An inference engine for RDF

Master thesis

G.Naudts
831708907

22 augustus 2003

Open University Netherlands

Agfa Gevaert

This document is the Master Thesis made as a part of my Master Degree in Computer Science Education (Software Systems) at the Open University of the Netherlands. The work has been done in collaboration with the research department of the company Agfa in Mortsel Belgium.

Student data

Name	Guido Naudts
Student number	831708907
Address	Secretarisdreef 5 2288 Bouwel
Telephone work	0030-2-542.76.01
Home	0030-14-51.32.43
E-mail	Naudts_Vannoten@yahoo.com

Coaching and graduation committee

Chairman:	dr J.T. Jeuring, professor at the Open University
Secretary :	ir. F.J. Wester, senior lecturer at the Open University
Coach:	ir. J. De Roo, researcher at Agfa Gevaert.

Acknowledgements

I want to thank Ir.J.De Roo for giving me the opportunity for making a thesis about a tremendous subject and his guidance in the matter.
Pr.J.Jeuring and Ir. F.Wester are thanked for their efforts to help me produce a readable and valuable text.
I thank M.P.Jones for his Prolog demo in the Hugs distribution that has very much helped me.
I thank all the people from the OU for their efforts during the years without which this work would not have been possible.
I thank my wife and children for supporting a father seated behind his computer during many years.

Summary

The Semantic Web is an initiative of the *World Wide Web Consortium* (W3C). The goal of this project is to define a series of standards. These standards will create a framework for the automation of activities on the *World Wide Web* (WWW) that still need manual intervention by human beings at this moment.
A certain number of basic standards have been developed already. Among them XML is without doubt the most known. Less known is the *Resource Description Framework* (RDF). This is a standard for the creation of meta information. This is the information that has to be given to computers to permit them to handle tasks previously done by human beings.
Information alone however is not sufficient. In order to take a decision it is often necessary to follow a certain reasoning. Reasoning is connected with logics. The consequence is that computers have to be fed with programs and rules that can take decisions, following a certain logic, based on available information.
These programs and reasonings are executed by *inference engines*. The definition of inference engines for the Semantic Web is at the moment an active field of experimental research.
These engines have to use information that is expressed following the standard RDF. The consequences of this requirement for the definition of the engine are explored in this thesis.
An inference engine uses information and, given a *query*, reasons with this information with a *solution* as a result. It is important to be able to show that the solutions, obtained in this manner, have certain characteristics. These are, between others, the characteristics *completeness* and *validity.* This proof is delivered in this thesis in two ways:

1) by means of a reasoning based on the graph theoretical properties of RDF
2) by means of a model based on first order logics.

On the other hand these engines must take into account the specificity of the World Wide Web. One of the properties of the web is the fact that sets are not closed. This implies that not all elements of a set are known. Reasoning then has to happen in an *open world.* This fact has far reaching logical implications.
During the twentieth century there has been an impetous development of logic systems. More than 2000 kinds of logic were developed. Notably *first order logic* (FOL) has been under heavy attack, beside others, by the dutch Brouwer with the *constructive* or *intuistionistic* logic. Research is also done into kinds of logic that are suitable for usage by machines.
In this thesis I argue that an inference engine for the World Wide Web should follow a kind of constructive logic and that RDF, with a logical

implication added, satisfies this requirement. It is shown that a constructive approach is important for the verifiability of the results of inferencing.

The structure of the World Wide Web has a number of consequences for the properties that an inference engine should satisfy. A list of these properties is established and discussed.
A query on the World Wide Web can extend over more than one server. This means that a process of subquerying must be defined. This concept is introduced in an experimental inference engine, RDFEngine.
An important characteristic for an engine that has to be active in the World Wide Web is *efficiency*. Therefore it is important to do research about the methods that can be used to make an efficient engine. The structure has to be such that it is possible to work with huge volumes of data. Eventually these data are kept in relational databases.
On the World Wide Web there is information available in a lot of places and in a lot of different forms. Joining parts of this information and reasoning about the result can easily lead to the existence of *contradictions* and *inconsistencies*. These are inherent to the used logic or are dependable on the application. A special place is taken by inconsistencies that are inherent to the used ontology. For the Semantic Web the ontology is determined by the standards *rdfs* and *OWL*.
An ontology introduces a classification of data and applies restrictions to those data. An inference engine for the Semantic Web needs to have such characteristics that it is compatible with rdfs and OWL.
An executable specification of an inference engine in Haskell was constructed. This permitted to test different aspects of inferencing and the logic connected to it. This engine has the name RDFEngine. A large number of existing testcases was executed with the engine. A certain number of testcases was constructed, some of them for inspecting the logic characteristics of RDF.

Samenvatting

Het Semantisch Web is een initiatief van het *World Wide Web Consortium* (W3C). Dit project beoogt het uitbouwen van een reeks van standaarden. Deze standaarden zullen een framework scheppen voor de automatisering van activiteiten op het *World Wide Web* (WWW) die thans nog manuele tussenkomst vereisen.
Een bepaald aantal basisstandaarden zijn reeds ontwikkeld waaronder XML ongetwijfeld de meest gekende is. Iets minder bekend is het *Resource Description Framework* (RDF). Dit is een standaard voor het creëeren van meta-informatie. Dit is de informatie die aan de machines moet verschaft worden om hen toe te laten de taken van de mens over te nemen.
Informatie alleen is echter niet voldoende. Om een beslissing te kunnen nemen moet dikwijls een bepaalde redenering gevolgd worden. Redeneren heeft te maken met logica. Het gevolg is dat computers gevoed moeten worden met programmas en regels die, volgens een bepaalde logica, besluiten kunnen nemen aan de hand van beschikbare informatie.
Deze programmas en redeneringen worden uitgevoerd door zogenaamde *inference engines*. Dit is op het huidig ogenblik een actief terrein van experimenteel onderzoek.
De engines moeten gebruik maken van de informatie die volgens de standaard RDF wordt weergegeven. De gevolgen hiervan voor de structuur van de engine worden in deze thesis nader onderzocht.
Een inference engine gebruikt informatie en, aan de hand van een *vraagstelling*, worden redeneringen doorgevoerd op deze informatie met een *oplossing* als resultaat. Het is belangrijk te kunnen aantonen dat de oplossingen die aldus bekomen worden bepaalde eigenschappen bezitten. Dit zijn onder meer de eigenschappen *volledigheid* en *juistheid*. Dit bewijs wordt op twee manieren geleverd: enerzijds bij middel van een redenering gebaseerd op de graaf theoretische eigenschappen van RDF; anders bij middel van een op first order logica gebaseerd model.
Anderzijds dienen zij rekening te houden met de specificiteit van het World Wide Web. Een van de eigenschappen van het web is het open zijn van verzamelingen. Dit houdt in dat van een verzameling niet alle elementen gekend zijn. Het redeneren dient dan te gebeuren in een *open wereld.* Dit heeft verstrekkende logische implicaties.
In de loop van de twintigste eeuw heeft de logica een stormachtige ontwikkeling gekend. Meer dan 2000 soorten logica werden ontwikkeld. Aanvallen werden doorgevoerd op het bolwerk van de *first order logica* (FOL), onder meer door de nederlander Brouwer met de *constructieve* of *intuistionistische* logica. Gezocht wordt ook naar soorten logica die geschikt zijn voor het gebruik door machines.

In deze thesis wordt betoogd dat een inference engine voor het WWW een constructieve logica dient te volgen enerzijds, en anderzijds, dat RDF aangevuld met een logische *implicatie* aan dit soort logica voldoet. Het wordt aangetoond that een constructieve benadering belangrijk is voor de verifieerbaarheid van de resultaten van de inferencing.
De structuur van het World Wide Web heeft een aantal gevolgen voor de eigenschappen waaraan een inference engine dient te voldoen. Een lijst van deze eigenschappen wordt opgesteld en besproken.
Een query op het World Wide Web kan zich uitstrekken over meerdere servers. Dit houdt in dat een proces van subquerying moet gedefinieerd worden. Dit concept wordt geïntroduceerd in een experimentele inference engine, RDFEngine.

