Methods Survey

Marc Ehrig, Steffen Staab, and Christoph Tempich
(University of Karlsruhe)

with contributions from:

Marta Sabou and Heiner Stuckenschmidt (Vrije Universiteit Amsterdam)

Executive Summary.
SWAP EU IST-2001-34103 Project Deliverable D3.1
Many methods exist in the field of P2P and in the field of ontology processing. Before starting to design new methods and to integrate the existent methods a careful investigation of the current state-of-the-art is necessary. The survey is provided in this deliverable.

Document Id.	SWAP/2002/D3.1/v1.4
Project	SWAP EU IST-2001-34103
Date	September 30, 2002
Distribution	public

SWAP Consortium

This document is part of a research project partially funded by the IST Programme of the Commission of the European Communities as project number IST-2001-34103. The partners in this project are: Institute AIFB / University of Karlsruhe (coordinator, Germany), Vrije Universiteit Amsterdam VUA (Netherlands), Meta4 (Spain), empolis UK ltd. (UK), empolis Polska (Poland), Dresdner Bank AG (Germany), and IBIT (Spain).

University of Karlsruhe
Institute AIFB
Englerstr. 28
D-76128 Karlsruhe
Germany
Tel: +49 721 608 3923, Fax: +49 721 608 6580
Contactperson: Steffen Staab
E-mail: staab@aifb.uni-karlsruhe.de

Vrije Universiteit Amsterdam (VUA)
Division of Mathematics and Informatics W&I
De Boelelaan 1081a
1081 HV Amsterdam
The Netherlands
Tel: +31 20 4447786, Fax: +31 20 4447653
Contactperson: Hans Akkermans
E-mail: hansakkermans@cs.vu.nl

Meta4 Spain S.A.
Rozabella, 8 Centro Europa Empresarial
28230 Las Rozas (Madrid)
Spain
Tel: +34 91 634 85 00, Fax: +34 91 634 86 86
Contactperson: Adelma Stolbun
E-mail: adelmas@meta4.com

empolis UK ltd.
Unit B, The Dorcan Complex, Faraday Road
SN3 5HQ Swindon
United Kingdom
Tel: +44 77 99694621, Fax: +44 17 93485451
Contactperson: Graham Moore
E-mail: gdm@empolis.co.uk

empolis Polska Sp. z o. o.
Plocka 5a
01-231 Warsaw
Poland
Tel: +48 22 535 88 06, Fax: +48 22 535 88 14
Contactperson: Mariusz Olko
E-mail: mariusz.olko@empolis.pl

Dresdner Bank AG
Jürgen-Ponto-Platz 1
60301 Frankfurt am Main
Germany
Tel: +49 69 263 58265, Fax: +49 69 263 81385
Contactperson: Martin Lorenz
E-mail: martin.lorenz@dresdner-bank.com

Fundación IBIT
Reverend Francesc Sitjar 1
07010 Palma de Mallorca
Spain
Tel: +34 971 177 271, Fax: +34 971 177 279
Contactperson: Esteve Lladó Marti
E-mail: esteve@ibit.org

Contents

Chapter 1

Introduction

1.1 Scope

Before the project partners can start to design and implement the SWAP-system they have to gain a broad basic understanding of the current state in research. The intention of this document is to give a thorough investigation of this. Existing methods and tools are presented. This overview is by no means complete. Additionally the focus lies on listing rather than giving a very detailed description on each method. The interested reader is referred to the quoted references for additional information.

Only in the next step the methods and tools can be utilized and integrated into the SWAP-project. In this next step, which is beyond the scope of this document, actual decisions on design and implementation have to be taken. This document gives ideas for that, but the designers are not restricted in any kind to the proposed methods.

1.2 Document Structure

As a first step towards checking the current state of research, the respective communities doing work related to our project are identified. This is done in chapter 2.

Besides thinking about the general process model and elaborating the room of potential design decisions, one needs the adaptation and invention of a diversity of methods. They have to support the P2P paradigm for decentralized knowledge sharing, search, and structuring. This is done along the following lines:

- P2P for O (Peer-to-Peer for Ontologies): The next level of ontology technology is going to let semantics emerge from the way in which knowledge is accessed and used - across several peers. The peer-to-peer knowledge management system will dynamically come up with ontologies (chapter 3). A part of this topic will also be updating existing ontological structures (chapter 4).

- P2P for KM (Peer-to-Peer for Knowledge Management): Knowledge is now updated and searched in a decentralized manner. For this purpose a selection service around peers and content is necessary (chapter 5).
- O for P2P & KM (Ontologies for Peer-to-Peer and Knowledge Management): With peer-to-peer-based solutions, existing ontology-based knowledge management approaches do not stay the same. Ontology inferencing and alignment (chapter 6) must be adapted and new visualization techniques (chapter 7) must be investigated in order to deal with the distinction between local and shared ontologies.

Chapter 2

Involved Communities

The SWAP project covers various topics. Due to this diversity methods from a wide range of research communities have to be regarded and integrated.

Five main research communities were identified:

- Agents
- Computer Networks
- Databases
- Graph Theory
- Knowledge Management

They will be presented in more detail.

Agents: Agents are commonly regarded as independently acting individuals. These agents can fulfill a variety of tasks, from distributed computing to artificial intelligence. Analogously peers should be self-organizing and as far as possible act independently from any central instances. Distributed agents face similar difficulties as peers in our distributed knowledge network will be facing. [SWKL02]

Computer Networks: From a technical perspective the Internet has some similarity to a peer-to-peer network. Questions of how to log on to the network and how to forward requests to a particular computer had to be investigated. Efficiency has been optimized for years. Although the peer-to-peer network pursued in our context does not work on a level of physical connections, but on a much higher level, some of the protocols or concepts from the area of network computing can most probably be adapted and transferred.

Databases: The idea of distributed databases is strongly related to our idea of distributed repositories in the peer-to-peer network. Many concepts on how to efficiently organize the distributed databases have been pursued and tested for years. This also includes the particular interesting topic of retrieval of relevant information across wide areas. Consistency of updates and databases has been approached from different directions.

Graph Theory: For us graph theory is of interest for two reasons. On the one hand it covers the idea of a peer network from a more theoretical point of view. On the other hand it is the basis for all considerations around ontology graphs. The knowledge around the SWAP project is basically saved in a big ontology graph which itself is distributed on a network (graph) of peers. Building, organizing, and traversing this graph has to be done respecting premises from graph theory. Aspects of matching and mapping also play a major role in this context. Many of the difficulties have been investigated in Operations Research.

Knowledge Management: The representation models from knowledge management will be the foundation when setting up the knowledge organization on the peer-to-peer network. The knowledge structures with their formalisms can be easily reused in SWAP. How to create representations of knowledge automatically and semi-automatically is one topic where a lot of research has been pursued. Mapping and comparison of different representations as they can be found on the independent peers have been investigated by this community as well.

Chapter 3

Ontology Creation

Knowledge exists in various formats and structures. Before it can be shared it has to be formalized in such a manner that others can access and understand it. This also has to hold true for machines automatically processing it. In the past years the notion of using ontologies for this purpose has become popular. How to create the ontologies for the SWAP project is investigated in this chapter. This will also include the field of emerging semantics, where ontologies can be scraped from existing legacy information.

3.1 Ontologies

Before focusing on the goal of building ontologies we would like to present a short introduction of what ontologies are and how they can be represented.

3.1.1 Definition of Ontologies

The most prominent definition of ontology is: *"An ontology is an explicit specification of a conceptualization"*[Gru93]. A 'conceptualization' refers to an abstract model of some phenomenon in the world by identifying the relevant concept of that phenomenon. 'Explicit' means that the types of concepts used and the constraints on their use are explicitly defined. This definition is often extended by three additional conditions: "An ontology is an explicit, *formal* specification of a *shared* conceptualization *of a domain of interest*". 'Formal' refers to the fact that the ontology should be machine readable (which excludes for instance natural language). 'Shared' reflects the notion that an ontology captures consensual knowledge, that is, it is not private to some individual, but accepted as a group. The reference to 'a domain of interest' indicates that for domain ontologies one is not interested in modelling the whole world, but rather in modelling just the parts which are relevant to the task at hand. [Stu01]

3.1.2 Ontology structure

We define a more practical view: An ontology structure is a tuple consisting of classes, properties, and a hierarchy. Classes can be seen as general object definitions. They describe a whole class of objects, not one specific object. An example of an object class i.e. a concept is *Airline*. The specific airline *Lufthansa* is not. A property is a non-taxonomic relation between classes. An example for such a relation would be *operate* in the sense of: *Lufthansa operates* a fleet of *Boeing 747s*. Properties can be enhanced by special rules covering e.g. reflexivity, symmetry, and transitivity. First Order Logic is an important tool for this purpose. Ultimately, specific instances of a class are created. In contrast to concepts instances represent a specific object. Staying with the above example: *Lufthansa* is an instance of the concept *Airline*. Figure 3.1 shows an example of ontology and metadata structures in a graph.

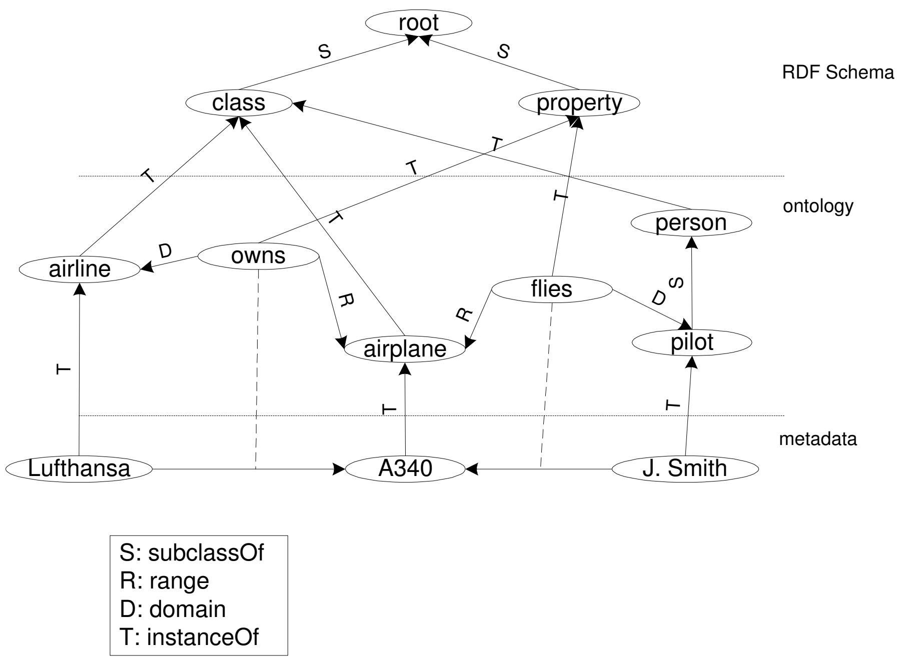


Figure 3.1: Ontology and Metadata

3.1.3 Ontology representation

There has been a lot of effort within the research community to come up with an internationally accepted standard for ontology representation [FHvH$^+$01]. However, until now, no Gold Standard has been approved, but the standardization effort is very advanced. As we focus on Semantic Web technologies, only OWL is described with more detail. Others are F-Logic [KLW95] or CyCL [LG90].

OWL: The OWL Web Ontology Language is being designed by the W3C Web Ontology Working Group in order to provide a language that can be used for applications that need to understand the content of information instead of just understanding the human-readable presentation of content. OWL facilitates greater machine readability of web content than XML, RDF, and RDF-S support by providing a additional vocabulary for term descriptions [MDS02]. It builds on prior efforts like currently widespread DAML+OIL format[Hor02].

3.2 General

To get a better understanding of the ontology creation process some related areas are presented in this section.

3.2.1 Knowledge Acquisition

Knowledge acquisition is a complex process which traditionally is extremely expensive. The typical process can be split into three steps:

- Knowledge Elicitation: Initially knowledge is elicited either from a set of documents or from an expert of the domain.
- Knowledge Base Definition: With this input the structure of an ontology is defined.
- Knowledge Base Population: Finally a set of instances are associated to the concepts in the ontology.

