w3.org/TR/WAI-WEBCONTENT/.

[WBI, 2000] Web Intermediaries (WBI). http://www.almaden.ibm.com/cs/wbi/.

[WTP, 2001] WebSphere Transcoding Publisher, IBM Corp. http://www.ibm.com/software/webservers/transcoding/.

[WTP Developer's Guide, 2000]. IBM WebSphere Transcoding Publisher Version 3.5 Developer's Guide. IBM Corp.

[Xforms, 2001]. XForms 1.0. W3C Working Draft, http://www.w3.org/TR/xforms/.

[XML-names, 1999]. Namespaces in XML. W3C Recommendation, http://www.w3.org/TR/REC-xml-names/.

[XHTML-prof, 1999]. XHTML Document Profile Requirements. W3C Working Draft, http://www.w3.org/TR/xhtml-prof-req/.

[Xlink, 2000]. XML Linking Language (XLink) Version 1.0. W3C Proposed Recommendation, http://www.w3.org/TR/xlink/.

[Xpath, 1999]. XML Path Language (XPath) Version 1.0. W3C Recommendation, http://www.w3.org/TR/xpath.

[Xpointer, 2001]. XML Pointer Language (XPointer) Version 1.0. W3C Working Draft, http://www.w3.org/TR/xptr.

[XSLT, 1999]. XSL Transformations (XSLT) Version 1.0. W3C Recommendation, http://www.w3.org/TR/xslt.

Footnotes

[1] IBM Tokyo Research Laboratory 1623-14 Shimo-tsuruma, Yamato-shi, Kanagawa 242-8502 Japan, Email: horim@jp.ibm.com Phone: +81-462-73-4667, Fax: +81-462-74-4282

15. Task Achieving Agents on the World Wide Web

Austin Tate[1], Jeff Dalton[1], John Levine[1] and Alex Nixon[1]

1. Introduction and Motivation

The World Wide Web currently acts as a vast electronic library, serving information and providing search facilities for accessing that information. However, given that the Web actually consists of a vast network of task achieving agents (humans and computers), this view of the Web as a static pool of information is only using a small fraction of its real capabilities.

The idea of the Web being a place where you can ask agents to *do* things and to *plan* activities seems intuitively attractive. However, the data models and standards developed to date for the Web mostly relate to information retrieval, rather than activity and the planning of future activity. In order to make the Web a place for "doing things" as well as "finding things", we need shared models and ontologies to represent the entities involved in planning and doing: activities, tasks, plans, agent capabilities, and so on.

The AI planning and the process modelling communities have recently started to develop standards in these areas, for the purpose of working on common models and sharing information about activities and processes [Tate, 1998]. These common models and ontologies might form the generic core of a shared ontology to support the movement of information and data relating to activities over the World Wide Web.

This paper is in two parts. In the first part, we describe work towards the creation of a common ontology and representation for plans, processes and other information related to activity. We briefly describe the work going on in two areas: military planning and standards for representing activities and processes.

Our own systems are based on an underlying activity ontology called <I-N-OVA>; this is described, together with the more general <I-N-CA> constraint-based model for representing synthesised artefacts. In both of these models, the I stands for *issues*, which allows us to represent synthesised artefacts which are not yet complete or which have some outstanding issues to address. The list of outstanding issues is crucial in the communication of partial results between agents, which is clearly needed in multi-agent systems which work together to synthesise solutions.

In the second part, we describe our work on producing collaborative multi-agent systems consisting of human and computer agents engaging in planning and plan execution support over the World Wide Web. These applications are based on a generic interface for web-based task achieving agents called Open Planning Process Panels or O-P^3 [Levine, Tate & Dalton, 2000]. These panels are described briefly to introduce the work that follows. Three web-based applications are then described: the O-Plan Web demonstration, the Air Campaign Planning Process Panel (ACP^3) and a version of O-Plan that can run over the Web using a WAP-enabled mobile telephone. These applications are indicative of the kind of systems which we see being deployed in the near future, where the Web site acts as an interface to one or more intelligent agents and the common representation of activity-related information is crucial.

2. Standards for Representing Activities

In the first part of this paper, we describe work towards the creation of a common ontology and representation for plans, processes and other information related to activity.

There are two major stands of work here. In military planning work, there has already been much work in developing shared models for planning and representing plans, such as the KRSL plan representation language, the Core Plan Representation (CPR) and the Shared Planning and Activity Representation (SPAR).

At the same time, work in the standards community has attempted to standardize the terminology for talking about activities and processes: examples include the Process Interchange Format (PIF), NIST Process Specification Language (PSL), and work by the Workflow Management Coalition (WfMC).

[Tate, 1998] gives an overview and history of all these efforts and shows their relationship to the Shared Planning and Activity Representation (SPAR) developed under the DARPA and USAF Research Laboratory planning initiative (ARPI). Full references are provided in that paper and in its on-line copy[2].

Our own systems are based on the <I-N-OVA>[3] activity ontology; this relates well to the other ontologies of activity described above, such as SPAR, and can be considered as an abstract model which can underlie these. The <I-N-OVA> model is described in the following sections, together with the more general <I-N-CA> model for representing synthesised artefacts.

2.1. <I-N-OVA> and <I-N-CA>

This section presents an approach to representing and manipulating plans and other synthesised artefacts in the form of a set of constraints. The <I-N-OVA> (*Issues – Nodes – Orderings/Variables/Auxiliary*) constraints model is used to characterise plans and processes. The more general <I-N-CA> (*Issues – Nodes – Critical/Auxiliary*) constraints model can be used for wider applications in design, configuration and other tasks which can be characterised as the synthesis and maintenance of an artefact or product.

2.2. Motivation

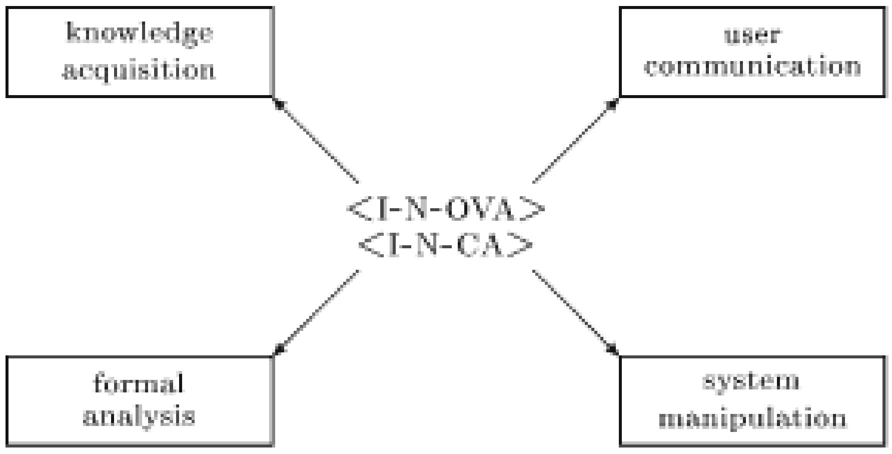


Figure 15.1 Uses of <I-N-OVA> and <I-N-CA>

As shown in Figure 15.1, the <I-N-OVA> and <I-N-CA> constraint models are intended to support a number of different uses:

- for automatic manipulation of plans and other synthesised artefacts and to act as an ontology to underpin such use;
- as a common basis for human communication about plans and other synthesised artefacts;
- as a target for principled and reliable acquisition of plans, models and product information;
- to support formal reasoning about plans and other synthesised artefacts.
- These cover both formal and practical requirements and encompass the requirements for both human and computer-based planning and design systems.

The <I-N-OVA> model is a means to represent plans and activity as a set of constraints. By having a clear description of the different components within a plan, the model allows for plans to be manipulated and used separately from the environments in which they are generated. The underlying thesis is that activity can be represented by a set of constraints on the behaviours possible in the domain being modelled and that activity communication can take place through the interchange of such constraint information.

<I-N-OVA>, when first designed [Tate, 1996b], was intended to act as a bridge to improve dialogue between a number of communities working on formal planning theories, practical planning systems and systems engineering process management methodologies. It was intended to support new work then emerging on automatic manipulation of plans, human communication about plans, principled and reliable acquisition of plan information, mixed-initiative planning and formal reasoning about plans. It has since been used as the basis for a number of research efforts, practical applications and emerging international standards for plan and process representations. For some of the history and relationships between earlier work in AI on plan representations, work from the process and design communities and the standards bodies, and the part that <I-N-OVA> played in this, see [Tate, 1998].