Een belangrijke eigenschap voor een engine die actief dient te zijn in het World Wide Web is *efficientie*. Het is daarom belangrijk te onderzoeken op welke wijze een efficiente engine kan gemaakt worden. De structuur dient zodanig te zijn dat het mogelijk is om met grote volumes aan data te werken, die eventueel in relationele databases opgeslagen zijn.
Op het World Wide Web is informatie op vele plaatsen en in allerlei vormen aanwezig. Het samen voegen van deze informatie en het houden van redeneringen die erop betrekking hebben kan gemakkelijk leiden tot het ontstaan van *contradicties* en *inconsistenties*. Deze zijn inherent aan de gebruikte logica of zijn afhankelijk van de applicatie. Een speciale plaats nemen de inconsistenties in die inherent zijn aan de gebruikt ontologie. Voor het Semantisch Web wordt de ontologie bepaald door de standaarden *rdfs* en *OWL*. Een ontologie voert een classifiering in van gegevens en legt ook beperkingen aan deze gegevens op. Een inference engine voor het Semantisch Web dient zulkdanige eigenschappen te bezitten dat hij compatibel is met rdfs en OWL.
Een uitvoerbare specificatie van een inference engine in Haskell werd gemaakt. Dit liet toe om allerlei aspecten van inferencing en de ermee verbonden logica te testen. De engine werd RDFEngine gedoopt. Een groot aantal bestaande testcases werd onderzocht. Een aantal testcases werd bijgemaakt, sommige speciaal met het oogmerk om de logische eigenschappen van RDF te onderzoeken.

Chapter 1. Introduction

1.1. Overview

In chapter 1 a case study is given and then the goals of the research that is the subject of the thesis are exposed with the case study as illustration. The methods used are explained and an overview is given of the fundamental knowledge upon which the research is based. The relation with current research is indicated. Then the structure of the thesis is outlined.

1.2. Case study

1.2.1. Introduction

A case study can serve as a guidance to what we want to achieve with the Semantic Web. Whenever standards are approved they should be such that important case studies remain possible to implement. Discussing all possible application fields here would be out of scope. However one case study will help to clarify the goals of the research.

1.2.2. The case study

A travel agent in Antwerp has a client who wants to go to St.Tropez in France. There are rather a lot of possibilities for composing such a voyage. The client can take the train to France, or he can take a bus or train to Brussels and then the airplane to Nice in France, or the train to Paris then the airplane or another train to Nice. The travel agent explains the client that there are a lot of possibilities. During his explanation he gets an impression of what the client really wants. Fig.1.1. gives a schematic view of the case study.

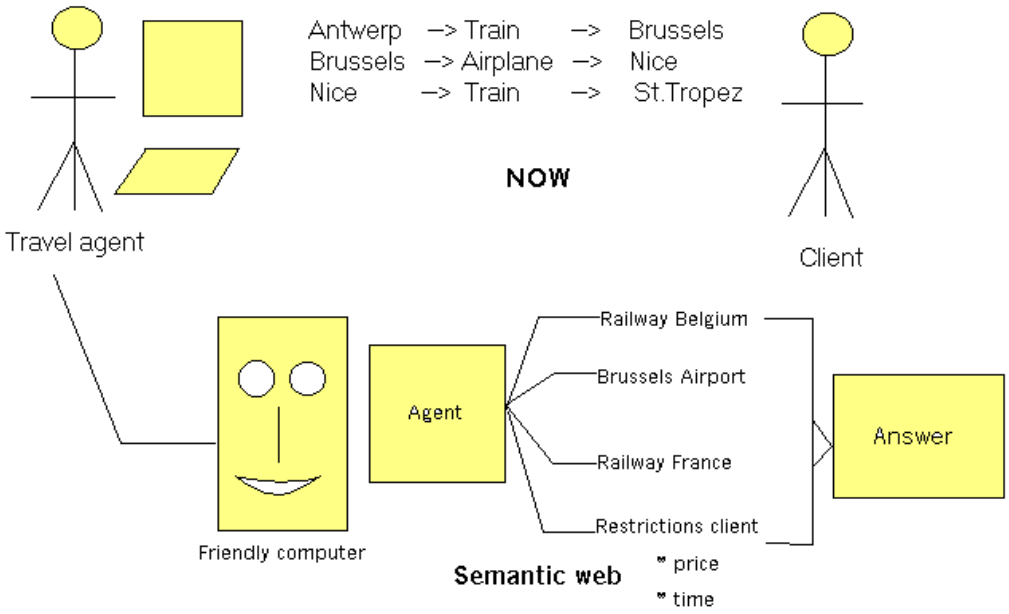

Fig.1.1. A Semantic Web case study

The travel agent agrees with the client about the itinerary: by train from Antwerp to Brussels, by airplane from Brussels to Nice and by train from Nice to St. Tropez. This still leaves room for some alternatives. The client will come back to make a final decision once the travel agent has said him by mail that he has worked out some alternative solutions like price for first class vs second class etc...
Remark that the decision for the itinerary that has been taken is not very well founded; only very crude price comparisons have been done based on some internet sites that the travel agent consulted during his conversation with the client. A very cheap flight from Antwerp to Cannes has escaped the attention of the travel agent.
The travel agent will now further consult the internet sites of the Belgium railways, the Brussels airport and the France railways to get some alternative prices, departure times and total travel times.

Now let's compare this with the hypothetical situation that a full blown Semantic Web would exist. In the computer of the travel agent resides a Semantic Web agent that has at its disposal all the necessary standard tools. The travel agent has a specialised interface to the general Semantic Web agent. He fills in a query in his specialised screen. This query is translated to a standardised query format for the Semantic Web agent. The agent consult his rule database. This database of course contains a lot of rules about travelling as well as facts like e.g. facts about internet sites

where information can be obtained. There are a lot of 'path' rules: rules for composing an itinerary (for an example of what such rules could look like see: [GRAPH]. The agent contacts different other agents like the agent of the Belgium railways, the agents of the French railways, the agent of the airports of Antwerp, Brussels, Paris, Cannes, Nice etc...
With the information recieved its inference rules about scheduling a trip are consulted. This is all done while the travel agent is chatting with the client to detect his preferences. After some 5 minutes the Semantic Web agent gives the travel agent a list of alternatives for the trip; now the travel agent can immediately discuss this with his client. When a decision has been reached, the travel agent immediately gives his Semantic Web agent the order for making the reservations and ordering the tickets. Now the client only will have to come back once for getting his tickets and not twice. The travel agent not only has been able to propose a cheaper trip as in the case above but has also gained an important amount of time.

1.2.3.Conclusions of the case study

That a realisation of a similar system is interesting is evident. Clearly, the standard tools do have to be very flexible and powerful to be able to put into rules the reasoning of this case study (path determination, itinerary scheduling). All these rules then have to be made by someone. This can of course be a common effort for a lot of travel agencies.
What exists now? A quick survey learns that there are web portals where a client can make reservations (for hotel rooms). However the portal has to be fed with data by the travel agent. There also exist software that permits the client to manage his travel needs. But all that software has to be fed with information obtained by a variety of means, practically always manually.
A partial simulation namely the order of a ticket for a flight can be found in 5.16.5.