Up to now this construction cycle is normally done manually. But efforts are taken to automate parts or the whole of the cycle. [BCW01]

3.2.2 Learning

In the context of the SWAP project a lot of the ontology creation process has to be done on the fly and therefore automatically without human intervention.

Machine Learning

The classical definition of machine learning is as follows:
"The system learns from some experience E according to some performance measure P if it improves its performance (as measured by P) after passing the experience E."[Mit97]
There are several dimensions of machine learning which will now be briefly outlined.

Basics: off-line vs. on-line learning: In off-line learning the system learns from before actually giving an output decision. In contrast on-line learning gets one piece of information at a time, learns from this, and produces decision on the fly.
supervised vs. unsupervised learning: The other dimension is the grade of interaction by the human. Supervised learning requires a set of training data with existing output data. Unsupervised learning produces clusters on its own. The clusters are built upon a heuristics but the system has no notion of what the actual meaning of the clusters is. [Ome00]

Machine Learning Techniques: This section will give a brief list of techniques which can be used for ontology learning. [Ome01a]

- Propositional Rule Learning , learn association rules, or other attribute-value rules.
- Bayesian Learning , is mostly represented by Naive Bayes classifier. It generates probabilistic attribute-value rules.
- First-order Logic Rules Learning , searches for rules that cover all available training instances.
- Clustering Algorithms , group instances based on similarity measures.

Ontology Learning

Ontology learning goes along the knowledge acquisition approach, from knowledge elicitation, ontology schema extraction (base definition), and ontology instance extraction (base population).

A general problem of exclusive machine learning is the fact that ontologies created this way will lose the characteristic of being human-readable and -understandable. [Ome01a]

3.3 Existing Creation Approaches

Ontologies are no new topic and therefore some approaches for ontology creation are already available. Many have been the topic of extensive research for years.

3.3.1 Human Engineer

A time-consuming approach is to have human engineers provide the ontology. Still, this approach is the currently most used approach, because a cooperation between knowledge engineers and ontology engineers allows very consistent and intuitive ontologies.

In [SEA$^+$02] a road-map for the ontology creation process is provided. It ranges from a feasibility study, the ontology kick-off, refinement routines, and evaluation, to a regular maintenance and evolution module.

3.3.2 Natural Language Processing

The semi-automatic approach Text-To-Onto is presented by [MNS01]. The authors rely on the SMES-system[NBB$^+$97], a natural language processing system for German. By extracting syntactical information from texts important concepts and relations between them can be found. This can be used as input for building an ontology.

3.3.3 Language Games

A very interesting approach is given by [Ste02]. He created two independent agents with knowledge only about themselves. In an evolutionary way the agents try to communicate to one another in an arbitrary alphabet. This way both finally learn a language.

3.4 Emergent Semantics

Each user creates and maintains various knowledge structures locally. One possibility of ontology creation is to copy these structures and adapt them to our use. Adapting especially means to combine the input from several structures or even different peers. This topic is also known as "emergent semantics"[Mae02]. Matching and merging routines are also required in this context. There are several structures one can use for this task, as presented in the next paragraphs.

3.4.1 Bookmarks

Bookmarks are a very prominent example of structured knowledge. When browsing on the Web a user comes across interesting pages. Once in a while he finds the page worth to remember and saves it in his bookmarks. As big numbers of links are too complex to manage, a hierarchy is normally built. This hierarchy already fits to a small ontology. The idea of sharing bookmarks around a network is not new. PowerBookmarks is a system which saves various bookmarks on a server. Every participant can then set up queries to find topics which correspond his need. [LVC$^+$99]

3.4.2 File-Folders

Another possible structure to build upon are the folders on each personal computer system. Files with specific titles are grouped within a taxonomy explicitly created to achieve tidiness of the working space. Besides file-title and the folder-structure one could also include metadata from the file. Existing file sharing approaches allow access to the files but the folder structures are not evaluated any further.

3.4.3 Existing Local Ontologies

A very straight forward approach is the use of already existent ontologies. If the user already utilizes an ontology this normally explicitly modelled structure should be integrated into the whole picture. The quality should always be better than with the other alternatives.

3.4.4 Emails

Emails have a similar structure as file-folders. Additionally each email has a short description of its content - the subject field. Again a taxonomy can be extracted. One drawback of emails are the difficulties of accessing the different email programs and possibly locked emails.

3.4.5 Texts

Ontology creation from text has been around for some time. As texts occur everywhere the main goal is to identify potentially useful texts out of the big number. A lot of work has already been done in extracting data or even ontologies from plain text. As an example we can mention [MNS01].

3.4.6 Databases

Most of today's business applications use databases. It will be of great value if one can integrate part of them. Especially the structures of the databases could be used in the sense of emergent semantics.

The following last two methods use a different view of emergent semantics. Whereas the previous methods were all based on existing structures created by a user, they focus on the behavior of the user.

3.4.7 User Actions

A very complex way of building ontologies and especially setting up relations in between concepts would be to track user actions. An example would be to realize if a user frequently opens two specific files in parallel e.g. the letter template and an address list. Naturally this hasn't have to be limited by one program but can be across a wide range of applications. Still this seems very difficult as one can normally not track these relations very easily.

3.4.8 Collaborative Filtering

Collaborative filtering (also known as "social filtering"/"community knowledge") is the process of filtering documents by determining what documents other users with similar interests and/or needs found relevant. This technique is already widely used by some very popular web sites. For example, Amazon.com uses collaborative filtering to recommend books to you that were read by people with similar interests. In the same way the citeseer research index site recommends articles that, according to the behavior of site users, are related. Such systems (Amazon, citeseer) could serve as source for (semi-) automatic ontology instantiation: given an instance document, the system can recommend other instances, which, according to user actions are semantically close.

A created ontology can be further extended. One good approach for this is Google Set. Google Set[1] is a new feature of Google which returns a set of related items given a few examples. For the set (eye, leg, mouth) it generates a list of body parts. It works fine with nouns, and it is less accurate for verbs. This is another possible support for (semi-) automatic ontology instantiation.

[1]http://labs1.google.com/sets

Chapter 4

Update

In a P2P environment, one cannot expect any maintenance to happen on the ontologies (in fact, users will often not know what is in the ontologies on their machine). At the same time it is a highly dynamic system. As a result, we must design mechanisms that allow the ontologies to update themselves, in order to cope with ontological drift. Based on the queries and answers elsewhere in the P2P network, ontologies will have to adjust their own definitions accordingly. This has to be done on a local level and in the context of the distributed network.

4.1 Ontology Update

Assuming that the changes to the ontology are known, different kinds of update must be pursued to keep consistency.

4.1.1 Local Update

An overview of ontology evolution is given by [NK02]. In general there are several dimensions to respect:

- Instance Expansion
- Structure Expansion
- Axioms Preservation

4.1.2 Area Update

After the local ontology has been updated the new information has to be distributed in the peer-to-peer network. Depending on the system this requires different levels of adaption.

An indexing system needs new information about what to save in its index tables. Other systems have to update the routing tables as the content of the neighbor changed. If the content is cached at more than one point, all these points have to be informed and updated. An environment using broadcast to propagate the queries no update is required, as each peer decides for itself if the query is relevant or not.

4.2 Other Structure Update

Updates have been a source for problems for a long time. In the database community these problems have been addressed and are solved with standard methods. On the other hand it is still a problem when having distributed repositories and indexes, because one has to ensure that no inconsistencies arise. Special algorithms are used to achieve a consistent state throughout the network. At this point one can mention update broadcasts, registering services, or expiration times. Aspects from distributed databases can be certainly taken over to our research field.

Chapter 5

Selection Service

SWAP consists of a distributed network of peers. Each peer has its own representation of knowledge. In order to receive the right answers without flooding the peer network one must determine and ask the “right” peer. Several existing mechanisms for knowledge discovery and peer selection will be investigated in this chapter.

5.1 Query

A user accesses the system with a specific goal in mind. This goal has to be transformed into a machine readable request. There are various ways of representing this request.

5.1.1 Keywords

The most simple format for a request is a list of keywords. At times there are more than one parameter which are assigned keywords (e.g. title, author, year). Keyword queries are most common today, but lack the possibility of handling complex context aware structures as intended in our project. Database search mechanisms as well as every Internet search engines uses keywords as query input.

5.1.2 QEL - Query Language

Edutella[NWQ^{+}02] follows a layered approach for defining the query exchange language, which is based on RDF. Currently the language levels RDFQEL- 1, -2, -3, -4 and -5 are defined, differing in expressiveness.

- The most simple language (RDF-QEL-1) can be expressed as unreified RDF graph. It can be interpreted as a logical (conjunctive) formula that is to be proven from a knowledge base.

- Extending REF-QEL-1 with disjunction leads to RDF-QEL-2.
- RDF-QEL-3 allows conjunction, disjunction, and negation of literals. It is essentially Datalog.
- RDF-QEL-4 allows recursion to express transitive closure and linear recursive query definitions, compatible with the SQL3 capabilities.
- RDF-QEL-5: Further levels allow arbitrary recursive definitions in stratified or dynamically stratified Datalog, guaranteeing one single minimal model and thus unambiguous query results.

Depending on the query requirements one can choose among the different levels of QEL. The lower levels have been so far used successfully.

5.1.3 TRIPLE

TRIPLE is a query and transformation language for RDF. Its core is a syntactical extension of Horn logic similar to F-Logic, but specialized for the requirements on the Semantic Web by making Web resources, (RDF) models, and statements first class citizens.

Its main purpose is to query web resources in a declarative way, e.g. for intelligent information retrieval based on background knowledge like ontologies and search heuristics.

TRIPLE's layered architecture allows extensions of RDF to be implemented as extension modules (via parameterized models). Simple object-oriented extensions like RDF Schema can be directly implemented with the extended Horn logic features of TRIPLE, other extensions like DAML+OIL are realized via interaction with external reasoning components like a description logics classifier.

TRIPLE's model concept (esp. the parameterized models) enables the transformation of models, thus enabling knowledge base and ontology mapping/integration tasks which are needed in distributed settings as the semantic web. Since models are first class citizens in TRIPLE, modal functionalities as needed in agent communication are also provided (e.g., agent A believes statements in model M, which has been received from agent B).

TRIPLE is currently being under development. A first implementation of TRIPLE based on XSB is available[1]. In this version, all RDF data and TRIPLE rules are true compiled into a single PROLOG program, therefore restricting the size of the knowledge base to what the underlying PROLOG system (i.e., XSB) can handle. Future versions will implement the complete TRIPLE language and allow querying distributed RDF data a without compiling remote data to the local (PROLOG) knowledge base.[SD01]

[1]http://www.dfki.uni-kl.de/frodo/triple/

5.1.4 RQL

RQL (RDF Query Language) is a typed language following a functional approach (a la ODMG-OQL) and supports generalized path expressions featuring variables on both labels for nodes (i.e., classes) and edges (i.e., properties). RQL relies on a formal graph model (as opposed to other triple-based RDF QLs) that captures the RDF modeling primitives and permits the interpretation of superimposed resource descriptions by means of one or more schemas.

The novelty of RQL lies in its ability to smoothly combine schema and data querying while exploiting - in a transparent way - the taxonomies of labels and multiple classifications of resources. An RQL Interpreter has been implemented in C++ on top of an ORDBMS (PostgreSQL) using a standard client-server architecture for Solaris and Linux platforms. It consists of three modules (a) the Parser, analyzing the syntax of queries; (b) the Graph Constructor, capturing the semantics of queries in terms of typing and interdependencies of involved expressions; and (c) the Evaluation Engine, accessing RDF descriptions from the underlying database via SQL3 queries. [KAC$^+$02]

AIdministrator, has some experience on RQL using it in the Sesame system.