2.3. Representing Plans in <I-N-OVA>

A plan is represented as a set of constraints which together limit the behaviour that is desired when the plan is executed. The set of constraints are of three principal types with a number of sub-types reflecting practical experience in a number of planning systems.

Plan Constraints
- I - Issues (Implied Constraints)
- N - Node Constraints (on Activities)
- OVA - Detailed Constraints
  - O - Ordering Constraints
  - V - Variable Constraints
  - A - Auxiliary Constraints
    - - Authority Constraints
    - - Condition Constraints
    - - Resource Constraints
    - - Spatial Constraints
    - - Miscellaneous Constraints

Figure 15.2 <I-N-OVA> Constraint Model of Activity

The node constraints (these are often of the form "include activity") in the <I-N-OVA> model set the space within which a plan may be further constrained. The I (issues) and OVA constraints restrict the plans within that space which are valid. Ordering (temporal) and variable constraints are distinguished from all other auxiliary constraints since these act as *cross-constraints* [4], usually being involved in describing the others – such as in a resource constraint which will often refer to plan objects/variables and to time points or ranges.

In [Tate, 1996b], the <I-N-OVA> model is used to characterise the plan representation used within O-Plan [Currie & Tate, 1991; Tate, Drabble and Dalton, 1994] and is related to the plan refinement planning method used in O-Plan.

We have generalised the <I-N-OVA> approach to design and configuration tasks with I, N, CA components - where C represents the "critical constraints" in any particular domain - much as certain O and V constraints do in a planning domain. We believe the approach is valid in design and synthesis tasks more generally - we consider planning to be a limited type of design activity. <I-N-CA> is used as an underlying ontology for the I-X project [5].

2.4. Rationale for the Categories of Constraints within <I-N-OVA>

Planning is the taking of planning decisions (I) which select the activities to perform (N) which creates, modifies or uses the plan objects or products (V) at the correct time (O) within the authority, resources and other constraints specified (A). The Issues (I) constraints are the items on which selection of Plan Modification Operators is made in agenda based planners.

Others have recognised the special nature of the inclusion of activities into a plan compared to all the other constraints that may be described. [Khambhampati & Srivastava, 1996] differentiate Plan Modification operators into "progressive refinements" which can introduce new actions into the plan, and "non-progressive refinements" which just partitions the search space with existing sets of actions in the plan. They call the former genuine planning refinement operators, and think of the latter as providing the scheduling component.

If we consider the process of planning as a large constraint satisfaction task, we may try to model this as a Constraint Satisfaction Problem (CSP) represented by a set of variables to which we have to give a consistent assignment of values. In this case we can note that the addition of new nodes ("include activity" constraints in <I-N-OVA>) is the only constraint which can add variables dynamically to the CSP. The Issue (I) constraints may be separated into two kinds: those which may (directly or indirectly) add nodes to the plan and those which cannot. The I constraints which can lead to the inclusion of new nodes are of a different nature in the planning process to those which cannot.

Ordering (temporal) and variable constraints are distinguished from all other auxiliary constraints since these act as cross-constraints, usually being involved in describing the others – such as in a resource constraint which will often refer to plan objects/variables and to time points or intervals.

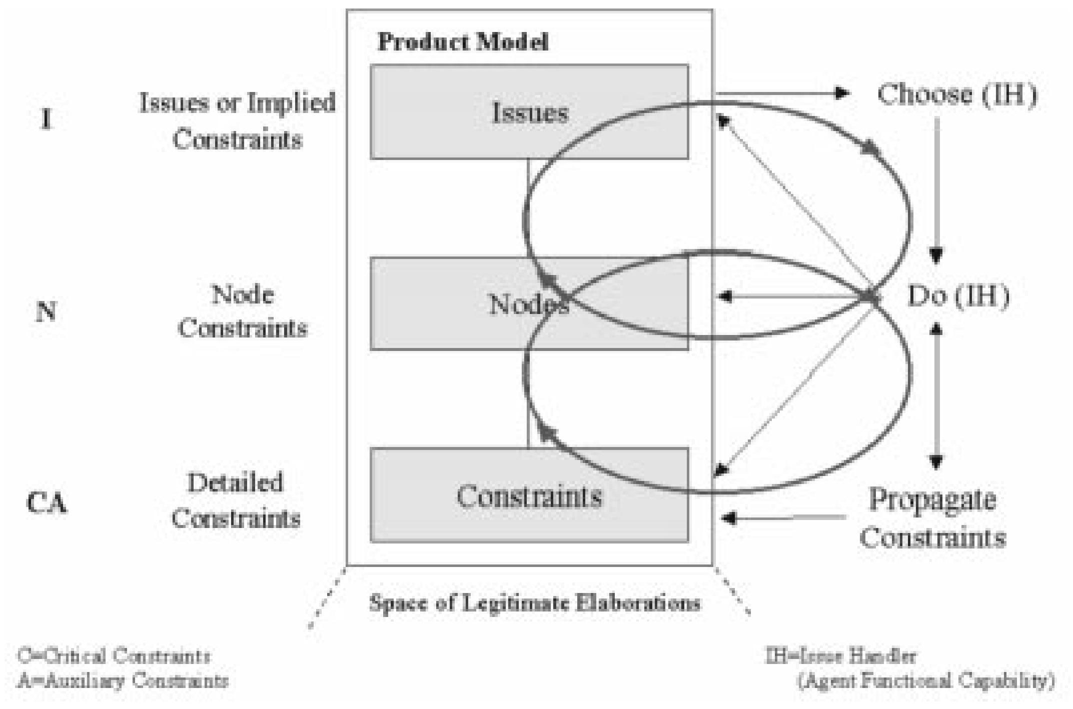


Figure 15.3 I-X - Two Cycles of Processing - Handle Issues, Respect Constraints

2.5. Sorted First Order Logic Base, and XML

<I-N-OVA> and <I-N-CA> are meant as conceptual models which can underlie any of a range of languages which can describe activities, plans, processes and other synthesised artefacts. For example, O-Plan is based on <I-N-OVA>, but utilises the Task Formalism domain description language which has a simple keyword introduced syntax.

It is anticipated that any <I-N-OVA> or the more general <I-N-CA> model in whatever language or format it is expressed can be reduced to a conjunctive set of statements in first order logic with strong requirements on the type of the terms involved in each statement – i.e. a sorted first order logic.

See [Polyak & Tate, 1998] for further details, and for a use described in a planning domain modelling support system.

<I-N-OVA> and <I-N-CA> constraint sets lend themselves very well to being used in eXtendible Markup Language (XML) representations of synthesised artefacts, especially when these are still in the process of being designed or synthesised. The processes that are used to do this synthesis and the collaborations and capabilities involved can also be described in <I-N-OVA> and/or <I-N-CA>.

3. Web-Based Applications

In the second part, we describe our work on producing collaborative multi-agent systems consisting of human and computer agents engaging in planning and plan execution support over the World Wide Web. These applications are based on a generic interface for web-based task achieving agents called Open Planning Process Panels or O-P^3 [Levine, Tate & Dalton, 2000]. These panels are described briefly to introduce the work that follows. Three web-based applications are then described: the O-Plan Web demonstration, the Air Campaign Planning Process Panel (ACP^3), and a version of O-Plan that can run over the Web using a WAP-enabled mobile telephone. These applications are indicative of the kind of systems which we see being deployed in the near future, where the Web site acts as an interface to one or more intelligent agents and the common representation of activity-related information is crucial.

3.1. Open Planning Process Panels

Real world planning is a complicated business. Courses of action to meet a given situation are constructed collaboratively between teams of people using many different pieces of software. The people in the teams will have different roles, and the software will be used for different purposes, such as planning, scheduling, plan evaluation, and simulation. Alternative plans will be developed, compared and evaluated, and more than one may be chosen for briefing. In general, planning is an example of a multi-user, multi-agent collaboration in which different options for the synthesis of a solution to given requirements will be explored.

The process of planning is itself the execution of a plan, with agents acting in parallel, sharing resources, communicating results and so on. This planning process can be made explicit and used as a central device for workflow coordination and visualisation.

We have used this idea to create Open Planning Process Panels (O-P^3). These panels are used to coordinate the workflow between multiple agents and visualise the development and evaluation of multiple courses of action (COAs). The generic notion of O-P^3 has been used to implement an O-Plan two-user mixed-initiative planning Web demonstration and an Air

Campaign Planning Process Panel (ACP^3). In the former, O-P^3 technology is used to enable the development and evaluation of multiple COAs by a commander, a planning staff member and the O-Plan automated planning agent. In the latter, O-P^3 is used to build a visualisation panel for a complex multi-agent planning and evaluation demonstration which uses 11 different software components and involves several users.