1.3.Research goals

1.3.1. Standards

In the case study different information sites have to communicate with another. If this has to be done in an automatic way a standard has to be used. The standard that is used for expressing information is RDF. RDF is a standard for expressing meta-information developed by the W3C. The primary purpose is adding meta-information to web pages so that these become more usable for computers [TBL]. Later the Semantic Web initiative was launched by Berners-Lee with the purpose of developing standards for automating the exchange of information between computers.

The information standard is again RDF [DESIGN]. However other standards are added on top of RDF for the realisation of the Semantic Web (chapter 2).

1.3.2. Research questions

The following research questions were defined:

1) Define an inference process on top of RDF and give a specification of this process in a functional language.

   In the case study the servers dispose of information files that contain facts and rules. The facts are expressed in RDF. The syntax of the rules is RDF also, but the semantic interpretation is different. Using rules implies inferencing. On top of RDF a standardized inference layer has to be defined. A functional language is very well suited for a declarative specification. At certain points the inference process has to be interrupted to direct queries to other sites on the World Wide Web. This implies a process of *subquerying* where the inference process does not take place in one site (one computer) but is distributed over several sites (see also fig. 1.1).

2) Which kind of logic is best suited for the Semantic Web?

   Using rules; making queries; finding solutions; this is the subject of logic. So the relevance of logic for an inferencing standard based on top of RDF has to be investigated.

3) Is the specified inference process sound and complete?

   It is without question of course that the answers to a query have to be valid. By completeness is meant that all answers to a query are found. It is not always necessary that all answers are found but often it is.

4) What can be done for augmenting the efficiency of the inference process?

   A basic internet engine should be fast because the amount of data can be high; it is possible that data need to be collected from several sites over the internet. This will already imply an accumulation of transmission times. A basic engine should be optimized as much as possible.

5) How can inconsistencies be handled by the inference process?

> In a closed world (reasoning on one computer system) inconsistencies can be mastered; on the internet, when inferencing becomes a distributed process, inconsistencies will be commonplace.

As of yet no standard is defined for inferencing on top of RDF. As will be shown in this thesis there are a lot of issues involved and a lot of choices to be made. The definition of such a standard is a necessary, but difficult step in the progress of the Semantic Web. In order to be able to define a standard a definition has to be provided for following notions that represent minimal extensions on top of RDF: rules, queries, solutions and proofs.

1.4. Research methods

The primary research method consists of writing an implementation of an RDF inference engine in Haskell. The engine is given the name RDFEngine. The executable specification is tested using test cases where the collection of test cases of De Roo [DEROO] plays an important role. Elements of logic, efficiency and inconsistency can be tested by writing special test cases in interaction with a study of the relevant literature.

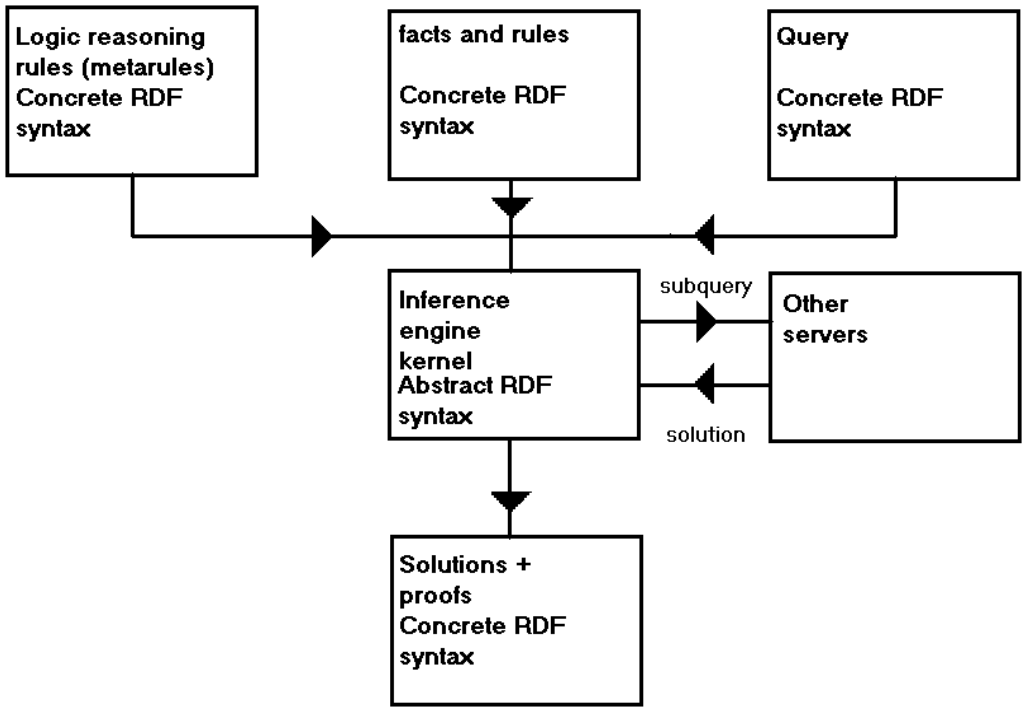

Fig.1.2. The inference engine RDFEngine in a server with its inputs and outputs.

In fig.1.2 the global structure of the inference engine is drawn. The inputs and outputs are in a concrete RDF syntax. Notation 3 (N3) and RDFProlog are used by RDFEngine. RDFProlog was developed with no special purpose, but often I found it more convenient to use than Notation 3.
The engine is given a modular structure and different versions exist for different testing purposes.
For making a Haskell implementation the choice was made for a resolution based engine. A forward reasoning engine could have been chosen as well but a resolution based engine is more goal directed and has better characteristics for subquerying.
Other used methods are:

- The Haskell specification describes the abstract syntax and the data structures used by RDFEngine.
- The study of aspects of inferencing that are specific to the World Wide Web. These aspects include the problems related to the notions open/closed world, distributed inferencing, verifiability of results and inconsistencies.
- The graph description of the inferencing process

1.5. The foundations

The research used amongst others the following material:

a) The collection of test cases of De Roo [DEROO]
b) The RDF model theory [RDFM] [RDFMS]
c) The RDF rules mail list [RDFRULES]
d) The classic resolution theory [VANBENTHEM] [STANFORD]
e) Logic theories and mainly first order logic, constructive logic and paraconsistent logic. [VANBENTHEM, STANFORDP]

1.6. Related work

The mail list RDF rules [RDFRULES] has as a purpose to discuss the extension of RDF with inferencing features, mainly rules, queries and answers. There are a lot of interesting discussions but no clear picture is emerging at the moment.
Berners-Lee [DESIGN] seeks inspiration in first order logic. He defines logic predicates (chapter 2), builtins, test cases and a reference engine CWM. However the semantics corresponding to the input and output formats for a RDF engine are not clearly defined.
A common approach is to use existing logic systems. RDF is then expressed in terms of these systems and logic (mostly rules) is added on top e.g. Horn logic or frame logic. An example is found in [DECKER]. In this case the semantics are those of the used logic.

The implementation of an ontology like OWL (Ontology Web Language) implies also inferencing facilities that are necessary for implementing the semantics of the ontology [OWL features].
Euler [DEROO] and CWM take a graph oriented approach for the construction of the inference program. The semantics are mostly defined by means of test cases.
The approach taken by my research for defining the semantics of the inference input and output has two aspects:

1) It is graph oriented and, in fact, the theory is generally applicable to labeled graphs.
2) From the logic viewpoint a constructive approach is taken for reasons of verifiability (chapter 4).
3) A constructive graph oriented definition is given for rules, queries, solutions and proofs.

1.7. Outline of the thesis

Chapter two gives an introduction into the basic standards that are used by RDF and into the syntax and model theory of RDF.

Chapter three discusses related work. It consists of three parts: automated reasoning, logic and existing software systems.

Chapter four explores in depth the relationship of RDF inferencing with logic.
Here the accent is put on a constructive logic interpretation.