5.1.5 KQML

KQML [FFMM94] or the Knowledge Query and Manipulation Language is a language and protocol for exchanging information and knowledge. It is part of a larger effort, the ARPA Knowledge Sharing Effort which is aimed at developing techniques and methodology for building large-scale knowledge bases which are sharable and reusable. KQML is both a message format and a message-handling protocol to support run-time knowledge sharing among agents. KQML can be used as a language for an application program to interact with an intelligent system or for two or more intelligent systems to share knowledge in support of cooperative problem solving. KQML focuses on an extensible set of performatives, which defines the permissible operations that agents may attempt on each other's knowledge and goal stores. The performatives comprise a substrate on which to develop higher-level models of inter-agent interaction such as contract nets and negotiation. In addition, KQML provides a basic architecture for knowledge sharing through a special class of agent called communication facilitators which coordinate the interactions of other agents. The ideas which underlie the evolving design of KQML are currently being explored through experimental prototype systems which are being used to support several testbeds in such areas as concurrent engineering, intelligent design and intelligent planning and scheduling. [2]

[2] http://www.cs.umbc.edu/kqml/

5.1.6 DQL

DQL [HT02] is a proposal for a query language to query DAML+OIL. The standardization effort towards a general excepted representation language for ontologies has almost succeeded. To exploit this, the Semantic Web community in now discussing to come up with a corresponding query language. [3] First results demonstrate, that a conjunctive query language, like DQL will be, can enrich knowledge representation languages like DAML+OIL. In any case, there does not exist an implementation of DQL yet, but it is worth following the ongoing discussion.

5.2 Structuring the Peer-to-Peer Network

A network which is supposed to be queried needs some kind of regular structure. How one can organize information on the network will be described in this section.

5.2.1 Hash-Identifiers

An approach tailored to the network aspect of peer-to-peer is to use hash-identifiers for traversing the network. Each file, document, or object is assigned a hash-identifier. If at the same time a query can be transformed to such a hash-identifier, a very efficient search and retrieval can be done.

Short Description of Hash-Identifiers: A hash-identifier is a numerical unique representation of a file, document, or object on a peer. They are calculated using a hashing function with a specific input.

Content of Identifiers: The identifiers can cover various aspects of a peer.

- Physical Location: One idea is to respect the physical location of the peer. This idea is related to the IP-protocol of the Internet where each address is directly assigned to a specific hardware.
- Arbitrary: Many systems use an arbitrary number. This way the peer has a unique ID independent from where it is logged on in the network. It is also independent from the content provided by the peer. A well known example are the identifiers of instant messengers.
- Content: Another possibility for setting up the identifier is to link it to the content on the local peer. The identifier will therefore present an abstract of the content.

[3] http://www.daml.org/listarchive/joint-committee/

There is more than one possibility to create keys[Lan99]: content hash keys, which take the data as basis, and keyword signed keys, which take keywords as basis. Identifiers of this kind are not fixed, but change dynamically with the document.

Multidimensional Keys: Especially when using content to describe a peer one has to consider including more than one dimension into the hash-key. The Content Addressable Network (CAN) is organized in a d-dimensional torus whereas content and queries, in form of (key,value) pairs, are mapped into d dimensions using global hash functions [BW01]. The number of dimensions d has to be defined in the beginning though and can not be changed dynamically as one would hope to do for dynamic content.

Problems: A problem of search using hash-tables is the lack of possibilities to send fuzzy requests. Hash-identifiers as an exact intermediator can not deal with requests of this type. This is no problem for arbitrary or location keys, but for content search this does play a role. Nevertheless there are approaches for this problem [HHH02].

5.2.2 Indexes

Indexing systems require some form of registration. Each peer has to log on to the system by sending the index server(s) information about its state and content. Only this way the system will know which peers are around and where to send an incoming request to. This requires an agreed advertisement template. The content of the indexes should naturally contain some condensed form of the content of connected peers. Normally this would comprise e.g. a list of keywords. Indexes are easy to maintain when having a centralized server, in a dynamic multi-server environment additional protocols for passing around and updating information become necessary.

5.3 Peer Discovery

Having a well structured network and a well edited query, the next task is to find the peer with the requested information. An overview of approaches to find this information will follow. This includes traditional approaches but also approaches which we can adapt for use in our peer to peer network.[Cod02]

5.3.1 Flooding Broadcast

The basic approach is to simply flood the network with the query request. Each peer hands on the request to all of its neighbors. This method will very soon entail a stage of network saturation. To prevent congestion different algorithms extend the basic approach, e.g. caching of results or intelligent forwarding algorithms. It is absolutely essential to define a maximum number of hops, meaning the number of steps from one peer to the next peer.

5.3.2 Fireworks

This approach [NS02a] is related to the flooding approach. The idea of fireworks is for each peer to route the request to one single neighbor only. As soon as the current peer knows something about the request i.e. understands the query words, flooding broadcast with a relatively low number of maximum hops is started. This algorithm ensures that flooding is only done in the right domain. Network load is therefore minimized.

5.3.3 Routing (ID)

If the local peer can produce a hash-identifier directly from the query further search is easy and fast. There are several approaches which allow direct routing of the query or request to the right peer. Routing can be performed from the source to destination along the coordinate that reduces the remaining distance to the destination.

- CAN uses a d-dimensional key to route the request.[BW01]
- PASTRY routes the request by utilizing a table split in three groups: a local leaf set with resources which can be directly accessed, a routing table, and third a neighboring set. The routing table is used to forward requests to nodes which come closest to the key of the query [RD01].
- Tapestry uses the so called Plaxton routing. Each node can be reached by following one digit of the ID at a time. With each routing step the number of digits matching the destination ID rises (e.g. with the goal being 123, a possible path would be 000, 020, 120, 123). The difficulty of Tapestry is creating the IDs in the network. [ZKJ01]
- Bloom Filters allow using less complex IDs. The drawback of this approach is that the filters may yield false positives, i.e. they know which nodes do not have the requested information. If the path was wrong, the query is traced back and another path is tried. Wide areas of the network can be directly excluded from search, but the rest will have to be checked more or less complete[RK02].
- Hypercube topologies provide optimal paths to find a peer with a specific ID. Besides it is maintained locally and can therefore be built easily. The joining and removal of peers to the hypercube comes now under scrutiny, but it seems manageable even for a big number of peers. The protocol is currently implemented on a JXTA-based infrastructure.[SSDN02]

5.3.4 Selective Forwarding (Content)

Using selective forwarding, queries are forwarded only to peers which are considered to be likely to either solve the request or locate a better qualified peer. These peers can be chosen on the basis of various algorithms: a content routing table (e.g. filled by advertisements of each peer), or a list of peers which have often returned good answers. Unfortunately this reasonable idea implies some difficulties. An adequate model for creating the routing table of each peer needs to be designed. This requires a learning algorithm as the peer-to-peer network dynamically evolves. It is also not intuitive which content to use for routing purposes (possibly all?) [CW02].
Forwarding can be considerably sped up by setting up a few random long distance connections. This corresponds to Milgram's experiment who sent some letters across the US by just letting people handing them on to friends. [Hon99]

5.3.5 Centralized Indexes

Another approach is to have a central index server. Each peer can route its request to this server and the server returns which peer has the answer to the specific query. This will not be further pursued as a central server does not meet the goals of a decentralized peer-to-peer network. A well known example was Napster, but also the wide majority of commercial enterprise peer-to-peer systems use this approach.

5.3.6 Distributed Indexes, Super-Peers

A solution set up in between a completely distributed peer environment and a centralized server is to use distributed indexes. Not every peer knows about the whole network but super-peers collect the information from their neighboring peers. Different papers propose this solution as the most efficient one[YGM02]. Some algorithm is needed to define the peers which are supposed to act as index peers (super-peers), as this doesn't have to be fixed from the beginning. The index is filled by the content advertised by each individual peer. This algorithm should take into account parameters as bandwidth and CPU-speed of the peers. Performance and scalability of such an approach are very good. The efficiency can be further improved by using the described hash-tables for routing purposes.

5.4 Information Discovery

As soon as the right peer has been found one more step is required. The right information has to be extracted from the knowledge base. This topic is covered by matching and inferencing, which are described in chapter 6.

Chapter 6

Alignment, Mapping, Inference Engine

Different peers will use different, though overlapping ontologies. Alignment, mapping, and inference tools will have to cope with the different ontologies, even though no alignments are explicitly specified. But for comparison and merging considerations a mapping is necessary. An inference engine for ontologies must be able to ask and answer queries to peers in a robust, scalable, often locally contained, manner.

6.1 Matching

Matching of ontologies has been a research topic in various papers. There are different levels of matching. This starts with simple comparisons on instance level. The next level is to check if single concepts are equal. And the chain ends when checking whole trees to determine similarity, with this being the preliminary stage for ontology merging. To achieve good results the following strategies have to be combined.

6.1.1 Text

A very intuitive and comprehensible way to compare ontologies is to simply compare words. The textual comparison can be done on different levels of complexity:

- textual matching: exact comparison of words
- ignorable character removal; stemming
- de-hyphenation
- stop term removal
- substring matching

- content matching
- thesaurus matching

This list is adapted from [MGJ01].

6.1.2 Structure/Graph

The other possibility is to do comparison based on the structure. A thorough analysis can be done checking trees, cycles, etc. If similarity is found the two graphs are connected at this point. [MGMR01] gives a general overview on graph matching. An algorithm, which includes considerations about sub- and super-concepts, is included. An approach which can be used in an ontology environment is the RDF graph matching in [Car02]. Besides an example and the easements of RDF are presented in this paper.

6.2 Merging

Merging describes the task of merging two ontology structures to build one new structure.
PROMPT [NM00, NM01] is an interactive ontology-merging tool that guides the user through the merging process making suggestions, determining conflicts, and proposing conflict -resolution strategies. Besides classical comparison methods their focus here lies on matching based on the concept-representation structure of ontologies. The result is a semi-automatic tool called PROMPT, which does propositions to the user who can then easily merge two different ontologies.
[Ome01b] sets up a procedure on how to integrate existing structures: merge classes, create a superclass, rename classes, mark classes as disjoint, add a subclass-of relation, and remove a subclass-of relation between other pairs of classes. The presented algorithm can be summarized as name preprocessing, comparison of class names, comparison of attribute sets, comparison of attributes, and a final step of decision making.
[AS01] has a slightly different focus in integrating documents from different sources into a master catalog. The master catalog is not changed. The approach is based on a Naive Bayes Classification.
A very good and complete algorithm for full-automatic integration is provided by [DMDH02]. The proceeding in their GLUE-system comprises three steps: the distribution estimator, the similarity estimator, and the relaxation estimator. First the distributions as P(A,B) are calculated, specifying whether an entity is a sub-entity of A and B, with A and B being entities from the two ontologies. This is done by learning the characteristics of a class. The similarity can be estimated using the standard Jaccard-similarity. The relaxation estimator includes considerations about the neighborhood or other general rules e.g. if the children of two nodes match, the nodes also match.

The overall aim of the KRAFT (Knowledge Reuse and Fusion / Transformation)[1] project is to enable the sharing and reuse of information contained in heterogeneous databases and knowledge systems. In this context the information is exchanged rather than merged. Facilitators match requests and services based on a KQML expression.

6.3 Inference Methods

Acknowledgement: I would like to thank Sean Bechofer for providing me with an electronic version of the interface definition saving many hours of my valuable time.

The intelligent behaviour of SWAP nodes with respect to communication and query answering will heavily depend on reasoning methods over terminological knowledge. The aim of the project however is not the development of new methods for terminological reasoning, but rather to use reasoning methods that have proven to be useful for the semantic in order to facilitate the cooperation of peers in the network. Consequently, existing reasoners should be used in the context of the project whenever possible. Currently, we face a situation, where a number of reasoners for terminological knowledge exist, each having its own reasoning capabilities and its own way of representing and interacting with knowledge. In order to overcome this situation, the Description Logics Implementation Group (`http://dl.kr.org/dig/`) is working on a standardized interface that will allow to communicate with different reasoners in a uniform way.

In the following we present the first version of the interface definition as a basis for enabling SWAP nodes to access inference services provided by external reasoners.