O-P^3 technology could have an impact on several important research areas:

Automated planning: O-P^3 shows how automated planning aids such as AI planners can be used within the context of a wider workflow involving other system agents and human users.

Computer-supported cooperative work (CSCW): O-P^3 uses explicit models of the collaborative planning workflow to coordinate the overall effort of constructing and evaluating different courses of action. This is generalisable to other team-based synthesis tasks using activity models of the task in question (e.g. design or configuration).

Multi-agent mixed-initiative planning: O-P^3 facilitates the sharing of the actions in the planning process between different human and system agents and allows for agents to take the initiative within the roles that they play and the authority that they have [Tate, 1993].

Workflow support: O-P^3 provides support for the workflow of human and system agents working together to create courses of action. The workflow and the developing artefact (i.e. the course of action) can be visualised and guided using O-P^3 technology.

The kind of planning system that we envisage O-P^3 being used for is one in which the planning is performed by a team of people and a collection of computer-based planning agents, who act together to solve a hard, real world planning problem. Both the human and the software agents will act in given roles and will be constrained by what they are authorised to do, but they will also have the ability to work under their own initiative and volunteer results when this is appropriate. When the planning process is underway, the agents will typically be working in parallel on distinct parts of the plan synthesis. The agents will also be working in parallel to explore different possible courses of action; for example, while one COA is being evaluated, another two may be in the process of being synthesised.

This section introduces O-P^3 technology. It begins with a description of the generic O-P^3 ideas, based on the central notion of an explicit shared model of the activities involved in creating a plan – the planning process.

3.2. Generic O-P^3 Technology

The generic O-P^3 is based on an explicit model of the planning process, which is encoded using an activity modelling language, such as <I-N-OVA>. This represents the planning process as a partially-ordered network of actions, with some actions having expansions down to a finer level of detail (i.e. to another partially-ordered network).

The purpose of O-P^3 is to display the status of the steps in the planning process to the users, to allow the users to compare the products of the planning process (i.e. the courses of action) and to allow the users to control the next steps on the "workflow fringe" (i.e. what actions are possible next given the current status of the planning process). In the context of creating plans, O-P^3 is designed to allow the development of multiple courses of action and the evaluation of those courses of action using various plan evaluations.

A generic O-P^3 panel would have any of a number of "sub-panels", which can be tailored to support specific users or user roles. These include:

A course of action comparison matrix showing:

COAs vs elements of evaluation, with the plan evaluations being provided by plug-in plan evaluators or plan evaluation agents;

the steps in the planning process (from the explicit process model), the current status of those steps (the *state model*), and control for the human agent of what action to execute next;

the *issues* outstanding for a COA that is being synthesised and which must be addressed before the COA is ready to execute;

a graphical display showing the status of the planning process as a PERT chart, which is a useful alternative view of the planning process to that given by the tabular matrix display;

other visualisations, such as bar charts, intermediate process product descriptions, and textual description of plans.

The generic O-P^3 methodology for building Open Planning Process Panels consists of the following steps:

Consider the agents (human and system) who are involved in the overall process of planning. Assign roles and authorities to these agents.

Construct an activity model of the planning process, showing the partial ordering and decomposition of the actions and which agents can carry out which actions. This activity model could be represented using an activity modelling language such as <I-N-OVA>.

Build a model of the current state of the planning process and an activity monitor which will update this state model as actions in the planning process take place.

Construct appropriate O-P^3 interfaces for each of the human agents in the planning process, taking into account the role which they play in the interaction. This means that each different user role will have an O-P^3 interface which is tailored to the overall nature of their task.

The O-P^3 agent interfaces then allow the human agents to play their part in the overall planning process, alongside the system agents, which will be AI planners, schedulers, plan evaluators and so on. This is illustrated in Figure 15.4.

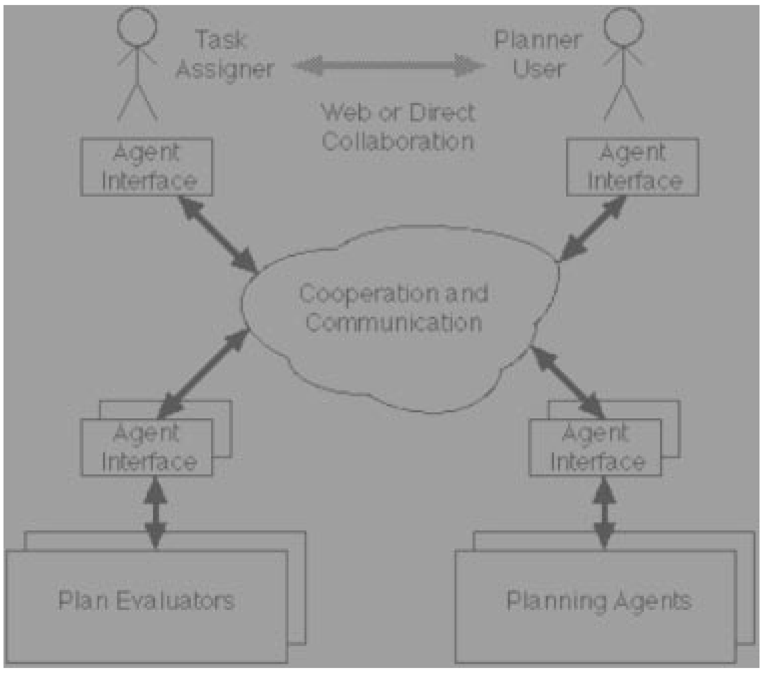


Figure 15.4 Using O-P^3 Interfaces

4. Application 1 – O-Plan on the Web

The O-Plan project [Tate, Drabble & Dalton, 1996; Tate, Dalton and Levine, 1998] was concerned with providing support for mixed-initiative planning. The web-based demonstration described here [6] shows interaction between two human agents and one software planning agent (the O-Plan plan server). The overall concept for our demonstrations of O-Plan acting in a mixed-initiative multi-agent environment is to have humans and systems working together to populate the COA matrix component of the O-P^3 interface.