Chapter five exposes the meaning of inferencing using an RDF facts database. The Haskell specification, RDFEngine, is explained and a theory of resolution based on reasoning on a graph is presented. Some applications are reviewed.

Chapter six discusses optimalization techniques. Especially important is a technique based on the theory exposed in chapter five.

Chapter seven disusses inconsistencies that can arise during the course of inferencing processes on the Semantic Web.

Chapter eight finally gives a conclusion and indicates possible ways for further research.

Chapter 2. Preliminaries

2.1. Introduction

In this chapter a number of techniques are briefly explained. These techniques are necessary building blocks for the Semantic Web. They are also a necessary basis for inferencing on the web.

2.2.XML and namespaces

2.2.1. Definition

XML (Extensible Markup Language) is a subset of SGML (Standard General Markup Language). In its original definition a markup language is a language which is intended for adding information ("markup" information) to an existing document. This information must stay separate from the original hence the presence of separation characters. In SGML and XML 'tags' are used.

2.2.2. Features

There are two kinds of tags: opening and closing tags. The opening tags are keywords enclosed between the signs "<" and ">". An example: <author>. A closing tag is practically the same, only the sign "/" is added e.g. </author>. With these elements alone very interesting datastructures can be built. An example of a book description:

```
<book>
  <title>
      The Semantic Web
  </title>
  <author>
      Tim Berners-Lee
  </author>
</book>
```

As can be seen it is quite easy to build hierarchical datastructures with these elements alone. A tag can have content too: in the example the strings "The Semantic Web" and "Tim Berners-Lee" are content. One of the good characteristics of XML is its simplicity and the ease with which parsers and other tools can be built.

The keywords in the tags can have attributes too. The previous example could be written:

<book title="The Semantic Web" author="Tim Berners-Lee"></book>

where attributes are used instead of tags.

XML is not a language but a meta-language i.e. a language with as goal to make other languages ("markup" languages).
Everybody can make his own language using XML. A person doing this only has to use the syntaxis of XML i.e. produce wellformed XML. However more constraints can be added to an XML language by using a DTD or an XML schema. A valid XML document is one that uses XML syntax and respects the constraints laid down in a DTD or XML schema. If everybody creates his own language then the "tower-of-Babylon"-syndrome is looming. How is such a diversity in languages handled? This is done by using *namespaces*. A namespace is a reference to the definition of an XML language.
Suppose someone has made an XML language about birds. Then he could make the following namespace declaration in XML:

<birds:wing xmlns:birds="http://birdSite.com/birds/">

This statement is referring to the tag "wing" whose description is to be found on the site that is indicated by the namespace declaration xmlns (= XML namespace). Now our hypothetical biologist might want to use an aspect of the physiology of birds described however in another namespace:

<physiology:temperature xmlns:physiology="
http://physiology.com/xml/">

By the semantic definition of XML these two namespaces may be used within the same XML-object.

```
<?xml version="1.0" ?>
<birds:wing xmlns:birds="http://birdSite.com/birds/">
        large
</birds:wing>
<physiology:temperature xmlns:physiology="
http://physiology.com/xml/">
        43
</physiology:temperature>
```

2.3.URI's and URL's

2.3.1. Definitions

URI stands for **U**niform **R**esource **I**ndicator. A URI is anything that indicates unequivocally a resource.
URL stands for **U**niform **R**esource **L**ocator. This is a subset of URI. A URL indicates the access to a resource. URN refers to a subset of URI and indicates names that must remain unique even when the resource ceases to be available. URN stands for **U**niform **R**esource **N**ame. [ADDRESSING]
In this thesis only URL's will be used.

2.3.2. Features

The following example illustrates a URL format that is in common use. [URI]:

http:// www.math.uio.no/faq/compression-faq/part1.html

Most people will recognize this as an internet address. It unequivocally identifies a web page. Another example of a URI is a ISBN number that unquivocally identifies a book.

The general format of an http URL is:

http://<host>:<port>/<path>?<searchpart>.

The host is of course the computer that contains the resource; the default port number is normally 80; eventually e.g. for security reasons it might be changed to something else; the *path* indicates the directory access path. The *searchpart* serves to pass information to a server e.g. data destinated for CGI-scripts.
When an URL finishes with a slash like *http://www.test.org/definitions/* the directory *definitions* is addressed. This will be the directory defined by adding the standard prefix path e.g. */home/netscape* to the directory name: */home/netscape/definitions*.The parser can then return e.g. the contents of the directory or a message 'no access' or perhaps the contents of a file 'index.html' in that directory.
A path might include the sign "#" indicating a named anchor in an html-document. Following is the html definition of a named anchor:

<H2><A NAME="semantic">The Semantic Web</A></H2>

A named anchor thus indicates a location within a document. The named anchor can be called e.g. by:

http://www.test.org/definition/semantic.html#semantic

2.4.Resource Description Framework RDF

2.4.1. Introduction

RDF is a language. The semantics are defined by [RDFM]; some concrete syntaxes are: XML-syntax, Notation 3, N-triples, RDFProlog. N-triples is a subset of Notation 3 and thus of RDF [RDFPRIMER]. RDFProlog is also a subset of RDF and is defined in this thesis.
Very basically RDF uses triples: *(subject, predicate, object)*. The predicate is called *the property*. This simple statement however is not the whole story; nevertheless it is a good point to start.
An example [ALBANY] of a statement is:
"Jan Hanford created the J. S. Bach homepage.". The J.S. Bach homepage is a resource. This resource has a URI: *http://www.jsbach.org/*. It has a property: creator with value 'Jan Hanford'. Figure 2.1. gives a graphical view of this.

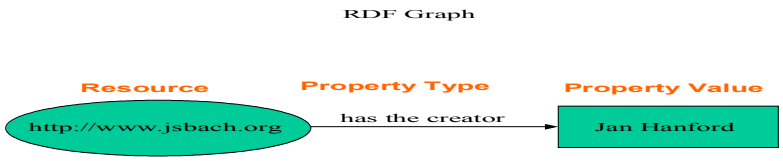

Fig.2.1. An example of a RDF relation.

In simplified RDF this is:

```
<RDF>
  <Description about= "http://www.jsbach.org">
    <Creator>Jan Hanford</Creator>
  </Description>
</RDF>
```

However this is without namespaces meaning that the notions are not well defined. With namespaces added this becomes:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:dc="http://purl.org/DC/">
  <rdf:Description about="http://www.jsbach.org/">
    <dc:Creator>Jan Hanford</dc:Creator>
  </rdf:Description>
</rdf:RDF>
```

The first namespace *xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"* refers to the document describing the (XML-)syntax of RDF; the second namespace *xmlns:dc="http://purl.org/DC/*" refers to the description of the Dublin Core, a basic ontology about authors and publications. This is also an example of two languages that are mixed within an XML-object: the RDF and the Dublin Core language.

In the following example is shown that more than one predicate-value pair can be indicated for a resource.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:bi="http://www.birds.org/definitions/">
  <rdf:Description about= "http://www.birds.org/birds#swallow">
    <bi:wing>pointed</bi:wing>
    <bi:habitat>forest</bi:habitat>
  </rdf:Description>
</rdf:RDF>
```

2.4.2. Verifiability of a triple

RDF triples can have different origins. When site *A* recieves a triple from site *D: (B:s, C:p, C:o)* where *B* and *C* are namespaces, then *A* may wish to verify the trustworthiness of the labels *B:s, C:p* and *C:o.*
Consider the subject of the triple *B:s*. Things that can be verified are:

a) the existence of the namespace *B*.
b) whether the site or owner of namespace *B* is trusted.
c) the notion associated with *s* is defined in namespace *B*.
d) eventually the triple might be signed and the signature of the triple can be verified.