6.3.1 The Logical Language

In order to ensure a predictable behavior of the interface regardless of the underlying reasoner the logical language that serves as a basis for the reasoning process has to defined. This holds for the syntactic description of the knowledge but especially for the intended meaning. For this purpose the interface defines an XML-based syntax for an expressive description logic ($\mathcal{SHOIQD}_n^-$) which is rich enough to represent DAML+OIL ontologies. The semantics of this language is based on an interpretation mapping $\mathcal{I}$ into an abstract domain $\Delta^\mathcal{I}$ in the following way:

$$CN^\mathcal{I} \subseteq \Delta^\mathcal{I}$$
$$RN^\mathcal{I} : \Delta^\mathcal{I} \rightarrow 2^{\Delta^\mathcal{I}}$$

[1]http://www.csc.liv.ac.uk/%7Ekraft/

$$FN^{\mathcal{I}} : dom(FN^{\mathcal{I}}) \rightarrow \Delta^{\mathcal{I}}$$
$$AN^{\mathcal{I}} : dom(AN^{\mathcal{I}}) \rightarrow \mathcal{S} \cup \mathcal{Z}$$
$$IN^{\mathcal{I}} \in \Delta^{\mathcal{I}}$$

This mapping interprets individual names (IN) as elements and concept names (CN) as subsets of the abstract domain. Further, role names (RN) are interpreted as functions assigning sets of related objects to individuals. Two special cases of roles are explicitly considered. Firstly, functional role names (FN) that are mapped to a partial function that assign a single related object to individuals and attributes names (AN) that are mapped on partial functions from individuals to and integer ($\mathcal{Z}$) or a string ($\mathcal{S}$). Besides this general definitions, the logical language provides possibilities for further restricting the interpretation of concepts and roles. These kinds of restriction are the basis for deciding whether a class definition is equivalent, more specialized or more general than another. Formally, we can decide whether one of the following relations between two expressions holds in a knowledge base] K:

- An instance a is said to be member of a concept C, denoted as $a : C$ iff $I^{\mathcal{I}} \in C^{\mathcal{I}}$
- A pair of instances (a, b) is said to be member of a Relation R, denoted as $(a, b) : R$ iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$.
- A concept C is subsumed by another concept D, denoted as $C \sqsubseteq D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.
- A concept C is equivalent to another concept D, denoted by $C \equiv D$ iff $C^{\mathcal{I}} = D^{\mathcal{I}}$

The main task of a reasoner is to compute these relations based on the definition of concepts as more complex reasoning tasks can often be phrased in terms of subsumption, equivalence or membership.

Concepts

The basic notions of the knowledge model assumed by the interface are the notions of concepts and instances. The supported logical language therefore contains language elements for representing the universal concept `top` with is equivalent to the complete domain $\Delta^{\mathcal{I}}$ as well as its negation, the empty concept `bottom`. Further, arbitrary concept and individual names can be referred to, and a concept can be defined extensionally but enumerating its members. The table below summarizes the corresponding syntax and semantics.

Concrete XML Form	**Semantics**
`<top/>`	$\Delta^{\mathcal{I}}$
`<bottom/>`	$\emptyset$
`<catom name="CN"/>`	$CN^{\mathcal{I}}$
`<individual name="IN"/>`	$IN^{\mathcal{I}}$
`<iset>I1...In</iset>`	$\{I1^{\mathcal{I}}, \ldots, In^{\mathcal{I}}\}$

Boolean Operators

In order to build more complex concept definitions, the logical language contains the Boolean operators that can be applied to concept names and also to concept expressions which will be discussed later. The semantics of concept disjunction and conjunction operators is defined in terms of the well-known set operators union and intersection applied to their interpretation. Negation is defined as the complement set with respect to the interpretation domain.

Concrete XML Form	Semantics
`<and>E1...En</and>`	$E1^{\mathcal{I}} \cap \ldots \cap En^{\mathcal{I}}$
`<or>E1...En</or>`	$E1^{\mathcal{I}} \cup \ldots \cup En^{\mathcal{I}}$
`<not>E</not>`	$\Delta^{\mathcal{I}} \setminus E^{\mathcal{I}}$

Property Restrictions

Concepts can not only be defined by enumerating their instances but also intentionally in terms of constraints on the relations members of the concept have to other instances in the domain. The language allows to claim that members of a class have to be related to at least one object of a specific type by a specific relation and that members of a concept may only be related to objects of a specific type by a specific relation. Further, upper and lower bounds for then umber of objects of a certain type related by a specific relation can be given .

Concrete XML Form	Semantics
`<some>R E</some>`	$\{d \in \Delta^{\mathcal{I}} \vert R^{\mathcal{I}}(d) \cap E^{\mathcal{I}} \neq \emptyset\}$
`<all>R E</all>`	$\{d \in \Delta^{\mathcal{I}} \vert R^{\mathcal{I}}(d) \subseteq E^{\mathcal{I}}\}$
`<atmost num="n">R E</atmost>`	$\{d \in \Delta^{\mathcal{I}} \vert \vert R^{\mathcal{I}}(d) \cap E^{\mathcal{I}} \vert \leq n\}$
`<atleast num="n">R E</atleast>`	$\{d \in \Delta^{\mathcal{I}} \vert \vert R^{\mathcal{I}}(d) \cap E^{\mathcal{I}} \vert \geq n\}$

Concrete Domain Expressions

The language further allows to restrict the allowed value of the attributes of the members of a class. Different from general relations, attributes are are functional by definition and they do only relate members of a certain concept to a value of a predefined data type. The current version of the interface supports the data types string and integer. Possible restrictions of the value of an attribute available in the language are in first place the trivial restriction to the complete value range of an attribute. Further, the language allows to specify a concrete value, upper and lower bounds for the value as well as a value range. These options are available for string and for integer values.

Concrete XML Form	Semantics
`<defined>A</defined>`	$\{d \in \Delta^{\mathcal{I}} \mid d \in dom(A^{\mathcal{I}})\}$
`<stringmin val="s">A</stringmin>`	$\{d \in \Delta^{\mathcal{I}} \mid s \leq_{\mathcal{S}} A^{\mathcal{I}}(d)\}$
`<stringmax val="s">A</stringmax>`	$\{d \in \Delta^{\mathcal{I}} \mid A^{\mathcal{I}}(d) \leq_{\mathcal{S}} s\}$
`<stringequals val="s">A</stringequals>`	$\{d \in \Delta^{\mathcal{I}} \mid A^{\mathcal{I}}(d) =_{\mathcal{S}} s\}$
`<stringrange min="s" max="t">A</stringrange>`	$\{d \in \Delta^{\mathcal{I}} \mid s \leq_{\mathcal{S}} A^{\mathcal{I}}(d) \leq_{\mathcal{S}} t\}$
`<intmin val="i">A</intmin>`	$\{d \in \Delta^{\mathcal{I}} \mid i \leq A^{\mathcal{I}}(d)\}$
`<intmax val="i">A</intmax>`	$\{d \in \Delta^{\mathcal{I}} \mid A^{\mathcal{I}}(d) \leq i\}$
`<intequals val="i">A</intequals>`	$\{d \in \Delta^{\mathcal{I}} \mid A^{\mathcal{I}}(d) = i\}$
`<intrange min="i" max="j">A</intrange>`	$\{d \in \Delta^{\mathcal{I}} \mid i \leq A^{\mathcal{I}}(d) \leq j\}$

Role Expressions

As relations are the main basis for constraining the set of members of a concept, the interface definition also includes a set of language elements for defining relations. These elements include reference to general roles (binary relations between objects in the domain), functional roles, called features and to attributes. Further, the inverse of a role and chains of functional rules can be described.

Concrete XML Form	Semantics
`<ratom name="RN"/>`	$RN^{\mathcal{I}}$
`<feature name="FN"/>`	$FN^{\mathcal{I}}$
`<attribute name="AN"/>`	$AN^{\mathcal{I}}$
`<inverse>R</inverse>`	$\{(d, d') \mid (d', d) \in R^{\mathcal{I}}\}$
`<chain>F1...Fn A</chain>`	$F1^{\mathcal{I}} \circ \ldots \circ Fn^{\mathcal{I}} \circ A^{\mathcal{I}}$

6.3.2 Interaction with a Reasoner - Tell

The main function of the logical language presented above is to serve as a basis for communicating with a logical reasoner. In principle there are two types of interactions, the first one discussed in this section is the so-called `tell` mode, where knowledge we want to reason about is shipped to the reasoner. This is done by special commands described below. Some of these commands refer to the definition of concepts and/or roles. In these cases the logical language introduced above has to be used to represent these definitions.

The first thing that we might want to tell a reasoner are the names of the knowledge elements we want to reason about. For this purpose, the interface specifies the following operations for introducing concepts, roles, features, attributes and individuals.

```
<defconcept name="CN"/>
```

```
<defrole name="RN"/>
<deffeature name="FN"/>
<defattribute name="AN"/>
<defindividual name="IN"/>
```

In order to add further information about these elements in terms of axioms that restrict the interpretation of the names, there are more operations that will be discussed in the following.

Concept Definitions

Using the interface definition, we can define the relation between concepts in a knowledge base. More specifically, we can state that a concept expression subsumes another expressions and that two concept expressions are equivalent or disjoint. In most cases, this is used to connect a concept name to a restricting definition phrased in the logical language described in the previous section. In general, however, we are free to state these relations between two complex expressions.

Concrete XML Form	Semantics
`<impliesc>C1 C2</impliesc>`	$C1^{\mathcal{I}} \subseteq C2^{\mathcal{I}}$
`<equalc>C1 C2</equalc>`	$C1^{\mathcal{I}} = C2^{\mathcal{I}}$
`<disjoint>C1...Cn</disjoint>`	$C1^{\mathcal{I}} \cap Cn^{\mathcal{I}} = \emptyset, i \neq j$

Role Definitions

In a similar way, assertions can be made about relations. We can define a role to subsume or to be equivalent to another one. Further, the range and the domain of a role can be restricted to a certain concept expression on a global level and the range of an attribute can be defined to be one of the data types string or integer. Finally, we can define a role to be transitive or functional.

Concrete XML Form	Semantics
`<impliesr>R1 R2</impliesr>`	$R1^{\mathcal{I}} \subseteq R2^{\mathcal{I}}$
`<equalr>R1 R2</equalr>`	$R1^{\mathcal{I}} = R2^{\mathcal{I}}$
`<domain>R C</domain>`	$dom(R^{\mathcal{I}}) \subseteq C^{\mathcal{I}}$
`<range>R C</range>`	$ran(R^{\mathcal{I}}) \subseteq C^{\mathcal{I}}$
`<rangeint>A</rangeint>`	$ran(A^{\mathcal{I}}) \subseteq \mathcal{Z}$
`<rangestringA</rangestring>`	$ran(A^{\mathcal{I}}) \subseteq \mathcal{S}$
`<transitive>R</transitive>`	$R^{\mathcal{I}} \circ R^{\mathcal{I}} \subseteq R^{\mathcal{I}}$
`<functional>R</functional>`	$\forall d \in dom(R^{\mathcal{I}}).\lvert R^{\mathcal{I}}(d)\rvert = 1$

Individual Definitions

Besides concepts and relations, the interface can be sued to define assertional knowledge in terms of individuals, their relations and attributes. For this purpose there are operators to state that an individual is member of a concept expression, that two individuals are related by a specific role and that for assigning a value to the attribute of an individual. Further, there are special constructors to introduce concrete string and integer values to be assigned as an attribute value.

Concrete XML Form	Semantics
`<instanceof>I C</instanceof>`	$I^{\mathcal{I}} \in C^{\mathcal{I}}$
`<related>I1 R I2</related>`	$I2^{\mathcal{I}} \in R^{\mathcal{I}}(I1^{\mathcal{I}})$
`<toldValues>I A V</toldValues>`	$I^{\mathcal{I}} \in dom(A^{\mathcal{I}}) \wedge A^{\mathcal{I}}(I^{\mathcal{I}}) = V^{\mathcal{I}}$
`<sval>s</sval>`	s
`<ival>i</ival>`	i

Example

The actual interaction with a reasoner is done using the HTTP post statement. the body of the message sent to the reasoner is an XML encoding of a series of tell operations that might use parts of the logical language described in the first section. The example below show a message that defines a simple knowledge base to be used by the reasoner. The knowledge base contains the concepts *person* and *vehicle*, the role *drives* as well as the concept *driver* that is defined in terms of the others stating that a driver is a person who drives some vehicle.

6.3.3 Interaction with a Reasoner - Ask

The reason for shipping knowledge to a reasoner using the tell operations is that we want to ask the reasoner questions about the knowledge that require logical reasoning to be answered. Consequently, the interface also defines a set of operations for asking questions about the knowledge being told to the reasoner. Again, some of these operations may use the logical language of the first section to specify the concrete question being asked.