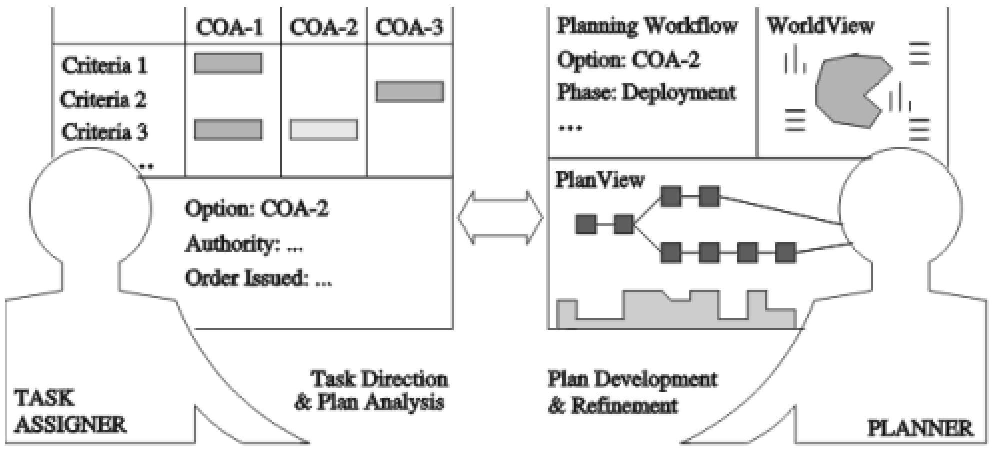


Figure 15.5 Communication between TA and Planner

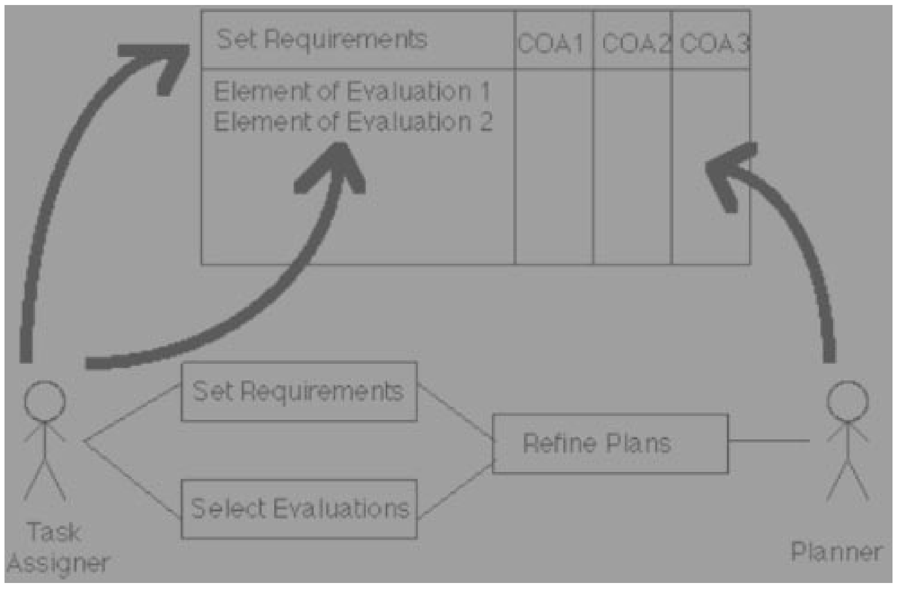


Figure 15.6 Roles of the Task Assigner and the Planner

As shown in Figure 15.5, we envisage two human agents acting in the user roles of Task Assigner and Planner User, working together to explore possible solutions to a problem and making use of automated planning aids to do this. Figure 15.6 shows how the two human agents work together to populate the matrix. The Task Assigner sets the requirements for a particular course of action (i.e. what top level tasks must be performed), selects appropriate evaluation criteria for the resulting plans and decides which courses of action to prepare for briefing. The Planner User works with O-Plan to explore and refine the different possible course of action for a given set of top level requirements. The two users can work in parallel, as is demonstrated in the example scenario (Levine, Tate and Dalton, 2000).

The overall planning task is thus shared between three agents who act in distinct user and system roles. The Task Assigner (TA) is a commander who is given a crisis to deal with and who needs to explore some options. This person will be given field reports on the developing crisis and environmental conditions. The Planner User is a member of staff whose role is to provide the Task Assigner with plans which meet the specified criteria. In doing this, the Planner User will make use of the O-Plan automated planning agent, whose role is to generate plans for the Planner User to see. The Planner User will typically generate a number of possible course of action using O-Plan and only return the best ones to the Task Assigner.

For our current demonstration, we are using a general purpose logistics and crisis operations domain which is an extension of our earlier Non-Combative Evacuation Operations (NEO) and logistics-related domains [Reece et al., 1993]. This domain, together with the O-Plan Task Formalism (TF) implementation, is described in detail by [Tate, Dalton & Levine 1998].

The two human users are provided with individual O-P^3 panels which are implemented using a CGI-initiated HTTP server in Common Lisp and which therefore run in any World Wide Web browser – the Common Lisp process returns standard HTML pages. This way of working has many advantages:

the two users can be using different types of machine (Unix, PC, Mac) and running different types of Web browser (Netscape, Internet Explorer, Hotjava, etc.);

the only requirement for running O-Plan is a World Wide Web connection and a Web browser (i.e. no additional software installation is needed);

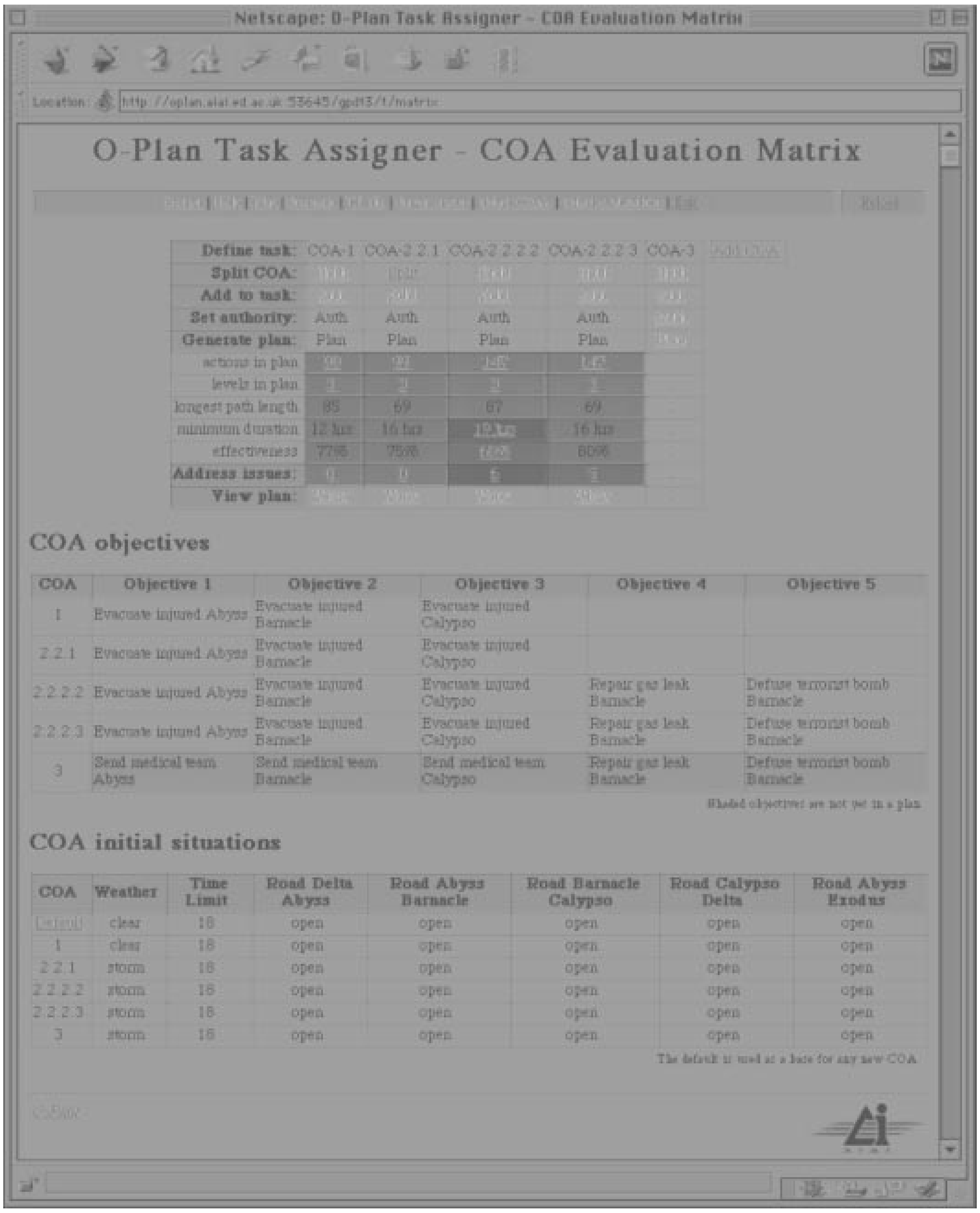
the two users can be geographically separate – in this case, voice communication via the telephone or teleconferencing is all that is required in addition to the linked O-P^3 interfaces.

The planning process for the Task Assigner and the Planner User is made explicit through the hypertext options displayed in the process parts of the O-P^3 panels. These are either not present (not ready to run yet), active (on the workflow fringe) or inactive (completed). Further parts of the planning process are driven by *issues* which O-Plan or the plan evaluation agents can raise about a plan under construction and which can be handled by either or both of the human agents. Because the planning process is made explicit to the two users through these two mechanisms, other visualisations of the planning process itself are not required. However, the products of the planning process (the courses of action) are complex artefacts for which multiple views are needed. In the current version, the courses of action can be viewed as a PERT network, as a textual narrative, or as a plan level expansion tree (all at various levels of detail).

The user roles are arranged such that the Task Assigner has authority over the Planner User who in turn has authority over O-Plan. This means that the Task Assigner defines the limits of the Planner User's activity (e.g. only plan to level 2) and the Planner User then acts within those bounds to define what O-Plan can do (e.g. only plan to level 2 and allow user choice of schemas). Other aspects of what the two users are authorised to do are made explicit by the facilities included in their respective panels.

4.1. The COA Comparison Matrix

The two panels for the Task Assigner and Planner User are shown in Figure 15.7 and Figure 15.8. Each user has control over the plan evaluation elements which are shown, to enable the critical elements of evaluation to be chosen. In the example scenario given later, the Task Assigner is only interested in the minimum duration and the effectiveness, so only these are selected. On the other hand, the Planner User wants a variety of data to pick the best COA, so all evaluations are shown.