2.4.3. The graph syntax

In the graph representation subjects and objects are *nodes* in the graph and the *arcs* represent properties. No nodes can occur twice in the graph. The consequence is that some triples will be connected with other triples; other triples will have no connections. Also the graph will have different connected subgraphs.

A node can also be an *blank* node which is a kind of anonymous node. A blank node can be *instantiated* with a URI or a literal and then becomes a 'normal' node.

2.4.4.Conclusion

What is RDF? It is a language with a simple semantics consisting of triples: (subject, predicate, object) and some other elements. Several syntaxes exist for RDF: XML, graph, Notation 3, N-triples, RDFProlog. Notwithstanding its simple structure a great deal of information can be expressed with it.

2.5.Notation 3

2.5.1. Introduction

Notation 3 is a syntax for RDF developed by Tim Berners-Lee and Dan Connolly and represents a more human usable form of the RDF-syntax with in principle the same semantics [RDF Primer]. Most of the test cases used for testing RDFEngine are written in Notation 3. An alternative and shorter name for Notation 3 is N3.

2.5.2. Basic properties

The basic syntactical form in Notation 3 is a *triple* of the form : <*subject*> <*property*> <*object*> where subject, property and object are atoms. An atom can be either a URI (or a URI abbreviation) , a blank node or a variable[1]. Subject and object can also be triplelists. There also is some " syntactic sugar".

In N3 URI's can be indicated in a variety of different ways. A URI can be indicated by its complete form or by an abbreviation. These abbreviations have practical importance and are intensively used in the testcases. In what follows the first item gives the complete form; the others are abbreviations.

- <*http://www.w3.org/2000/10/swap/log#log:forAll*> : this is the complete form of a URI in Notation 3. This ressembles very much the indication of a tag in an HTML page.
- ***xxx:*** : a label followed by a ':' is a prefix. A prefix is defined in N3 by the @prefix instruction :
  @prefix ont: <http://www.daml.org/2001/03/daml-ont#>.
  This defines the prefix ont: . After the label follows the URI represented by the prefix.

[1] Variables are an extension to RDF.

So *ont:TransitiveProperty* is in full form *<http://www.daml.org/2001/03/daml-ont#TransitiveProperty>* .

- *<#param>* : Here only the tag in the HTML page is indicated. To get the complete form the default URL must be added.the complete form is : *<URL_of_current_document#param>*.
  The default URL is the URL of the current document i.e. the document that contains the description in Notation 3.
- <> or <#>: this indicates the URI of the current document.
- **:** : a single double point is by convention referring to the current document. However this is not necessarily so because this meaning has to be declared with a prefix statement :
  @prefix **: <#>.**

Two substantial abbreviations for *sets of triples* are property lists and object lists. It can happen that a subject recieves a series of qualifications; each qualification with a verb and an object,
e.g. *:bird :color :blue; height :high; :presence :rare.*
This represents a bird with a blue color, a high height and a rare presence. These properties are separated by a semicolon.
A verb or property can have several values e.g.
:bird :color :blue, yellow, black.
This means that the bird has three colors. This is called an object list. The two things can be combined :

:bird :color :blue, yellow, black ; height :high ; presence :rare.

The objects in an objectlist are separated by a comma.
A semantic and syntactic feature are anonymous subjects. The symbols '[' and ']' are used for this feature. [:can :swim]. means there exists an anonymous subject x that can swim. The abbreviations for propertylist and objectlist can here be used too :

[:can :swim, :fly ; :color :yellow].

The property rdf:type can be abbreviated as "a":

:lion a :mammal.

really means:

:lion rdf:type :mammal.

2.6.The logic layer

In [SWAPLOG] an experimental logic layer is defined for the Semantic Web. I will say a lot more about logic in chapter 4. An overview of the most salient features (RDFEngine only uses log:implies, log:forAll, log:forSome and log:Truth):

log:implies : this is the implication, also indicated by an arrow: →.

{:rat a :rodentia. :rodentia a :mammal.}→ {:rat a :mammal}.

log:forAll : the purpose is to indicate universal variables :

{:a rdfs:subClassoF :b. :b rdfs:subClassOf :c. rdfs:subClassoF a owl:transitiveProperty}→ {:a rdfs:subClassOf :c.}; log:forAll :a, :b, :c.

indicates that *:a, :b* and *:c* are universal variables.

log:forSome: does the same for existential variables:

{:x a :human.}→ {:x a :man}; log:forSome :x.

log:Truth : states that this is a universal truth. This is not interpreted by RDFEngine.

Here follow briefly some other features:
log:falseHood : to indicate that a formula is not true.
log:conjunction : to indicate the conjunction of two formulas.
log:includes : *F* includes *G* means *G* follows from *F*.
log:notIncludes: *F* notIncludes *G* means *G* does not follow from *F*.

2.7.Semantics of N3

2.7.1. Introduction

The semantics of N3 are the same as the semantics of RDF [RDFM]. The semantics indicate the correspondence between the syntactic forms of RDF and the entities in the universum of discourse. The semantics are expressed by a model theory. A *valid triple* is a triple whose elements are defined by the model theory. A *valid RDF graph* is a set of valid RDF triples.

2.7.2. The model theory of RDF

The vocabulary *V* of the model is composed of a set of URI's.

LV is the set of *literal values* and *XL* is the mapping from the literals to *LV*.

A *simple interpretation I* of a vocabulary *V* is defined by:

1. A non-empty set *IR* of resources, called the domain or universe of *I*.

2. A mapping *IEXT* from *IR* into the powerset of *IR X (IR union LV)* i.e. the set of sets of pairs <*x,y*> with *x* in *IR* and *y* in *IR* or *LV*. This mapping defines the properties of the RDF triples.

3. A mapping *IS* from *V* into *IR*

IEXT(x) is a set of pairs which identify the arguments for which the property is true, i.e. a binary relational extension, called the *extension* of x. [RDFMS]

Informally this means that every URI represents a resource which might be a page on the internet but not necessarily: it might as well be a physical object. A property is a relation; this relation is defined by an extension mapping from the property into a set. This set contains pairs where the first element of a pair represents the subject of a triple and the second element of a pair represent the object of a triple. With this system of extension mapping the property can be part of its own extension without causing paradoxes. This is explained in [RDFMS].

2.7.3. Examples

Take the triple:

:bird :color :yellow.

In the set of URI's there will be things like: :bird, :mammal, :color, :weight, :yellow, :blue etc...These are part of the vocabulary *V*.

In the set IR of resources will be: #bird, #color etc.. i.e. resources on the internet or elsewhere. #bird might represent e.g. the set of all birds.

There then is a mapping *IEXT* from #color (resources are abbreviated) to the set {(#bird,#blue),(#bird,#yellow),(#sun,#yellow),...}

and the mapping *IS* from *V* to *IR*:

:bird → #bird, :color → #color, ...

The URI refers to a page on the internet where the domain *IR* is defined (and thus the semantic interpretation of the URI).

2.8.RDFProlog

I have defined a prolog-like structure by the Haskell module RDFProlog.hs (for a sample see chapter 9. Applications) where e.g.
name(company,"Test") is translated to the triple: *(company,name,"Test")*.
This language is very similar to the language DATALOG that is used for a deductive access to relational databases [ALVES].
The alphabet of DATALOG is composed of three sets: VAR,CONST and PRED
denoting respectively variables, constants and predicates. One notes the absence of functions. Also predicates are not nested.
In RDFProlog the general format of a rule is:

predicate-1(subject-1,object-1),..., predicate-n(subject-n,object-n) :>
predicate-q(subject-q,object-q), ...,predicate-z(subject-z,object-z).

where all predicates,subjects and objects can be variables.
The format of a fact is:

predicate-1(subject-1,object-1),..., predicate-n(subject-n,object-n).

where all predicates,subjects and objects can be variables.
Variables begin with capital letters.
All rules and facts are to be entered on one line. A line with a syntax error is neglected. This can be used to add comment.
The parser is taken from [JONES] with minor modifications.