Signature

The simplest kind of requests considered by the interface are requests about the names of concepts, relations and individuals in the current knowledge base of the reasoner (Signature). We can ask for all concept names, for all role names and for all individual names.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<tells
  xmlns="http://dl.kr.org/dig/lang"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://dl.kr.org/dig/lang
  http://potato.cs.man.ac.uk/dig/level0/dig.xsd">
    <clearKB/>
    <defconcept name="driver"/>
    <equalc>
        <catom name="driver"/>
        <and>
            <catom name="person"/>
            <some>
                <ratom name="drives"/>
                <catom name="vehicle"/>
            </some>
        </and>
    </equalc>
    <defconcept name="person"/>
    <defconcept name="vehicle"/>
    <defrole name="drives"/>
</tells>
```

Figure 6.1: Sample Tells

Concrete XML Form	**Semantics**
`<allConceptNames/>`	$\mathcal{CN}$
`<allRoleNames/>`	$\mathcal{RN}$
`<allIndividuals/>`	$\mathcal{IN}$

Concepts

As the notion of a concept is a central one in terminological reasoning, there are a number of possible queries about concepts and their relation to each other. First of all, it is possible to ask whether the definition of a concept is consistent with the knowledge of the reasoner (satisfiability). Further we can ask whether two concept definitions are disjoint or in a subsumption relation top each other. Based on the notion of subsumption, the interface allows to ask more general questions about the concept hierarchy. In particular, we can ask for the parents and children of a specific concepts in the hierarchy as well as for ancestors and descendants. Finally, it is possible to retrieve concepts that are equivalent to a given one.

Concrete XML Form	Semantics
`<satisfiable>C</satisfiable>`	$K \not\models \mathtt{C} \equiv \bot$
`<subsumes>C1 C2</subsumes>`	$K \models \mathtt{C1} \sqsubseteq \mathtt{C2}$
`<disjoint>C1 C2</disjoint>`	$K \models (\mathtt{C1} \sqcap \mathtt{C2}) \equiv \bot$
`<parents>C</parents>`	$\{D \,\vert\, K \models D \sqsubseteq C, \nexists D' : D \sqsubseteq D' \sqsubseteq C\}$
`<children>C</children>`	$\{D \,\vert\, K \models C \sqsubseteq D, \nexists D' : C \sqsubseteq D' \sqsubseteq D\}$
`<ancestors>C</ancestors>`	$\{D \,\vert\, K \models D \sqsubseteq C\}$
`<descendants>C<descendants/>`	$\{D \,\vert\, K \models C \sqsubseteq D\}$
`<equivalents>C</equivalents>`	$\{D \,\vert\, K \models D \equiv C\}$

Roles

As mentioned above, roles can also be arranged in a subsumption hierarchy. Consequently, we can ask the same information about this hierarchy as about the concept hierarchy. In particular, we can ask for the parents, the children, the ancestors and the descendants of a role in that hierarchy.

Concrete XML Form	Semantics
`<rparents>R</rparents>`	$\{P \,\vert\, K \models P \sqsubseteq R, \nexists P' : P \sqsubseteq P' \sqsubseteq R\}$
`<rchildren>R</rchildren>`	$\{P \,\vert\, K \models R \sqsubseteq P, \nexists P' : R \sqsubseteq P' \sqsubseteq P\}$
`<rancestors>R</rancestors>`	$\{P \,\vert\, K \models P \sqsubseteq R\}$
`<rdescendants>R<rdescendants/>`	$\{P \,\vert\, K \models R \sqsubseteq P\}$

Individuals

Questions concerning individuals will be of special importance when actually reasoning about data. The Interface allows to query a reasoner about the nature of individuals in different ways. The first one is the relation between concepts and individuals. With respect to this relation we can test whether an individual belongs to a concept and query all individuals that belong to a concept as well as all concepts an individual belongs to. The second kind of questions concern the relations between individuals. Here we can retrieve all pairs of individuals that are related by a certain role and we can ask for all individuals that are related to a given individual by a specific relation. Finally,w e can ask for the value of a specific attribute of an individual.

Concrete XML Form	Semantics
`<instances>C</instances>`	$\{I \mid K \models I : C\}$
`<types>I</types>`	$\{C \mid K \models I : C\}$
`<instance>I C</instance>`	$K \models I : C$
`<roleFillers>I R</roleFillers>`	$\{J \mid K \models (I, J) : R\}$
`<relatedIndividuals>R</relatedIndividuals>`	$\{(I, J) \mid K \models (I, J) : R\}$
`<toldValues>I A</toldValues>`	

6.3.4 Example:

the communication with a reasoner is done in the same way as for the tell-operations. An HTTP post request is send with an XML message containing the ask operations which are a combination of the requests described in this section and the logical language. The example below shows a message send to a reasoner with the following three requests:

q1: Is the concept vehicle consistent ?

q2: Give me all descendants of the concept of persons that drive some vehicle.

q3: Give me all the concepts the individual John Smiths belongs to.

6.3.5 Interaction with a Reasoner - Responses

In order to guarantee a predictable behaviour of external reasoners, the interface also defines a standard format for the responses a reasoner sends to querying clients. This format mainly defines data structures that are used to pass the names of concepts, relations and classes back as an answer to a query.

Data Structures

The simplest answer a reasoner can return is the error message `<error/>`.

As some of the defined queries require yes/no answers (e.g. asking whether an individual is member of a certain class), the Boolean value `<true/>` and `<false/>` my be returned on a request.

On the next level of complexity, the answer of a reasoner may be an integer or a string value as an answer to the question for the told values of a certain property of an individual. These are returned as `<sval>s</sval>` for a string value s or `<ival>i</ival>` for an integer value i.

```
<?xml version="1.0"?>
<asks
  xmlns="http://dl.kr.org/dig/lang">
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://dl.kr.org/dig/lang"
  http://potato.cs.man.ac.uk/dig/level0/ask.xsd" >
  <satisfiable id="q1">
    <catom name="Vehicle"/>
  </satisfiable>
  <descendants id="q2">
    <and>
      <catom name="person"/>
      <some>
        <ratom name="drives"/>
        <catom name="vehicle"/>
      </some>
    </and>
  </descendants>
  <types id="q3">
    <individual name="John Smith"></individual>
  </types>
</asks>
```

Figure 6.2: Sample Query

More complex responses are those that return sets of names that correspond to concepts, relations or individuals in the knowledge base. In the case of concepts these sets may contain different names for the (logically) same concept. In order to be able to distinguish between names that denote different concepts and different names for the same concept, the data structure for returning sets of concept names is a list of synonym names for classes:

```
<conceptSet>
  <synonyms>S11 ... S1N</synonyms>
  <synonyms>SN1 ... SNM</synonyms>
</conceptSet>
```

The same structure is used to return sets of role names as the problem of having different names for the same element also holds for relations.

```
<roleSet>
  <synonyms>R11 ... R1N</synonyms>
  <synonyms>RN1 ... RNM</synonyms>
</roleSet>
```

In the case of individuals, the interface defines a simpler structure that is just a list of individual names.

```
<individualSet>I1\ldots IN</individualSet>
```

As the answers for queries about the relation of an individual to other individuals expects pairs of individuals as answers the final data structure defined as part of the interface is a set of individual pairs that is represented as follows:

```
<individualPairSet>
  <individualPair>I1 I2</individualPair>
  <individualPair>J1 J2</individualPair>
</individualPairSet>
```

Response Messages

The above data structures are are used to describe the content of messages send by a reasoner as a response to a posted query message. A response message contains a `responses` tag with nested elements each containing the answer to a query. The answer that are not necessary ordered in the same way as the corresponding queries are linked to the query using the `id` attribute whose value corresponds to the id of the query. Error messages can be enriches by a short message also encoded as an attribute and may contain an explanation as free text. The names of classes, relations and individuals are represented using the logical language discussed above. Figure 6.3 shows a possible response message for the example query message shown in figure 6.2.

6.3.6 Reasoners

Provided that the proposed interface is widely adopted by the developers of terminological reasoning systems, it will an important way of receiving general purpose reasoning support for more specific tasks like query answering or re-writing. In the following, we give a brief overview of terminological reasoning systems that might provide useful services if being made accessible via the interface. Further, we discuss the problem of the different expressive power individual reasoners are able to handle and describe the solution to this problem adopted by the description logic implementation group.

```
<?xml version="1.0"?>
<responses
  xmlns="http://dl.kr.org/dig/lang">
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://dl.kr.org/dig/lang
  http://potato.cs.man.ac.uk/dig/level0/response.xsd"
  <error code="99" id="q3" message="No Such Individual">The individua
named does not exist in the knowledge base.</error>
  <true id="q1"/>
  <conceptset id="q2">
    <synonyms>
      <catom name="bus driver"/>
      <catom name="passenger service vehicle operator"/>
    </synonyms>
    <synonyms>
      <catom name="car driver"/>
    </synonyms>
  </conceptset>
</responses>
```

Figure 6.3: Sample Response

Available Reasoners

At the moment there is a number of different reasoning systems available that provide terminological inference services. It is not clear yet which systems will conform to the proposed interface but a rather wide adoption can be assumed. The following is a list of available systems with the reference to a web page where more information can be found. In most cases the system can also be downloaded at the indicated site.

- BOR `http://www.sirma.bg/OntoText/bor/index.html`
- Cerebra `http://www.networkinference.com/`
- CLASSIC `http://www.research.att.com/sw/tools/classic/`
- CRACK `http://www.cs.man.ac.uk/ franconi/crack/`
- DLP `http://www.bell-labs.com/user/pfps/dlp/`
- FaCT `http://www.cs.man.ac.uk/ horrocks/FaCT/`
- GOST `http://lat.inf.tu-dresden.de/systems/gost.html`
- Loom `http://www.isi.edu/isd/LOOM/LOOM-HOME.html`

- RACER `http://kogs-www.informatik.uni-hamburg.de/ race/`

The variety of the systems mentioned above ranges from experimental systems like GOST or CRACK to industrial strength developments such as Cerebra, BOR or CLASSIC. Unfortunately, there are also significant differences in the expressive power of the concept languages that are supported by the different systems. Here the systems range from simple logics decidable in polynomial time (CLASSIC) via reasoners that support almost the complete DAML+OIL languages (FaCT, RACER) to full scale programming systems including logical reasoning (Loom).

Reasoner Capabilities

The solution to the problem of reasoners that support logics of different strengths that has been adopted in the interface definition is to explicitly consider the capabilities of a reasoner. The interface provides the possibility to explicitly request a capabilities description (called identifier) from a reasoning system. The corresponding message is shown in figure 6.4.