Netscape: O-Plan Task Assigner - COA Evaluation Matrix

Location: http://oplan.aiai.ed.ac.uk:53645/gpdt3/t/matrix

O-Plan Task Assigner - COA Evaluation Matrix

Define task:	COA-1	COA-2.2.1	COA-2.2.2.2	COA-2.2.2.3	COA-3	[illegible]
Split COA:	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
Add to task:	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
Set authority:	Auth	Auth	Auth	Auth	[illegible]	
Generate plan:	Plan	Plan	Plan	Plan	[illegible]	
actions in plan	[illegible]	[illegible]	[illegible]	[illegible]		
levels in plan	[illegible]	[illegible]	[illegible]	[illegible]		
longest path length	85	69	67	69		
minimum duration	12 hrs	16 hrs	12 hrs	16 hrs		
effectiveness	77%	75%	[illegible]	80%		
Address issues:	0	0	6	1		
View plan:	[illegible]	[illegible]	[illegible]	[illegible]		

COA objectives

COA	Objective 1	Objective 2	Objective 3	Objective 4	Objective 5
1	Evacuate injured Abyss	Evacuate injured Barnacle	Evacuate injured Calypso		
2.2.1	Evacuate injured Abyss	Evacuate injured Barnacle	Evacuate injured Calypso		
2.2.2.2	Evacuate injured Abyss	Evacuate injured Barnacle	Evacuate injured Calypso	Repair gas leak Barnacle	Defuse terrorist bomb Barnacle
2.2.2.3	Evacuate injured Abyss	Evacuate injured Barnacle	Evacuate injured Calypso	Repair gas leak Barnacle	Defuse terrorist bomb Barnacle
3	Send medical team Abyss	Send medical team Barnacle	Send medical team Calypso	Repair gas leak Barnacle	Defuse terrorist bomb Barnacle

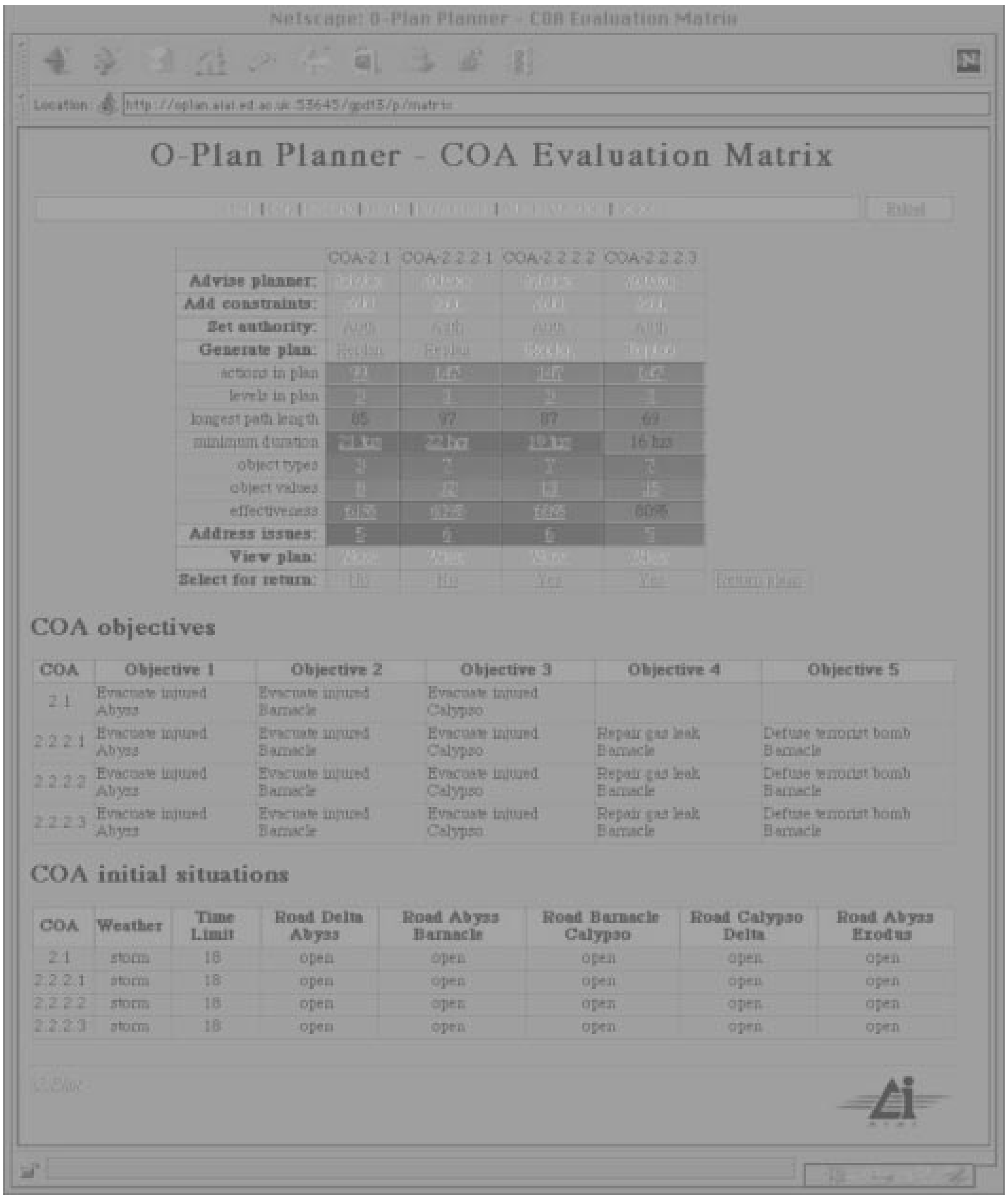
Shaded objectives are not yet in a plan.

COA initial situations

COA	Weather	Time Limit	Road Delta Abyss	Road Abyss Barnacle	Road Barnacle Calypso	Road Calypso Delta	Road Abyss Exodus
Default	clear	18	open	open	open	open	open
1	clear	18	open	open	open	open	open
2.2.1	storm	18	open	open	open	open	open
2.2.2.2	storm	18	open	open	open	open	open
2.2.2.3	storm	18	open	open	open	open	open
3	storm	18	open	open	open	open	open

The default is used as a base for any new COA.

Figure 15.7 O-Plan Task Assigner's Panel



Netscape: O-Plan Planner - COA Evaluation Matrix

Location: http://oplan.aiai.ed.ac.uk:53645/gpdt3/p/matrix

O-Plan Planner - COA Evaluation Matrix

	COA-2.1	COA-2.2.2.1	COA-2.2.2.2	COA-2.2.2.3
Advise planner:	Advice	Advice	Advice	Advice
Add constraints:	Add	Add	Add	Add
Set authority:	Auth	Auth	Auth	Auth
Generate plan:	Replan	Replan	Replan	Replan
actions in plan	[illegible]	[illegible]	[illegible]	[illegible]
levels in plan	3	3	3	3
longest path length	85	97	87	69
minimum duration	[illegible]	22 hrs	19 hrs	16 hrs
object types	3	7	7	7
object values	8	12	13	15
effectiveness	[illegible]	[illegible]	[illegible]	80%
Address issues:	5	6	6	5
View plan:	View	View	View	View
Select for return:	No	No	Yes	Yes

Return plans

COA objectives

COA	Objective 1	Objective 2	Objective 3	Objective 4	Objective 5
2.1	Evacuate injured Abyss	Evacuate injured Barnacle	Evacuate injured Calypso		
2.2.2.1	Evacuate injured Abyss	Evacuate injured Barnacle	Evacuate injured Calypso	Repair gas leak Barnacle	Defuse terrorist bomb Barnacle
2.2.2.2	Evacuate injured Abyss	Evacuate injured Barnacle	Evacuate injured Calypso	Repair gas leak Barnacle	Defuse terrorist bomb Barnacle
2.2.2.3	Evacuate injured Abyss	Evacuate injured Barnacle	Evacuate injured Calypso	Repair gas leak Barnacle	Defuse terrorist bomb Barnacle

COA initial situations

COA	Weather	Time Limit	Road Delta Abyss	Road Abyss Barnacle	Road Barnacle Calypso	Road Calypso Delta	Road Abyss Exodus
2.1	storm	18	open	open	open	open	open
2.2.2.1	storm	18	open	open	open	open	open
2.2.2.2	storm	18	open	open	open	open	open
2.2.2.3	storm	18	open	open	open	open	open

O-Plan

Figure 15.8 O-Plan Planner User's Panel

The role of the Task Assigner is to set up the top level requirements for a course of action. Once this is done, the COA is passed across to the Planner User, whose matrix is initially blank. The Planner User then explores a range of possible COAs for the specified requirements and returns the best ones to the Task Assigner. When the Planner User returns a COA to the Task Assigner, the column for that COA appears in the Task Assigner's matrix. The Planner User and the Task Assigner can be working in parallel, as demonstrated in the scenario.