2.9. A global view of the Semantic Web

2.9.1. Introduction.

The Semantic Web will be composed of a series of standards. These standards have to be organised into a certain structure that is an expression of their interrelationships. A suitable structure is a hierarchical, layered structure.

2.9.2. The layers of the Semantic Web

Fig.2.2. illustrates the different parts of the Semantic Web in the vision of Tim Berners-Lee. The notions are explained in an elementary manner here. Later some of them will be treated more in depth.

Layer 1. Unicode and URI

At the bottom there is Unicode and URI. Unicode is the Universal code.

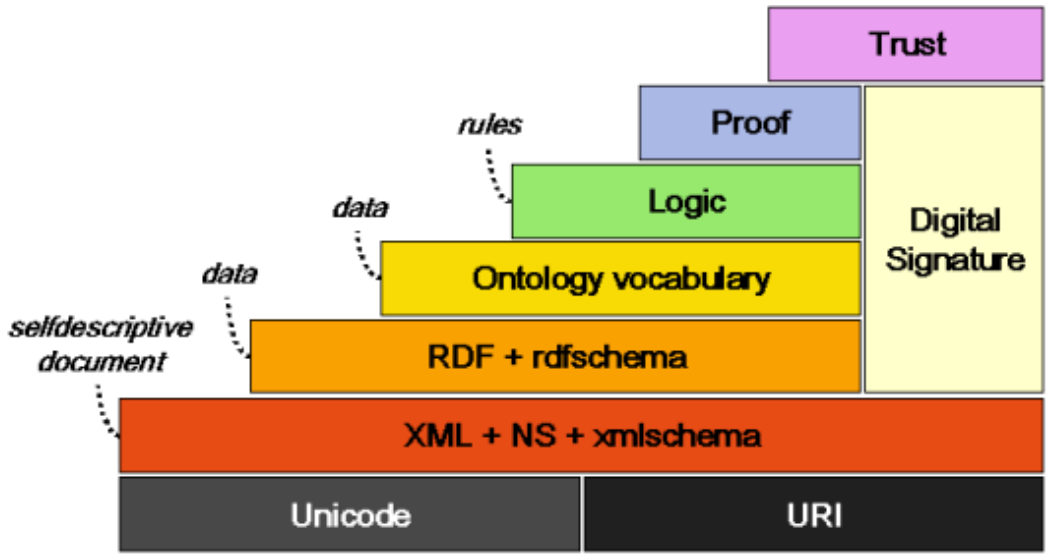

Fig.2.2. The layers of the Semantic Web [Berners-Lee]

Unicode codes the characters of all the major languages in use today. [UNICODE].

Layer 2. XML, namespaces and XML Schema

See 2.2. for a description of XML.

Layer 3. RDF en RDF Schema

The first two layers consist of basic internet technologies. With layer 3 the Semantic Web begins.
RDF Schema (rdfs) has as a purpose the introduction of some basic ontological notions. An example is the definition of the notion “Class” and “subClassOf”.

Layer 4. The ontology layer

The definitions of rdfs are not sufficient. A more extensive ontological vocabulary is needed. This is the task of the Web Ontology workgroup of the W3C who has defined already OWL (Ontology Web Language) and OWL Lite (a subset of OWL).

Layer 5. The logic layer

In the case study the use of rules was mentioned. For expressing rules a logic layer is needed. An experimental logic layer exists [SWAP/CWM]. This layer is treated in depth in the following chapters.

Layer 6. The proof layer

In the vision of Tim Berners-Lee the production of proofs is not part of the Semantic Web. The reason is that the production of proofs is still a very active area of research and it is by no means possible to make a standardisation of this. A Semantic Web engine should only need to verify proofs. Someone sends to site A a proof that he is authorised to use the site. Then site A must be able to verify that proof. This is done by a suitable inference engine. Three inference engines that use the rules that can be defined with this layer are: CWM [SWAP/CWM] , Euler [DEROO] and RDFEngine developed as part of this thesis.

Layer 7. The trust layer

Without trust the Semantic Web is unthinkable. If company B sends information to company A but there is no way that A can be sure that this information really comes from B or that B can be trusted then there remains nothing else to do but throw away that information. The same is valid for exchange between citizens. The trust has to be provided by a web of trust that is based on cryptographic principles. The cryptography is necessary so that everybody can be sure that his communication partners are who they claim to be and what they send really originates from them. This explains the column "Digital Signature" in fig. 2.2.
The trust policy is laid down in a "facts and rules" database. This database is used by an inference engine like RDFEngine. A user defines his policy using a GUI that produces a policy database. A policy rule might be e.g. *if* the virus checker says 'OK' *and* the format is .exe *and* it is signed by 'TrustWorthy' *then* accept this input.
The impression might be created by fig. 2.2 that this whole layered building has as purpose to implement trust on the internet. Indeed it is necessary for implementing trust but, once the pyramid of fig. 2.2 comes into existence, on top of it all kind of applications can be built.

Layer 8. The applications

This layer is not in the figure; it is the application layer that makes use of the technologies of the underlying 7 layers. An example might be two companies A and B exchanging information where A is placing an order with B.

3. Related work

3.1. Automated reasoning

3.1.1. Introduction

After an overview of systems of *automated reasoning*, the strategy followed in this thesis is briefly explained.
In connection with automated reasoning the terms *machinal reasoning* and, in a more restricted sense, *theorem provers*, are also relevant.
There are proof checkers i.e. programs that control the validity of a proof and proof generators.In the semantic web an inference engine will not necessarily serve to generate proofs but to check proofs; those proofs must be in a format that can be easily transported over the net.

3.1.2. General remarks

Automated reasoning is an important domain of computer science. The number of applications is constantly growing.

- proof of theorems in mathematics
- reasoning of intelligent agents
- natural language understanding
- mechanical verification of programs
- hardware verifications (also chips design)
- planning
- a proof checker controls a proof made by another system
- and ... whatever problem that can be logically specified (a lot!)

Generally, in automated reasoning there is a *database of expressions and a logic* system. Whenever a *lemma* has to be proved, the logic system tries to use the expressions in order to deduce the lemma.
In the Semantic Web the lemma is called a *query*. An *answer* is either the confirmation of the query if the query does not contain variables; or it is the query with the variables substituted with terms.
Hilbert-style calculi have been traditionally used to characterize logic systems. These calculi usually consist of a few axiom schemata and a small number of rules that typically include modus ponens and the rule of substitution. [STANFORD]

3.1.3. Reasoning using resolution techniques

3.1.3.1. Substitutions

When a substitution *(v, t)* where *v* is a variable and *t* is a term is applied to an expression, all occurrences in the expression of the variable *v* will be replaced by the term *t*.
Example: the substitution *(x," son")* applied to *("John", x, "Paul")* will give *("John", "son", "Paul")*.

The application of a substitution *s* to a term *t* is written *st*.
When s_1 and s_2 are substitutions and *t* is a term, then $s_1 s_2 t$ is a subtitution too.

3.1.3.2. Unification

The substitution *s* unifies expressions e_1 and e_2 if $s\ e_1 = s\ e_2$. It is possible that two expressions are unified by more than one substitution. The *most general unifier* of two expressions is the substitution *s* such that, if s_1 is also a unifier of these expressions, $s_1 = ss_x$.

3.1.3.3. Resolution reasoning

3.1.3.3.1. Introduction

In resolution reasoning the logic system tries to proof the negation of the query. If this proof has as a result the value *false* then the query must be *true*. This is the *refutation* of the negation of the query. For a certain logic system it must be proven that the system is *refutation complete* i.e. that it is always possible to proof that an expression is false.