```
<?xml version="1.0" encoding="UTF-8"?>
<getIdentifier
  xmlns="http://dl.kr.org/dig/lang"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://dl.kr.org/dig/lang
  http://potato.cs.man.ac.uk/dig/level0/dig.xsd"/>
```

Figure 6.4: Reasoner Identification Request

The reply of the reasoner is a message that contains a information about the subset of the interface in terms of the logical language as well as the ask and tell operations that are supported. The message in figure 6.5 shows an example response of a reasoner that supports Boolean operators and existential quantification as a logical language. Further, the message states that the it is possible to define subsumption relations between concept expressions and to ask whether a concept expression is satisfiable. The corresponding response language is not explicitly defined as it follows from the kind of questions that can be asked. In this case it would only contain the error message and Boolean values.

6.3.7 Implications for SWAP

Developing terminological reasoning methods is not the focus of the SWAP project. Therefore, the strategy should be to be able to use existing solutions without being bound to a specific system. The proposal for a standard interface to terminological reasoning

```
<?xml version="1.0" encoding="UTF-8"?>
<identifier
  xmlns="http://dl.kr.org/dig/lang"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://dl.kr.org/dig/lang
  http://potato.cs.man.ac.uk/dig/level0/identifier.xsd"
  name="FaCT" version="13.0.4" message="FaCT reasoner running on
potato.cs.man.ac.uk (130.88.196.149)">
        <supports>
                <language>
                        <and/>
                        <or/>
                        <not/>
                        <some/>
                </language>
                <tell>
                        <impliesc/>
                </tell>
                <ask>
                        <satisfiable/>
                </ask>
        </supports>
</identifier>
```

Figure 6.5: Reasoner Identification

systems recently proposed by the description logic implementation group seems to be a good choice for being able to use different terminological reasoning systems. For this reason, SWAP technology should use the interface as a mechanism to access reasoning services wherever they are needed. The choice of a specific reasoner to be used in the system should be based on the kind of reasoning support needed. This in turn depends on the choice of the ontology language as well as the concrete needs of the case studies.

6.4 Query Re-Writing

6.4.1 Introduction

In the following we discuss *query rewriting* as a method for facilitating the communication between heterogeneous systems and/or information structures. Our goal is to provide an overview of existing approaches for query rewriting and evaluate their potential use with respect to information exchange in peer to peer systems. We start with a discussion of the basic approach originated from the field of database systems. We then draw our attention to approaches that are of special interest with respect to applying query-rewriting in the context of the semantic web. These include approaches for query rewriting in weakly structured environments and for querying object structures.

Application Areas

Query rewriting plays an important role the field of information systems. Two main applications of rewriting can be distinguished. The first one is concerned with query optimization and aims at rewriting a given user query into an equivalent query that can be executed more efficiently over the given data model. The second main application we are interested in is information exchange between heterogeneous structures. This area contains the following applications.

Database Design Originally, the concept of query rewriting was introduced to achieve independence between physical data model and the data structures used in application systems. This independence is achieved by introducing the logical data model as an intermediate layer between physical model and so-called views on the data model used in applications. This, however, requires that queries posed to the individual views have to be re-formulated in terms of the logical data model in order to be answered.

Information Integration View-based integration has become the standard approach for providing uniform access to heterogeneous information systems. For this purpose a global data model is introduced that provides a common view on the integrated systems. In order to connect the individual information sources with this global schema, one is defined in terms of a view over the other. Again, queries posed to to global model have to be rewritten in order to be retrieve the answer from the different sources.

Peer-to-peer Systems In the context of peer to peer systems the need for query rewriting is even more evident as it does not happen behind the scenes of predefined views on a global model, it is rather essential part of the communication between different peers. Different peers in a system will have different models and even different capabilities for answering queries. In order to request information from another node in the network, peers have to rewrite their queries in order to adapt them to the information model and the capabilities of the target peer.

Levels of Rewriting

Rewriting a query can be done with respect to different aspects that depend on the type of heterogeneity that has to be overcome. We distinguish three different levels of rewriting each corresponding to a specific part of the query.

Structure-Based Rewriting Structure-based rewriting is the classical problem where a query to a specific data structure has to be rewritten into a query over another structure based on mappings between the data structures involved. In the case of conventional database systems query rewriting on the structural level replaces predicates that refer to tables in a relational database by different predicates that correspond to tables in the target database while the constants in the query expression remain unchanged.

Content-Based Rewriting Structure-based rewriting assumes that the target information structure only differs in the way the information is organized. However, it has been argued that information sources can also differ with respect to the actual information content. A typical case is the use of different scales for numerical data (e.g. a different currency for the price of a product). The alignment of these differences also known as context transformation has also been done in query rewriting if a query contains constants.

Capabilities-Based Rewriting Both, structure-based and content-based rewriting only referred to heterogeneity of the information to be queried. In a heterogeneous environment, however, it may as well be necessary to adapt the query language to the capabilities of the system that is queried. This kind of rewriting may include the use of logical operators in a query as well as the use of constants, recursive definitions and non-logical functions (e.g. string matching operators).

6.4.2 Foundations

In this first part we discuss query rewriting in a database setting assuming a relational data model and the use of conjunctive queries over the relational model.

Schemas, Views and Queries

A schema is normally seen as a set of named relations. The positions in the relations also called columns are named and considered part of the schema. The typical way of accessing information is in terms of conjunctive queries of the form:

$$q(\bar{X}) \Leftarrow e_1(\bar{X}_1) \wedge \cdots \wedge e_n(\bar{X}_n)$$

where $e_1, \cdots, e_n$ are relations and $\bar{X}_1, \cdots, \bar{X}_n$ are element tuples of these relations. The result of the query is a set of tuples satisfying $q(\bar{X})$. Multiple queries with the same head clause can be used to drive the union of individual schemas. A view is a named query.

The integration of heterogeneous schemas is normally done using a global schema that is connected to the heterogeneous schemas to be integrated by a number of views. We can distinguish two general approaches:

Global-As-View: In the global-as-view approach every relation in the global schema is defined as a view over the different schemas to be integrated. In especially, the integration is defined by horn rules where the consequence is a relation in the integrated schema and the clauses in the antecedents correspond to relations in the different heterogeneous schemas. Queries to the global schema can easily be answered by unfolding the clauses making use of the view definitions. The unfolded query can be computed using conventional techniques used in database systems. A drawback of this approach is that the independence of the individual information systems is lost due to their combined use in queries. This leads to problems if information sources are added or removed from the integrated system.

Local-As-View: In the local-as-view approach, views are used in the opposite way. In order to connect local schemas with the integrated one views of the following form are used:

$$s_i : e_j(\bar{X}_j) \Leftarrow q_1(\bar{X}_1) \wedge \cdots \wedge q_n(\bar{X}_n)$$

This means that single relations in the schemas to be integrated are used to define more complex information structures in the integrated schema. This also means that additional properties of the integrated information to be included in the global schema can be encoded in the mapping rules. As every local schema is defined as a view over the global one, the independence of the individual sources are preserved. The drawback of this approach is that answering queries over the integrated view is more difficult as it corresponds to the problem of abduction.

Query Equivalence and Containment

In order to understand query rewriting algorithms, we first have to define some basic notions connected with rewriting. First of all, we have to define the relation between two queries, i.e. the original and the rewritten query. With respect to this relation, the notion of query containment and equivalence is often used to define the rewriting task.

Definition 1 (Query Containment and Equivalence) *A query Q_1 is said to be contained in another query Q_2 denoted by $Q_1 \sqsubseteq Q_2$ if for all possible states D of a database the set of tuples computed for Q_1 is a subset of the tuples computed for D_2 ($\forall D, Q_1(D) \subseteq Q_2(D)$). The two queries are said to be equivalent, denoted as $Q_1 \equiv Q_2$ iff $Q_1 \sqsubseteq Q_2$ and $Q_2 \sqsubseteq Q_1$*

Based on these semantic relation between queries, we can distinguish two different kinds of rewriting, namely equivalent rewriting resulting in a rewritten query that is equivalent to the original query and maximally contained rewriting. In the latter case, the rewritten query is contained in the original query. Further the rewritten query is maximal with respect to the containment relation in order to capture as much of the results of the original query as possible.

Definition 2 (Query Rewriting) *Let Q be a query in query language $\mathcal{L}$ and $\mathcal{V}$ a set of views.*

- *A query Q' is an equivalent rewriting of Q in $\mathcal{L}$ over $\mathcal{V}$ if Q' only refers to views in $\mathcal{V}$ and $Q \equiv Q'$.*
- *A query Q' is a maximally contained rewriting of Q over $\mathcal{V}$ if Q' only refers to views in $\mathcal{V}$, $Q \sqsubseteq Q'$ and there is not rewriting Q_1 in $\mathcal{L}$ such that $Q' \sqsubseteq Q_1 \sqsubseteq Q$ and $Q_1 \not\equiv Q'$*

As long as we are concerned with query rewriting on a single database, we are interested in equivalent rewriting in order to ensure that no information is lost. In the case where we rewrite queries across different systems, it is often not possible to compute an equivalent rewriting due to mismatch between different information structures. The system becomes even more complex if we consider capabilities-based rewriting as the definition above assumes a single query language. However, we can extend the definitions by defining equivalent and contained rewriting in query language $\mathcal{L}'$ in a straightforward way.

Beside considering the relation between different queries, we are of course interested in the relation between queries and their correct answers. In particular, we are interested in the set of all answers to a query and its relation to the set of all answers we get for rewritings of this query. It is quite obvious that maximally contained rewriting will result in incomplete results. Further, the fact that queries are defined over views rather than over the actual database leads to problems, because views considered by a query will often not contain all information from the database. In order to deal with this problem, the notion of certain answers of a query is used. A tuple is a certain answer of a query if it is answer for every database state that is consistent with the view definition.

Definition 3 (Certain Answers) *Let Q be a query over a set of views $\mathcal{V} = \{V_1, \cdots, V_m\}$ and $v_1, \cdots, v_m$ are extensions of the Views in $\mathcal{V}$.*

- *A tuple a is a certain answer of the query Q with respect to $\mathcal{V}$ under a closed world assumption given $v_1, \cdots, v_m$ if $a \in Q(D)$ for all database states D with $V_i(D) = v_i$.*
- *A tuple a is a certain answer of the query Q with respect to $\mathcal{V}$ under a open-world assumption given $v_1, \cdots, v_m$ if $a \in Q(D)$ for all database states D with $V_i(D) \supseteq v_i$.*

The distinction between closed and open worlds is necessary, because if we consider the views to be complete with respect to the database leads to a larger set of certain answers that can be deduced from the completeness assumption.

Rewriting Algorithms

The problem of finding equivalent or maximally contained rewriting has been shown to be a hard one. Naive algorithms quickly fail to provide results in reasonable time. For this reason a number of query rewriting algorithms have been developed. We will briefly describe the ideas of three well known algorithms

The Bucket Algorithm The idea of the Bucket algorithm is to reduce the complexity of rewriting by considering sub-goals of the overall problem separately already ruling out combinations that do not lead to a solution. In this process, sub-goals are provided by finding alternative rewritings for the individual predicates in a conjunctive query. For this reason a so-called bucket is created for each predicate and those views that can be used to provide answers for that specific predicate are added to the bucket. In a second step, all consistent combinations of the views in the different buckets are returned as possible rewritings of the original query. While the bucket algorithm leads to a speedup in the first phase by breaking down the problem, it is still necessary to check containment for all combinations of the views in the buckets which has been shown to be computational very expensive.

The Inverse Rule Algorithm The idea of the inverse rule algorithm is to invert the rules that define the views involved. These inverted rules can then be used in order to compute an extension of the views from the actual content of the database. This computation is done by using the Datalog program that contains the inverted rules and the original query. They together constitute the rewriting. Referring to the definition of certain answers given above, we can see that answering the original query can actually be done using the extension of the views involved. The idea of computing inverse rules is very appealing as it can easily be extended to more complex query languages. The main drawback, however, is the need to compute the extension of the views without taking into account which part of the extension is actually needed in order to answer the given query. This especially becomes a problem in the process of actually answering queries using their rewriting, because this process involves computing the extension every time a query is posted.

The MiniCon Algorithm The MiniCon Algorithms is basically an extension of the Bucket algorithm that aims at reducing the effort of constructing the actual rewriting in the second step of the process. In the first step the algorithm creates so-called MiniCon Descriptions for sub-goals of the query to be rewritten. These sub-goals again correspond to predicates in the conjunctive query. In contrast to buckets, a MiniCon description does not contain whole views but rather all the predicates of a view that actually contribute to answering the query. The contribution is determined by trying to establish variable mappings between the corresponding view and the query. In the case where the algorithms is unable to create such a mapping, e.g. due to missing shared variables needed to link the predicates together, the view is not considered for the rewriting. This already leads to a significant reduction of possible re-writings. Further, tracing the predicates that actually contribute to query answering also forms the basis for selecting the best rewriting.

Experiments have shown that the MiniCon Algorithm outperforms the Bucket- and the Inverse-Rule algorithms in terms of computational efficiency.

6.4.3 Semi-Structured Information

Along with the development of the World Wide Web and its lack of strong data structures, the interest in modeling and querying semi-structured information has grown. Many techniques from database research have been extended to cope with XML-based information.

These techniques include query answering and query rewriting. In the following, we will describe a general model for semi-structured information, a corresponding query language that shows many similarities to the RDF data model as well as a view-based rewriting method for queries over semi-structured data.

Querying Semi-structured Data

The OEM data model for semi-structured information is based on a rooted directed graph with labeled nodes (objects) that are assumed to have a unique identity. The id's of the objects in a model are either atomic data or function terms applied to atomic data in order to distinguish them from other objects. The value of a node is determined by the sub-tree it roots. Only leave nodes are assumed to contain unparsed data, i.e. free text that uniquely defines the value of these nodes. In most cases, the model assumed to be a tree or a directed acyclic graph.

Query processing in this data model is based on subtree extraction from the graph that defines th information model. The corresponding query language called TSL (tree specification language) consist of rules similar to Datalog. The head of TSL rules defines the answer graph (actually a tree) and the body defines conditions that must hold for objects in the answer graph. Both the answer graph and the conditions are stated in terms of patterns of the form `<id label {value}>`. These triples refer to nodes in the graph model which can be matched by their id, their label of their value. In order to be able to capture non-atomic values of nodes, the value part of a pattern can not only contain terms but also sets of other patterns. Further the patterns in the rule body can be combined by conjunction. Every TSL query can be converted into normal form, i.e. an equivalent query where every value set only contains one pattern by introducing a new conjunct for every additional pattern in the set.

The meaning of TSL queries is define in terms of assignment mappings from id variables to object ids, from label variables to lables and from value variables to atomic values or subtrees of the data model. For every possible mapping that satisfies all the conditions of the rule body the corresponding nodes are created by instantiating the object patterns contained in the head of the query. Views on the graph model are defined in terms of named TSL queries in the same way as for relational databases.

Query Rewriting

The rewriting problem for semi structured information is defined in the obvious way. The only difference to the previously mentioned approaches in the nature of the query language used. Instead of relation names we have to find re-writings for object patterns of the kind described above. Papakonstantinou and Vassalos propose a rewriting method for TSL that consists of four steps:

Step 1: Find Mappings In the first step mappings between the variables in the body of the query to be re-written and elements of the bodies of the available views are created. The existence of such a mapping is a necessary condition for a successful rewriting. A mapping determines whether a view is useful for answering a query and it shows which part of the conditions in the query body are covered by a view. Conditions not covered by a mapping to one view will have to be considered in mappings to other available views.

Step 2: Construct Candidates In the second step, the mappings identified in the first step are used to construct potential re-writings of the given query. Candidates are constructed by applying the mapping to the view we want to use for re-writing. After the variables in the rule defining the view have been replaced by the mapping its head becomes the body of the query rule. For the case where a view only covers parts of the query, the same is done for all conjuncts in the query body.

Step 3: Compute the Composition In order to verify the result of the re-writing step, the re-written query Q' is tested against the original one Q. This is done a two step process. In the first step, the rewritten query is unfolded by computing its composition with the views that have been used during rewriting. The composition is computed by replacing all view heads in the body of Q' by the corresponding bodies of the views.

Step 4: Test Equivalence In the second step of the validation, the equivalence of the composition of Q' with the different views and the original query Q is tested. If the test succeeds, Q' is a correct rewriting of Q. The equivalence test again exploits the identification of variable mappings. In short, equivalence is present if mappings can be found in both directions.

Using Functional and Schema Dependencies

The general re-writing procedure sketched above often produces re-writings and compositions that contain redundant or hidden restrictions due to functional dependencies between variables in the query. The first kind of dependencies that can be used to identify additional restrictions are key dependencies imposed by the IDs of queried objects. Consider the triples `<A B C>` and `<A D E>` being conjuncts in the same rule body. As the data model is based on unique IDs, we can infer that for every possible binding the fillers of B and D as well as of C and E have to be the same. The second kind of dependencies are imposed by the conformance of the queried data to a specific schema, normally, e.g. a Document Type Definition (DTD). Consider the following simple DTD:

```
<!ELEMENT person (name, address)>
<!ELEMENT name (last, first)>
```

We assume the elements `address`, `last` and `first` to consist of unparsed data. If we now come across a triple of the from `<P person {<X Y {<Z first john >}>}>` we can infer that the variable `Y` can be fixed to `name`, because this is the only label that is allowed to occur between `person` and `first`.

These kinds of dependencies can be used to complete and simplify queries as and also to test equivalence as shown in the succeeding example.

Example

We now illustrate the four steps of the re-writing process sketched above using an example. The example assumes a database of people that are described in terms of some properties including their affiliation to a certain university. The view we consider as the basis for re-writing separates the properties of persons from the values of these properties by introducing two new tags `property` and `value`.

```
<view(P') person {<prop(X') property Y'> <val(X') value Z'>}> :-
    <P' person {<X' Y' Z'>}>
```

The query, we will rewrite asks for people affiliated to stanford university and their properties. The query body is defined on the basis of the general database.

```
<f(P) stan_student { <X Y Z> }> :-
    <P person {<U university stanford> }>@db AND
    <p person {<X Y Z>}>@db}
```

In order to rewrite this query into a form that uses the view defined above, we first determine a mapping from variables in the view to variables and terms in the query body. For the first conjunct of the query body the corresponding mapping is the following:

```
[P' \mapsto P, X' \mapsto U, Y' \mapsto university, Z' \mapsto stanford]
```

For the second conjunct, we get a slightly different mapping:

```
[P' \mapsto P, X' \mapsto X, Y' \mapsto Y, Z' \mapsto Z]
```

As the conjuncts require different mappings, we replace each of them independently by the modified head of the view definition receiving a rewriting of the original query:

```
<f(P) stan_student {<X Y Z>}> :-
    <view(P) person
        {<prop(X) property Y> <val(X) value Z>}>@View AND
    <view(P) person
        {<prop(U) property university> <val(U) value stanford>}>@View
```

In order to verify the rewriting we now compute the composition of the rewritten query with the view used for rewriting by unfolding the conjuncts in the query body:

```
<f(P) stan_student {<X Y Z>}> :-
    <P person {<X Y Z'>}>@db AND
    <P person {<X Y' Z>}>@db AND
    <P person {<U university Z''>}>@db AND
    <P person {<U Y'' stanford>}>@db
```

Using the dependencies imposed by the object ids, we can infer that Y $\equiv$ Y', Z $\equiv$ Z'. This enables us to remove one of the first, now equivalent conjuncts. Using the same argument we can remove one of the last two conjuncts as Z" $\equiv$ `stanford` and Y" $\equiv$ `university`. The resulting query is obviously equivalent to the original one, thus verifying the rewriting.

6.4.4 Object-Oriented Systems

A dominant characteristic of semantic web applications is the use of structured knowledge about the domain of discourse in terms of conceptual models. If the consider queries in the context of the semantic web, we also have to think about querying these conceptual models. With respect to query re-writing existing work that us closest to the task of re-writing conceptual queries is work on query re-formulation in object-oriented database systems. We therefore briefly describe the re-writing of queries in the Object Query Language OQL.

Object Data Model and Query Language

The Object Data Management Group OMDG defines a standard data model for object-oriented data. The model is based on atomic data types such as integer, Boolean or string. These atomic types can be used to recursively define structured data types used type constructors such as set, bag, list or tuple. These structured objects form the data model that can be accessed via object interfaces that describes structured objects as classes with attributes and relations to other classes as well as a limited set of integrity constraints. These classes are arranged in a subclass hierarchy. Subclasses inherit all attributes and relations from their super-classes. Beside these interfaces, the model considers typed variables that can be used in queries.

The Object Query Language OQL has been defined as a means for querying the object oriented model sketched above. The Query language has an SQL like structure with queries of the form:

```
select expression*
from (variable in expression)*
where expression
```

An expression in OQL can be a constant, a variable from a database interface, Further, it is possible to refer to attributes of a structured object, to call methods attached to objects and to use predefined functions. Expressions can be existentially and universally quantified. Further, expressions can be composed to to form complex type description using the constructors mentioned above.

Semantic Rewriting

The process of re-writing OQL queries is based on semantic knowledge about the relation between different OQL expressions. There are two sources of this semantic knowledge. First of all, it is assumed that there are mapping rules $Q \Rightarrow Q'$ between different interfaces involved that state how certain OQL expressions have to be re-formulated in order to match a different interface. A second source of semantic knowledge is provided by consistency constraints that state conditions that must always hold at the same time using OQL expressions. A consistency constraint consists of two OQL expressions Q_1 and Q_2 and a set of variables for which the constraint must hold. As such a constraint states the equivalence of the two OQL expressions, it gives rise to two additional re-writing rules, namely $Q_1 \Rightarrow Q_2$ and $Q_2 \Rightarrow Q_1$.

In order to be able to make use of the semantic re-writing rules, we have to be able to identify parts of a query that matches the body of a re-writing rule. The high expressiveness of OQL makes it almost impossible to perform this matching on a semantic basis. However, it is possible to define a syntactic matching algorithm. The use of a syntactic matching algorithms makes it necessary to resolve syntactic heterogeneity between equivalent OQL expression that arise from the flexibility of the OQL grammar. This problem is addressed by converting all OQL expressions into a standard normal form. Once the query to be re-written and all re-writing rules have been converted into the normal form, a matching algorithm can be used to identify parts of the query that can be replaced by expressions over the target interface(s). This process is applied recursively until no rules apply any more. Due to the syntactic nature of the matching, it cannot be guaranteed that all equivalent re-writings are found, but as the following example illustrates it is often possible to find a suitable re-writing.

An Example

We consider an example from the area of information integration where information about different kinds of products have to be accessed. In the example scenario, we have to information sources one containing information about computer monitors, the other about hard-disks. Beside a product code monitors are described in terms of their size and their manufacturer, hard-disks in terms of capacity, disk-group (type) and manufacturer. Both sources contain information about compatibility with other products. The corresponding interfaces are shown below.

```
class Monitor {
    attribute integer code;
    attribute integer dimension;
    attribute string manufacturer;
    attribute set(string) compatible;
    }

class Disk {
    attribute integer code;
    attribute integer capacity;
    attribute string manufacturer;
    attribute string disk-group;
    attribute set(string) compatible;
    }
```

The task is now to access the information in the two source via a global interface that talks about products in general adding information about a product type, the manufacturer, compatibility with other products and a general description. Again, the corresponding interface is given below:

```
class Product {
    attribute integer code;
    attribute string descriptor;
    attribute string manufacturer;
    attribute string type;
    relationship set(Product) compatible: inverse compatible;
    }
```

The query we consider in this example asks for a hard disk by the manufacturer HP that is compatible with some 19inch monitor. The corresponding OQL expressions, phrased against the general product interface is the following:

```
select [code:=x.code, descriptor:=x.descriptor] from x in products
where x.type="disks" and
      x.manufacturer like "HP" and
      (exists y in x.compatible : y.type="monitor" and
                                  y.descriptor like "19 inch")
```

In order to re-write this query in such a way that wit can be executed on the two specialized interfaces we need some semantic information in terms of re-writing rules that connect the general interface with the local information sources. Below, we give an example of such a rewriting rule that matches general queries for hard-disks and generates the corresponding OQL query over the interface of the Disks database. We assume that a similar rule exists for Monitors.

```
select [code:=x.code,
        descriptor:=x.descriptor,
        manufacturer:=x.manufacturer,
        compatible:=x.compatible]
from x in products
where x.type = "disks"

=>

select[code:=y.code,
       descriptor:=y.capacity,
       manufacturer:=y.manufacturer,
       compatible:= (select z
            from z in products
            where exists t in monitors :
                (t.code=z.code and
                 t.manufacturer in y.compatible))]
from y in disks
```

Another kind of semantic information we can use is the fact that the compatibility relation is reflexive. This fact is covered by the following consistency constraint.

```
for all x:Monitor and y:Disk
    y.disk-group in x.compatible ~ x.manufacturer in y.compatible
```

Using the mapping rules to the local databases an the consistency constraint (note that is can be applied to rewrite the last line in the resulting query of the mapping rule) we can re-write our example query in several steps that we omit due to lack of space. The result of the re-writing process after normalization is the following query that can now directly be executed on the local databases.