5. Application 2 – ACP^3

One of the integrated demonstrations from the US DARPA ARFL/Rome Planning Initiative (ARPI) [Tate, 1996a] brings together eleven, separately developed, software systems for planning and plan evaluation. When the demonstration is run, these systems work together to create and evaluate multiple courses of action in the domain of Air Campaign Planning. The systems communicate with each other by exchanging KQML messages [Finin, Labrou & Mayfield, 1997]. Finding out what is happening at any given time could (in theory) be done by watching these KQML messages, but this was obviously less than ideal as these messages use technological terms which are far removed from the terminology used by the user community.

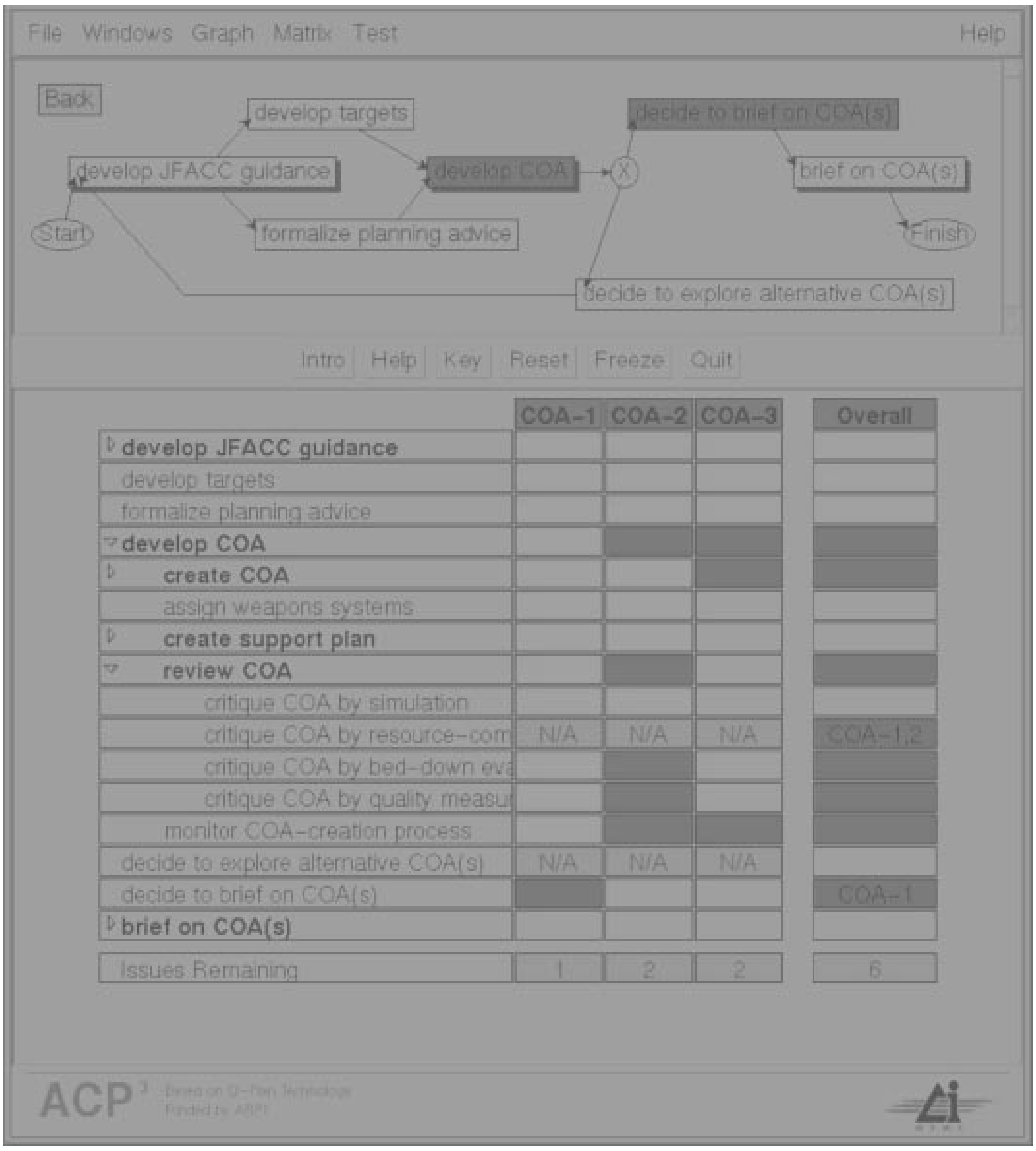
Our aim was to use O-P^3 technology to build a visualisation component for this demonstration which would allow the target end users to view the current state of the planning process in process terms they are familiar with. This has resulted in ACP^3 – the Air Campaign Planning Process Panel.

5.1. Modelling the Planning Process

The software components of the ARPI demonstration can be described as performing activities such as planning, scheduling, simulation and plan evaluation. Going into more detail, we can talk about hierarchical task network planning and Monte Carlo simulation methods. However, end users are more likely to conceive of the processes of Air Campaign Planning in more general, domain-related terms, such as "develop JFACC guidance" and "create support plan". The gaps in terminology and in levels of description can be bridged by building models of the planning process which are rooted in established ACP terminology. We have therefore made use of the previously elicited and verified ACP process models of [Drabble, Lydiard & Tate, 1997] as our source of terminology and as the basis of our IDEF3 models of the planning process for the ARPI demonstration. The full models used for building ACP^3 are described in [Aitken & Tate, 1997].

5.2. Building ACP^3

The ACP^3 viewer is shown in Figure 15.9. The purpose of ACP^3 is to track the overall planning process and display this to the viewers of the ARPI demonstration in a meaningful way using appropriate military process terminology. The planning process is shown in two separate sub-panels. The tabular COA comparison matrix shows COAs being developed (columns) against a tree-based view of the planning process. The graph viewer sub-panel shows the planning process as a PERT network. Since the planning process consists of many nodes with expansions, the graph viewer can only display one individual graph from the planning process for one COA. Other graphs may be reached by clicking on nodes with expansions, and the end user can choose which COA to view.



	COA-1	COA-2	COA-3	Overall
▷ **develop JFACC guidance**				
develop targets				
formalize planning advice				
▽ **develop COA**		▨	▨	▨
▷ **create COA**			▨	▨
assign weapons systems				
▷ **create support plan**				
▽ **review COA**		▨		▨
critique COA by simulation				
critique COA by resource-com	N/A	N/A	N/A	COA-1,2
critique COA by bed-down eva		▨		▨
critique COA by quality measu		▨		▨
monitor COA-creation process		▨	▨	▨
decide to explore alternative COA(s)	N/A	N/A	N/A	
decide to brief on COA(s)	▨			COA-1
▷ **brief on COA(s)**				
Issues Remaining	1	2	2	6

Figure 15.9 The ACP^3 Viewer

The two views are required because the planning process in the ARPI demonstration is a complex artefact. It is possible to see the whole process for every COA in the COA matrix, but information about the partial ordering of the actions in a graph is lost when the graph is converted to a tree structure. The graph viewer shows the full partial ordering but space considerations mean that only a single graph for a single COA can be shown at one time.

The ACP^3 process monitor works by watching for certain KQML messages which it can relate to the status of certain nodes in the ACP process models. As the demonstration proceeds, the status of actions in the model progress from white (not yet ready to execute), to orange (ready to execute), then to green (executing) and finally blue (complete). The final column in the COA matrix is labelled "overall" and summarises the overall status of the COA creation and evaluation process.

The panel is written entirely in Java to form the basis for future Web-based process editors and activity control panels.