3.1.3.3.2. The resolution principle

Resolution reasoning uses the resolution principle.

First two simple examples:

1) $a \rightarrow b$ can be written $\neg a \vee b$.
Given $a \rightarrow b$, *a* and the query *b*. The query is negated to $\neg b$, what must be proved to be false.
From $\neg a \vee b$ and $\neg b$ follows $\neg a$. But from $\neg a$ and *a* follows *false* what had to be proved.

2) From $\neg Af(x) \vee \neg Rh(x)y \vee Ay$ and $Rzf(u) \vee Bz$ follows
$\neg Af(x) \vee Af(u) \vee Bh(x)$ with unifying substitution
$s = [(z, h(x)),(y, f(u))]$.
After subtitution the factors $\neg Rh(x)f(u)$ and $Rh(x)f(u)$ cancel each other.

Resolution principle

If A and B are two expressions and s is a substitution unifying two factors $L1$ and $L2$ and $L1$ occurs in A and $\neg L2$ occurs in B, then the resolution principle gives $s((A \setminus L1) \vee (B \setminus L2))$ where the anti-slash stands for "without". [VANBENTHEM]

3.1.3.3.2. The resolution strategy

The mechanism by which the refutation of the negated query is obtained consists of repeatedly applying the resolution principle [STANFORD]. The resolution principle has to be applied to two expressions. The choice of these expressions is part of the *resolution strategy.* Another example of mechanisms that are part of the resolution strategy are the removal of redundant expressions or tautologies. *Subsumption* is the replacement of specific expressions by more general ones.
Whatever the resolution strategy, its refutation completeness must be proved.

A well known resolution strategy is *SLD-resolution*: Selection, Linear, Definite [CS]. SLD-resolution is used by *Prolog.* To explain SLD-resolution first some new notions have to be introduced.
A *term* is:
a) an individual variable or a constant
b) if f indicates a function and $t1, \ldots, tn$ are terms, then $f(t1, \ldots, tk)$ is a term
c) nothing else is a term
[VANBENTHEM].
An *atom* is a formula of the form $P(t1, t2, \ldots tn)$ where $t1, .., tn$ are terms. A *literal* is an atom (*positive literal)* or the negation of an atom (*negative literal)*. With literals *clauses* are formed. A clause is a disjunction of literals: $L1 \vee L2 \ldots \vee Lm$ where $L1, \ldots, Lm$ are literals.

Example of a clause: $P(a,b) \vee Z(D(b,c),G) \vee X$.

A clause is universally quantified i.e. all variables are universal variables.
A *Horn clause* has the form:

$\neg A1 \vee \neg A2 \vee \ldots \neg An \vee A$ where Ax are positive literals and all variables are universally quantified.
[VANBENTHEM]

Now back to SLD.
In SLD is a *selection function* for clauses; a quit simple one given the fact that the first in the list is selected. This is one of the points where optimization is possible namely by using another selection function.
Linear is explained here below.
Prolog works with Horn clauses.
In prolog the definition of *a definite database* is a database that has exactly one positive literal in each clause and the unification is done with this literal.
Following resolution strategies are respected by an SLD-engine [CS]:

- *depth first*: each alternative is investigated until a unification failure occurs or until a solution is found. The alternative to depth first is breadth first.
- *set of support*: at least one parent clause must be from the negation of the query or one of the "descendents" of a query clause. This is a complete procedure that gives a goal directed character to the search.
- *unit resolution*: at least one parent clause must be a "unit clause" i.e. a clause containing a single literal.
- *input resolution*: at least one parent comes from the set of original clauses (from the axioms and the negation of the goals). This is not complete in general but complete for Horn clause databases.
- *linear resolution*: one of the parents is selected with set of support strategy and the other parent is selected by the input strategy.
- *ordered resolution*: this is the way prolog operates; the clauses are treated from the first to the last and each single clause is unified from left to right.

3.1.3.4. Comparison with RDFEngine

The mechanism used in RDFEngine is resolution based reasoning with a SLD-resolution strategy. The proof of the refutation completeness (chapter 5) is given by reasoning on a graph instead of using first order logic.

3.1.4. Backwards and forwards reasoning

Suppose a database consisting of expressions $A,B,C \ldots$ that do not contain variables and rules $(X1,\ldots,Xn) \rightarrow Y$ where $X1, \ldots Xn, Y$ are expressions that

can contain variables. *X1, ...,Xn* are called the *antecedents, Y* is the *consequent*. Let Q be a query with or without variables.
In *forwards reasoning* a check is done whether the query *Q* can be unified with a fact. If this is the case then a solution is found. Then the rules are used to generate new facts *Yi, ...Yn* by those rules whose antecedents can be unified with facts. After all rules have been used, again a check is done for unifying the query with a fact. Again solutions can be found. Then the rules are used again etc… This process goes on till no more new facts can be deduced by the rules. All solutions will then have been found.

In *backwards reasoning* a check is also done whether *Q* can be unified with a fact. After that however the system tries to unify *Q* with the consequent of a rule. Suppose *Q* unifies with the consequent *Y* of the rule *(X1,...,Xn)* → *Y* with the susbtitution *s*. Then *sY* will be true if *sX1, ...,sXn* are true. Important here is that the rule is interpreted *backwards*. Resolution or backwards reasoning applied to RDF will be explained in detail in chapter 5.

3.1.5. Other mechanisms

There are many other mechanisms for automated reasoning. I will only list some of them with references.

- Sequent deduction [STANFORD] [VAN BENTHEM].
- Natural deduction [STANFORD] [VAN BENTHEM].
- The matrix connection method [STANFORD].
- Term rewriting [DICK].
- Mathematical induction [STANFORD] [WALSH].
- Higher order logic [STANFORD].
- Non-classical logics [STANFORD].
- Lambda calculus [GUPTA] [HARRISON].
- Proof planning [BUNDY].

3.1.6. Theorem provers

Three different kinds of provers can be discerned [DONALD]:

- those that want to mimic the human thought processes
- those that do not care about human thought, but try to make optimum use of the machine
- those that require human interaction.

There are domain specific and general theorem provers. Provers might be specialised in one mechanism e.g. resolution or they might use a variety of mechanisms.

In general theorem provers are used to solve difficult or complex problems in the realm of exact sciences and often those are problems that are difficult to handle manually.

[NOGIN] gives the following ordering:

- Higher-order interactive provers:
    - Constructive: ALF, Alfa, Coq, (MetaPRL, NuPRL)
    - Classical: HOL, PVS
- Logical Frameworks: Isabelle, LF, Twelf, (MetaPRL)
- Inductive provers: ACL2, Inka
- Automated:
    - Multi-logic: Gandalf, TPS
    - First-order classical logic: Otter, Setheo, SPASS
    - Equational reasoning: EQP, Maude
- Other: Omega, Mizar

3.1.7. Conclusion

There are a lot of possibilities for making a choice of a basic mechanism for inferencing for the Semantic Web. There is a substantial difference between the goals of a theorem prover and a basic engine for the Semantic Web. Such an engine should have following characteristics:

- usable for a broad spectrum of applications
- able to handle large data sets what implies a simple structure
- usable for subquerying what implies an inference process that can be interrupted

In first instance RDF with the addition of an implication could be sufficient. As inference mechanism an SLD-engine seems most appropriate for following reasons:

- it has already been used intensively for a broad spectrum of applications and specifically by Prolog programs
- it has been used for access to databases [ALVES].
- the refutation process and the corresponding goal directed reasoning (starting with the query) is most suitable for an interrupted inference process

3.2. Logic

3.2.1. First order logic

An obvious way to define inferencing on top of RDF is to make a description of RDF in first order logic. Then rules are defined by an implication (see e.g. [DECKER]). Reasoning is done using first order logic. In RDFEngine a constructive approach is used instead of an approach by first order logic.