6. Application 3 – WOPlan

The aim of this application [Nixon, 2000] ,[Nixon, Levine & Tate, 2000] was to create a mobile, limited media interface onto the O-Plan system. What was envisaged was the case of the mobile human agent, equipped with a small, hand-held wireless device, attempting to access a planning server in order to request some kind of course of action dependent on that user's current situation. Available web-based demonstrations of O-Plan [Tate, Dalton & Levine, 1998] ,[Levine, Tate & Dalton, 2000] propose problem domains involving various military disaster relief and evacuation operations, and it was thought that a mobile telephone or *Personal Digital Assistant* (PDA) could be a tool for plan delivery to mobile units in such a situation. Alternatively, a lone mobile user could access O-Plan to retrieve a plan to assist in a situation in which that user had insufficient experience. Someone who had no experience of engineering, for example, could retrieve a checklist to perform in the event of their car breaking down. The utility of such a system would depend not only on the design of the system itself, but also on the identification of a suitable problem domain, in particular a domain in which the users of the planner are likely to be on the move and in need of a course of action to solve an immediate task, and otherwise with no access to more conventional interfaces such as PCs. The name *WOPlan* (for "Wireless O-Plan") was given to the system.

The design of a mobile interface onto O-Plan is made more difficult by the limited screen sizes of mobile devices, especially in the case of the mobile telephone. Development of interfaces onto O-Plan to date has allowed for the luxury of a full-sized terminal screen [Tate, Dalton & Levine, 1998] , [Levine, Tate & Dalton, 2000]. Some issues of human-computer interaction which may not be critical when using a full-sized colour terminal interface become problematic in the case of the limited media interface. Generally users of mobile devices expect their interaction with the device to be brief, whereas a user sitting down at a workstation is prepared for a more prolonged session. Browsing with a mobile device, especially with a mobile telephone, is (with current devices) slow and cumbersome; data entry is difficult and should be kept to a minimum. A mobile telephone system needs only be slightly poorly designed to be rendered unusable; this is especially true if the system is attempting to serve long lists of data (such as delivering a plan description), as long pages increase download times and make navigation even slower and more frustrating.

WOPlan was developed as a web application which communicates with an instance of the core O-Plan engine and delivers *Wireless Mark-up Language* (WML) to a connected client. The client may be any device with has a browser which conforms to the *Wireless Application Protocol* (WAP), such as a WAP-enabled mobile telephone [7]. The WOPlan web application is a *Java Servlet*. In development and testing the *Nokia WAP Toolkit* WML browser emulator [8] was used in place of a physical WAP device, and the servlet was hosted within the *Tomcat Jakarta* webserver [9]. The client initiates a session by connecting to WOPlan, which connects to the O-Plan server and initialises the service, and provides the user (on the WAP device) with a list of available problem domains. The user is prompted to choose a planning *domain* (defined as a *Task Formalism* file), then choose a *task* within that domain, and then to view, execute or evaluate the resulting *plan*, or to get a different plan that fulfils the same specified task.

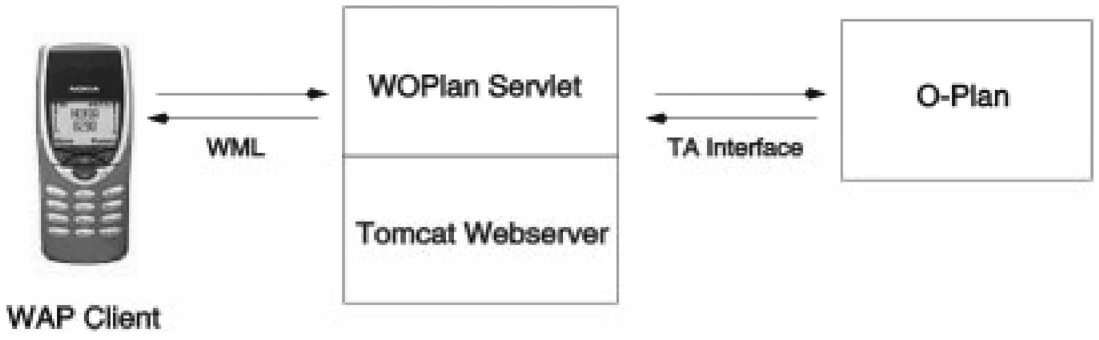


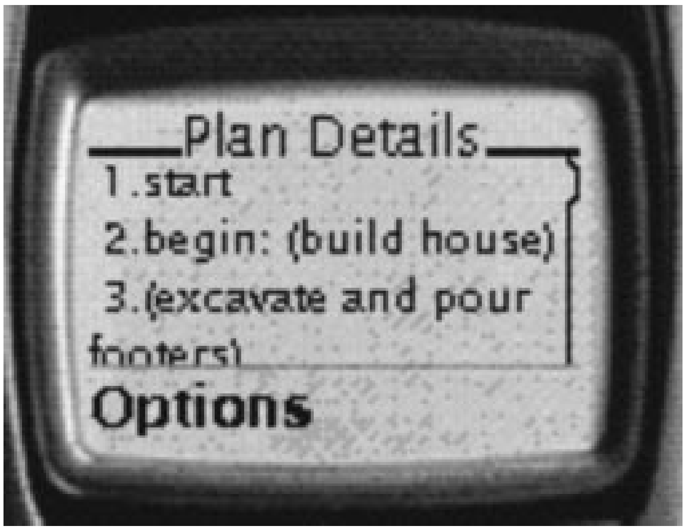
Figure 15.10 Architecture of WOPlan

The architecture of the WOPlan system is shown in Figure 15.10. The *WAP Client* is the component with which the WOPlan user interacts. The user activates WOPlan by initiating an internet session on the WAP Client and navigating to the internet address of the second component, the WOPlan servlet. The WAP client could be any device with a WAP-enabled browser and internet connectivity, although WOPlan has only been tested in use with WAP emulator browsers running on a workstation. Features to notice are the very limited screen size (only four lines of text), and the user interface objects. Directly below the screen are two arrow buttons (used for scrolling up and down a WML page), in between which is a single Select button (used for selecting whatever item is currently highlighted in the WML page). Below and left of the screen is the Options button, which when available and selected should display a context-sensitive list of options. Below and right of the screen is the Back button, which when available and selected should navigate the user back to the previous screen.

The *WOPlan Servlet* sits between the WAP Client and the core O-Plan system. It accepts WAP requests from the client (or from multiple clients simultaneously) and communicates with O-Plan, initially connecting to O-Plan as required and sending and receiving messages through the the standard O-Plan Task Assignment interface [10]. The development work for the WOPlan system has focused on the WOPlan servlet; it is in this component that the logic specific to this implementation resides. The servlet dynamically creates WML pages, depending on the responses it is receiving from O-Plan, which are then sent to the WAP Client for browsing. These WML pages may themselves contain logic such as navigational directives or actions to perform after a certain length of time has passed. Although these directives are executed by the WAP Client, their source is the servlet.

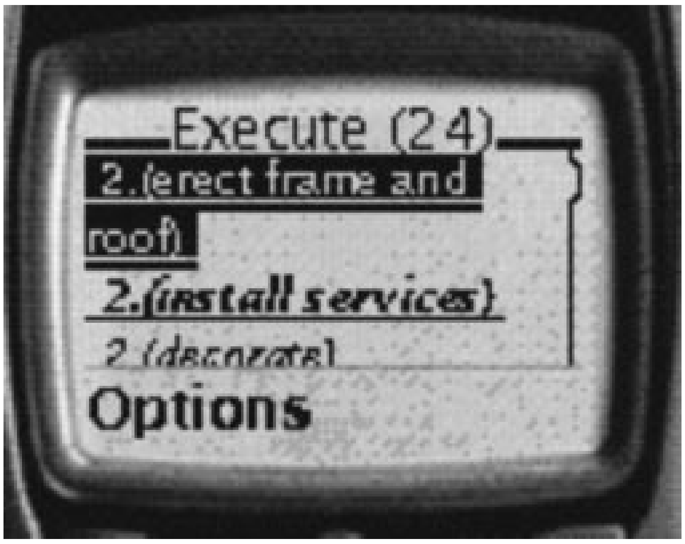
The *O-Plan Server* sits in the bottom tier of the architecture, responding to requests from the WOPlan servlet.

In user trials [Nixon, 2000], it was found that WOPlan provides reasonably stable, scalable and usable access to the O-Plan system through a mobile telephone. Although it does not provide all of the functionality which O-Plan, and in particular the standard Task Assignment interface, has to offer, it provides a useful subset of this functionality, and has addressed the core issues of plan review and execution through the narrative and execution facilities (shown in Figure 15.11 and Figure 15.12).