3.2.2. Intuistionistic or constructive logic

In intuistionistic logic the meaning of a statement resides not in its truth conditions but in the means of proof or verification. In classical logic *p v ~p* is always true ; in constructive logic *p or ~p* has to be 'constructed'. If *forSome x.F* then effectively it must be possible to compute a value for *x*.
The BHK-interpretation of constructive logic [STANFORD] (Brouwer, Heyting, Kolmogorov):

a) A proof of *A* and *B* is given by presenting a proof of *A* and a proof of *B*.
b) A proof of *A* or *B* is given by presenting either a proof of *A* or a proof of *B*.
c) A proof of *A* ➔ *B* is a procedure which permits us to transform a proof of *A* into a proof of *B*.
d) The constant false has no proof.

A proof of *~A* is a procedure that transforms a hypothetical proof of *A* into a proof of a contradiction.
For two reasons *verifiability* is important for the Semantic Web.
a) the processes are fully automated (no human intervention) so the verification has to be done by computers.
b) different parts of the inferencing proces (inference steps) are executed by different engines on different sites.
RDFEngine has a constructive view on RDF in that every triple in an inference process is constructed and is thus a verifiable triple (2.4.2). I argue that this constructive view is necessary in automated processes.

3.2.3. Paraconsistent logic

3.2.3.1. Elaboration

The development of *paraconsistent logic* was initiated in order to challenge the logical principle that anything follows from contradictory premises, *ex contradictione quodlibet (ECQ)* [STANFORD]. Let be a relation of logical consequence, defined either semantically or proof-theoretically. Let us say that is *explosive* iff for every formula *A* and *B*,

$\{A , \sim A\}$ B. Classical logic, intuitionistic logic, and most other standard logics are explosive. A logic is said to be *paraconsistent* iff its relation of logical consequence is not explosive.
Also in most paraconsistent systems the disjunctive syllogism does not hold:
From $A, \sim A \vee B$ we cannot conclude B. Some systems:
Non-adjunctive systems: from A , B the inference $A \& B$ fails.
Non-truth functional logic: the value of A is independent from the value of $\sim A$ (both can be one e.g.).
Many-valued systems: more than two truth values are possible.If one takes the truth values to be the real numbers between 0 and 1, with a suitable set of designated values, the logic will be a natural paraconsistent fuzzy logic.

3.2.3.2. Relevance for the Semantic Web

It is to be expected that contradictions will not be exceptional in the Semantic Web. It would not be acceptable under such circumstances that anything can be concluded from a contradiction. Therefore the logic of the Semantic Web must not be explosive.

3.2.4. Conclusion

Constructive logic and paraconsistent logic are important for the Semantic Web. Many other logics have the potential to be used in the Semantic Web. Some candidates are: higher order logics, modal logic, linear logic.

3.3. Existing software systems

3.3.1. Introduction

I make a difference between a *query engine* that just does querying on a RDF graph but does not handle rules and an *inference engine* that also handles rules.
In the literature this difference is not always so clear.
The complexity of an inference engine is a lot higher than a query engine. This is because rules permit to make sequential deductions. In the execution of a query this deductions are to be constructed. This is not necessary in the case of a query engine.
I will not discuss query engines. Some examples are: DQL [DQL], RQL [RQL], XQUERY [BOTHNER].
Rules also suppose a logic base that is inherently more complex than the logic in the situation without rules. For an RDF query engine only the simple principles of entailment on graphs are necessary (see chapter 5).

RuleML is an important effort to define rules that are usable for the World Wide Web [GROSOP].
The Inference Web [MCGUINESS2003] is a recent realisation that defines a system for handling different inferencing engines on the Semantic Web.

3.3.2. Inference engines

3.3.2.1.Euler

Euler is the program made by Deroo [DEROO]. It does inferencing and also implements a great deal of OWL (Ontology Web Language).
The program reads one or more triple databases that are merged together and it also reads a query file. The merged databases are transformed into a linked structure (Java objects that point to other Java objects). The philosopy of Euler is graph oriented.
This structure is different from the structure of RDFEngine. In RDFEngine the graph is not described by a linked structure but by a collection of triples, a tripleset.
The internal mechanism of Euler is in accordance with the principles exposed in chapter 3.
Deroo also made a collection of test cases upon which I based myself for testing RDFEngine.

3.3.2.2. CWM

CWM is the program of Berners-Lee.
It is based on forward reasoning.
CWM makes a difference between existential and universal variables[BERNERS]. RDFEngine does not make that difference. It follows the syntaxis of Notation 3: *log:forSome* for existential variables, *log:forAll* for universal variables. As will be explained in chapter 5 all variables in RDFEngine are quantified in the sense: *forAllThoseKnown.*
Blank (anonymous) nodes are instantiated with a unique URI.
CWM makes a difference between *?a* for a universal, local variable and *_:a* for an existential, global variable. RDFEngine maintains the difference between local and global but not the quantification.

3.3.2.3. TRIPLE and RuleML

3.3.2.3.1. Introduction

TRIPLE and RuleML are two examples of inference engines that have chosen to use an approach based on classic logic i.e. a logic that is first order logic or a subset.

3.3.2.3.2. Triple

TRIPLE is based on Horn logic and borrows many features from F-Logic [SINTEK].
TRIPLE is the successor of SiLRI.

3.3.2.3.3. RuleML

3.3.2.3.3.1. Introduction

RuleML is an effort to define a specification of rules for use in the World Wide Web.

3.3.2.3.3.2. Technical approach

The kernel of RuleML are *datalog* logic programs (see chapter 2) [GROSOF]. It is a declarative logic programming language with model-theoretic semantics. The logic is based on Horn logic.
It is a *webized* language: namespaces or defined as well as URI's.
Inference engines are available.
There is a cooperation between the RuleML Iniative and the Java Rule Engines Effort.

3.3.3. The Inference Web

3.3.3.1. Introduction

The Inference Web was introduced by a series of recent articles [MCGUINESS2003].
When the Semantic Web develops it is to be expected that a variety of inference engines will be used on the web. A software system is needed to ensure the compatibility between these engines.
The inference web is a software system consisting of:
a web based registry containing details on information sources and reasoners called the Inference Web Registry.

1) an interface for entering information in the registry called the Inference Web Registrar.
2) a portable proof specification
3) an explanation browser.

3.3.3.2. Details

In the Inference Web Registry data about inference engines are stored. These data contain details about *authoritative sources, ontologies, inference engines and inference rules.* In the explanation of a proof every inference step should have a link to at least one inference engine.

1) The Web Registrar is an interface for entering information into the Inference Web Registry.
2) The portable proof specification is written in the language DAML+OIL. In the future it will be possible to use OWL. There are four major components of a portable proof:
    a) inference rules
    b) inference steps
    c) well formed formulae
    d) referenced ontologies

These are the components of the inference process and thus produces the proof of the conlusion reached by the inferencing.

3) The explanation browser show a proof and permits to focus on details, ask additional information, etc…

3.3.3.3. Conclusion

I believe the Inference Web is a major and important step in the development of the Semantic Web. It has the potential of allowing a cooperation between different inference engines. It can play an important part in establishing trust by giving explanations about results obtained with inferencing.

3.3.4. Swish

Swish is a software package that is intended to provide a framework for building programs in Haskell that perform inference on RDF data [KLYNE]. Where RDFEngine uses resolution backtracking for comparing graphs, Swish uses a graph matching algorithm described in [CARROLL]. This algorithm is interesting because it represents an alternative general way, besides forwards and backwards reasoning, for building a RDF inference engine.