Plan Details
1.start
2.begin: (build house)
3.(excavate and pour footers)
Options

Figure 15.11 Example of WOPlan Narrative Display



Execute (24)
2.(erect frame and roof)
2.(install services)
2.(decorate)
Options

Figure 15.12 Example of WOPlan Execution Display

The investigation into the possible use of properties specific to mobile devices was a secondary aim of the project. Two such properties which have been discussed are voice technology and mobile positioning technology. No provision for voice or location integration exists in currently available WAP devices, although they will certainly become available in the near future. One possibility for the former would involve the use of *VoiceXML*, an XML variant intended to "make Internet content and information accessible via voice and phone". VoiceXML has the backing of industry giants IBM, AT\&T and Motorola. The Motorola *Mobile ADK* is a development environment for integrating VoiceXML and WML services [11]. The provision of *Location Services* (LCS) as a standard for mobile devices is still currently at the design stage. It is likely that some kind of service based on *Global Positioning System* (GPS) technology will be available to GSM (*Global System for Mobile Communications* – the current European standard) telephones in the near future.

The execution facility provided with this version of WOPlan is little more than a prototype, but it offers interesting possibilities for further development and research. Firstly it could certainly be improved, augmented and made more usable within the context of the WOPlan system. Perhaps more importantly, however, its simple, ordered, one-dimensional format, with action items emphasised according to what may done with those items, could provide a basic template for any system with a mobile limited media interface which is attempting to deliver courses of action (COAs) to human agents on the move.

Conclusions

In this paper, we have argued that the World Wide Web should be seen as a place for "doing things" as well as "finding things". In order to do this, we need shared models and ontologies to represent plans, processes, activities, tasks and issues. We have described work towards this aim, concentrating on the <I-N-OVA> constraint model of activity and the <I-N-CA> constraint model of synthesised artefacts. These are designed to relate strengths from a number of different communities: the AI planning community with both its theoretical and practical system building interests; the issue-based design community, those interested in formal ontologies for processes and products; the standards community; those concerned with new opportunities in task achieving agents on the world wide web; etc. We have described three web-based applications which use such models and have been implemented to "do things" on the Web: the O-Plan Web demonstration, the Air Campaign Planning Process Panel (ACP^3), and O-Plan use via a WAP phone – WOPlan. In the future, we envisage many more such applications, with the possibility that the individual planning applications can communicate with each other using the <I-N-OVA> issue-based constraint-based models of activity described in this paper.

Acknowledgements

The O-Plan and I-X projects are sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory Command and Control Directorate under grant number F30602-99-1-0024 and the UK Defence Evaluation Research Agency (DERA). The US Government, DERA and the University of Edinburgh are authorised to reproduce and distribute reprints for their purposes notwithstanding any copyright annotation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing official policies or endorsements, either express or implied, of DARPA, the Air Force Research Laboratory, the US Government, DERA, or the University of Edinburgh.

References

[Aitken & Tate, 1997].Aitken, S. and Tate, A. (1997). Process Modelling of the TIE 97-1 Demonstration: Modelling Complex Techniques Using ACP Terminology. ISAT Technical Report ISAT-AIAI/TR/6, Version 1, December 1997.

[Currie & Tate, 1991] Currie, K.W. and Tate, A. (1991). O-Plan: the Open Planning Architecture. Artificial Intelligence Vol. 51, No. 1, Autumn 1991, North-Holland.

[Drabble, Lydiard & Tate, 1997] Drabble, B., Lydiard, T. and Tate, A. (1997). Process Steps, Process Product and System Capabilities. ISAT Technical Report ISAT-AIAI/TR/4, Version 2, April 1997.

[Finin, Labrou & Mayfield, 1997] Finin, T., Labrou, Y. and Mayfield, J. (1997). KQML as an Agent Communication Language. In Bradshaw, J. (ed.) Software Agents, 291–316. MIT Press, Cambridge, MA.

[Khambhampati & Srivastava, 1996] Khambhampati, S. and Srivastava, B. (1996). Unifying Classical Planning Approaches. Arizona State University ASU CSE TR 96-006, July 1996.

[Levine, Tate & Dalton, 2000] Levine, J., Tate A. and Dalton, J. (2000). O-P^3: Supporting the Planning Process using Open Planning Process Panels. IEEE Intelligent Systems Vol. 15, No. 5, September/October 2000, 56–62.

[Nixon, 2000] Nixon, A. (2000). Limited Media Interface for AI Planning System. M.Sc. project dissertation, Division of Informatics, University of Edinburgh, September 2000.

[Nixon, Levine & Tate, 2000] Nixon, A., Levine, J. and Tate, A. (2000). Limited Media Interface for AI Planning System. Proceedings of the 19th Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG 2000), Open University, Milton Keynes, UK.

[Polyak & Tate, 1998] Polyak, S. and Tate, A. (1998). A Common Process Ontology for Process-Centred Organisations. Knowledge Based Systems, to appear. Earlier version published as University of Edinburgh Department of Artificial Intelligence Research paper 930, 1998.

[Reece et al., 1993] Reece, G.A., Tate, A., Brown, D. and Hoffman, M. (1993). The PRECiS Environment. Paper presented at the ARPA-RL Planning Initiative Workshop at AAAI-93, Washington D.C., July 1993.

[Tate, 1993] Tate, A. (1993). Authority Management – Coordination between Planning, Scheduling and Control. Workshop on Knowledge-based Production Planning, Scheduling and Control at the International Joint Conference on Artificial Intelligence (IJCAI-93), Chambery, France, 1993.

[Tate, 1994] Tate, A. (1994). Mixed Initiative Planning in O-Plan2. Proceedings of the ARPA/Rome Laboratory Planning Initiative Workshop, Tucson, Arizona, USA. Morgan Kaufmann, Palo Alto, CA.

[Tate, Drabble and Dalton, 1994] Tate, A., Drabble, B. and Dalton, J. (1994). Reasoning with Constraints within O-Plan2. Proceedings of the ARPA/Rome Laboratory Planning Initiative Workshop, Tucson, Arizona, USA. Morgan Kaufmann, Palo Alto, CA.

[Tate, 1996a] Tate, A. (ed.) (1996a). Advanced Planning Technology - Technological Achievements of the ARPA/Rome Laboratory Planning Initiative (ARPI). AAAI Press, Menlo Park, CA.

[Tate, 1996b] Tate, A. (1996b). Representing Plans as a Set of Constraints – the <I-N-OVA> Model. Proceedings of the Third International Conference on Artificial Intelligence Planning Systems (AIPS-96), 221–228. AAAI Press, Menlo Park, CA.

[Tate, Drabble and Dalton, 1996] Tate, A., Drabble, B. and Dalton, J. (1996). O-Plan: a Knowledge-Based Planner and its Application to Logistics. In Tate, A. (1996a), 259–266.

[Tate, 1998] Tate, A. (1998). Roots of SPAR - Shared Planning and Activity Representation. Knowledge Engineering Review Vol. 13, No. 1, March 1998. Cambridge University Press, Cambridge, UK. See also http://www.aiai.ed.ac.uk/project/spar/

[Tate, Dalton & Levine, 1998] Tate, A., Dalton, J. and Levine, J. (1998). Generation of Multiple Qualitatively Different Plan Options. Proceedings of Fourth International Conference on AI Planning Systems (AIPS-98), Pittsburgh, USA, 27–34. AAAI Press, Menlo Park, CA.

Footnotes

[1]Artificial Intelligence Applications Institute, Division of Informatics, The University of Edinburgh, 80 South Bridge, Edinburgh EH1 1HN, UK, {a.tate, j.dalton, j.levine}@ed.ac.uk, http://www.aiai.ed.ac.uk/project/ix/

[2]http://www.aiai.ed.ac.uk/project/spar/

[3]<I-N-OVA> is pronounced as in "Innovate".

[4]Temporal (or spatio-temporal) and object constraints are cross-constraints specific to the planning task. The cross-constraints in some other domain may be some other constraint type.

[5]I-X is the successor project to O-Plan – see http://www.aiai.ed.ac.uk/project/ix/

[6]You can try this demonstration out yourself at http://www.aiai.ed.ac.uk/project/oplan/

[7]See http://www.wapforum.org/

[8]See http://www.forum.nokia.com/wapforum/

[9]See http://jakarta.apache.org/tomcat/

[10]This is the standard API provided for external programs to communicate with O-Plan.

[11]See http://www.motorola.com/spin/mix/faqs.html

List of Figures