```
select [code:=a.code, descriptor:=a.capacity]
from a in disks, z in monitors
where a.manufacturer like "HP" and
      z.dimension like "19 inch" and
      a.disk-group in z.compatible
```

6.4.5 Ontology-Based Systems

Another extension of query re-writing towards more expressive queries is the development of approaches that allow the predicate names in a conjunctive query to be concepts and relations from an ontology. The inheritance relations between these concepts that might be explicitly stated or implied by the ontology make query answering more difficult. Most work done in thsio area is based on Description Logics as a basis for representing andreasoning about the ontology. In the following, we first describe general approaches for answering queries over Aboxes, we then discuss conjunctive query rewriting for the case where the predicates in the query body are taken from a TBox. Finally, we review some approaches that employ approximation techniques in order to compute query re-writings.

Querying Assertional Knowledge

Queries over assertional knowledge over a terminology $\mathcal{T}$ with concept names $c \in C$ and role names $r \in R$ can be phrased as usions of conjunctive queries, where the conjunctions are either unary predicates that correspond to concept or binary predicates that correspond to role names. The general form of a such a query is the following:

$$q(\bar{X}) :- \bigvee p_1 \wedge \cdots \wedge p_n, \text{where } p_i \in \{c_j(X) | \, c_j \in C\} \cup \{r_k(X,Y) | \, r_k \in R\}$$

Answering these queries has to involve dedcutive reasoning, as the definition of concepts and queries allows for the use of incomplete knowledge in terms of disjunctions or existentially quantified variables. A common approach to perform this deductive reasoning is to translate the query into a concept expression in a description logic. Unary predicates are translated to concept names, binary predicates into existential restrictions on the corresponding role. The query $c(X) \wedge r_1(X,Y) \wedge r_2(Y,Z) \wedge d(Z)$ is rolled up into the concept expression $(C \sqcap (\exists r_1.(\exists r_2.D)))$. Thereby the construction of the concept expression is guided by the dependency between the variables involved. These dependencies can be represented by a dependency graph. It has been shown that only queries whose dependencies form a directed acyclic graph can be answered. As the rolling-up of the query makes extensive use of existential quantification. Therefore the resulting conceot expression will be in a logic with existential quatifiers.

After rolling up the query there are two principled ways to proceed. The first one is to apply well kwon inference procedures in order to retrieve all instances of the query concept from the given Abox. This approach is straightforward, but it also suffers from the high complexity of the retrieval task. However, it has been shown that the resulting concepts can be approximated by concept without existential quantifiers, thus allowing for subsumption testing in polynomial time. A second approach to allow more efficient query answering is to re-write the query concept to a maximally contained query that only contains a predefiened set of concept and role names that are directly used in the ABox. The re-written query can then directly be stated against a database representation of the assertional knowledge.

Query Re-writing

Rewriting queries over terminologies is even harder a problem than answering queries. The possibility of obtaining maximally contained re-writings depend on both, the nature of the query and view definitions as well as the expressiveness of the logic used to define the terminology.

The first case that can be distinguished is the one where query and views do not contain existentially quantified variables. This is the case when they only contain unary predicates that correspond to concept names in the terminology. Assuming that the query is of the form $q(X) : -c(X)$ and the views of the are form $v_i(X) : -c_i(X)$ then $q'(X) : -v_{i_1}(X) \wedge \cdots v_{i_l}(X)$ a re-writing of q if $C \sqsubseteq c_{i_1} \sqcap \cdots \sqcap c_{i_l}$ holds. In the case of a global-as-view approach, this result can easily be extended to the case where q is an arbitrary Boolean expression over unary predicates that correspond to concept names. In this case, the query is first translated into negation normal form (negation only applies to single predicates). Then the predicates in the query are re-written individually without changing the structure of the query. Non-negated predicates are re-written in the way described above: the predicate is replaces by the conjunction of maximally contained views. In the case of a negated predicate $d(x)$ the rewriting is the disjunction $v_{i_1} \vee \cdots \vee v_{i_k}$ such that $c_{i_1} \sqcup \cdots \sqcup c_{i_k} \sqsubseteq d$. Due to the fact that for this first case, query rewriting can be reduced to subsumption reasoning, the expressiveness of the terminology is only bounded by the availability of sound and complete methods for determining subsumption.

In the case where we have existentially quantified variables in query and view definition, the re-writing problem is more difficult to solve. In this case it may happen that there is no bound on the size of the rewriting because we cannot guarantee that the variables in a view are bound to different variables. As a consequence, the maximally contained re-writing can be a *recursive* Datalog program. Obviously, subsumption testing is not applicable in this case. This problem can be avoided if we can guarantee that the data model does not contain cycles (tree model assumption). In this case a expansion algorithms can be employed to determine a set of facts the rewriting has to be constructed from. Possible re-writings are then determined by constructing and checking all possible combinations of these facts. This process, however, can only be performed for rather inexpressive logics.

For the related problem of answering queries on the basis of pre-defined views, it has been shown on a theoretical basis that re-writings can also be computed for a description logic supporting n-ary relations. In this case the re-writing problem is reduced to a satisfiability problem by constructing a single concept expressions containing the query, the view definitions and a possible a possible answer. It can be shown that the answer is a correct answer to the question with respect to the views if the constructed concept expression is satisfiable. However, there is no tractable algorithm for computing the set of all answers.

Examples

We illustrate the abilities of query rewritring over terminologies using two examples. In both cases the goal is to rephrase the query only using a predicates from a predefined set of concept or role names. The first one follows the global as view approach, i.e. the views define concepts used in the query in terms of complex concept expressions. These expressions are not necessarily part of the allowed vocabulary. In the second case, a local as view approach is used, that defines some of the predicates to be used in terms of complex concept expressions.

Global-as-View We start with a simple ontology that discriminates animals into domestic, foreign and production animals and contains some kinds of animals that fall under one or more of these categories. Now consider a second ontology that describes classes of animals in the way a child would possibly categorize them. The main distinctions made in this ontology are pets, farm animals and zoo animals. These distinction are based on the experience of a child that some animals are kept at home, at farms or in zoos. While both ontologies do not share any class except for the general class animal, it should be possible to establish a relation between the two. Using common world knowledge and the informal descriptions of the classes in the ontology we can conclude that pets should be a subclass of domestic animal and include Cats and Dogs. Farm animals should be a subclass of domestic animals and include Cows and Pigs. Finally zoo animals should be subsumed by Foreign animals and contain all the subclasses of foreign animal.

Assume that we want to post a query that is formulated using terms from the second ontology to an information source that has been classified according to the first ontology. In the case of a complex query, it first has to be transformed into negation normal form:

$$\begin{aligned} q(X) &:- \quad zoo-animal(X) \wedge (\neg(pet(X) \wedge farm-animal(X)) \\ q_{nnf}(X) &:- \quad zoo-animal(X) \wedge (\neg pet(X) \vee \neg farm-animal(X)) \end{aligned}$$

A replacement of the predicates by their upper and lower bounds repectively leads to the following complex query:

$$\begin{aligned} q'(X) \quad :- \quad & foreign - animal(X) \wedge (negdomestic - animal(X) \vee \\ & \neg(production - animal(X) \vee domestic - animal(X))) \end{aligned}$$

Applying simple equivalence rules, we see that the transformed query is equivalent to the following simple query, that is the wanetd re-writing of the query using terms from the second ontology:

$$q'_{norm}(X) : -foreign - animal(X)$$

Local As View We now consider query rewriting according to the local-as-view strategy. We allow queries with existential quatifiers, however, queries are restricted to union of conjunctive queries. The example query to be rewritten defines flights that either only stop in american cities or are operated by an american airline

$$\begin{aligned} q(X) \quad &:- \quad flight(X) \wedge \forall stop.AmCity(X) \\ q(X) \quad &:- \quad flight(X) \wedge airline(X, Y) \wedge AmComp(Y) \end{aligned} \tag{6.1}$$

The predicates allowed for re-writing are the role *stop* and the three concept predicates defined below:

$$\begin{aligned} flight1(X) \quad &\equiv \quad flight(X) \wedge (\leq 1stop) \\ flight2(X) \quad &\equiv \quad flight(X) \wedge (\geq 1airline) \wedge \forall airline.AmComp \\ eastCity(X) \quad &\equiv \quad AmCity(X) \wedge \forall located.EastCoast \end{aligned}$$

Using these definitions, we get two conatined re-writings of the original query displayed below:

$$\begin{aligned} q'(X) \quad &:- \quad flight1(X) \wedge stop(X, Y) \wedge eastCity(Y) \\ q'(X) \quad &:- \quad flight2(X) \end{aligned}$$

The first conjunction is subsumed by the original query, because it only has one stop and this one is known to be an american city, because eastCity is subsumed by american cities. The second is subsumed by the query, because it only has one operating airline which is american.

6.4.6 Implications for SWAP

Query re-writing is an important technique with respect to the aims of the SWAP project. Information between peers is exchanged by posing and answering queries. In a peer to peer system, we cannot expect that all peers use the same conceptual or logical data model. Results from the database area have shown that this problem can be solved by query rewriting, provided that certain mappings between the models exist. As SWAP looks at peer-to-peer system from a semantic web perspective, query rewriting for semi-structured information and object-centered representations are of special importance. In order to employ the approaches discussed above in SWAP, we have to adapt them to semantic web technology. In especially, we have to check if rewriting for TQL can be applied to re-write queries over RDF and RDF schema models. Further, we have to investigate how query rewriting for the Object Query Language can be applied to the query ontology query languages such as DQL that are currently being developed.

6.5 Multi language queries

A peer-to-peer system can comprise all kind of users. In some cases they use a common language in others the members of a certain peergroup share a common interest but not a common language. To enable the communication between peers using different languages it is necessary to translate automatically between these languages. The translation task is twofold:

- Query translation
  The translation of the query terms into the language of a certain peer

- Answer translation
  The translation of the answers to a query into the language of the querying peer

Within our scenarios the focus lies on the translation between different European languages. Furthermore we assume that it is within our context more important to translate the query than the answer, because the participants know more or less the languages of each other and answer translation is a particular hard task.

In this section we discuss query translation methods suggested from the information retrieval community and the semantic web (ontology using) community. Those communities have already some experience with the translation task.

6.5.1 Methods from information retrieval

[Bra02] describes recent developments in the area of multi lingual document retrieval. The information retrieval community uses an long established way to evaluate the quality of retrieval results. Therefor different retrieval approaches can be compared easily using the values for recall and precision. Two combinable solutions are generally used to resolve translation ambiguities. A corpus a used to construct a statistical model. This model can be applied to translations from machine-readable dictionaries or machine translations. The translation methods with the best results are presented in more detail.

[Sav02] uses a sequential approach to translate a query. They translate the entire query with the Bablefish system [GLY98] and then translate each word in the query with BABYLON [2] a bilingual translation system. The words from the two translations are than used to search the corpus. With this combined approach they get almost the same answer quality as with manual translated queries.
Others use statistical translation models. Those can differ in the way to construct the statistical information.

- [KKH00] uses random indexing. Random indexing is based on the idea, that (given a training corpus) words with the same meaning occur close to each other in the training corpus. Using this statistical information a dictionary is constructed and than used to translate the queries.
- [NS02b] uses Web pages as a training corpus. Web pages published from the same organization in different languages can be used to build a statistical translation model for a given context. This copes with the problematic of evolving languages and the use of domain specific vocabulary. The parallel corpora can be cleaned using stop word list and stemming algorithms.
- [BF02] uses hidden Marcov chains [Rab90] combined with a dictionary and a training corpus. The translations from the dictionary are weighted by the information from the HMC model which is constructed from the training corpus.

6.5.2 Methods from the Semantic Web

Much research on ontology translation has used ad-hoc mapping rules between ontologies. An extract from a recent survey [WVV$^+$01] on integration of information is included here. The translation approaches cited here comprise translations between different machine languages as well as natural languages. However the translation between natural languages is seen as a subproblem of the mapping problem between different labels.
The problem of mapping different ontologies is a well known problem in knowledge engineering. We will not try to review all research being conducted in this area. We rather discuss general approaches that are used in information integration systems.

[2]online available at www.babylon.com

Defined Mappings A common approach to the ontology mapping problem is to provide the possibility to define mappings. This approach is taken in KRAFT [PHG$^+$99], where translations between different ontologies are done by special mediator agents that can be customized to translate between different ontologies and even different languages. Different kinds of mappings are distinguished in this approach starting from simple one-to-one mappings between classes and values up to mappings between compound expressions. This approach allows a great flexibility, but it fails to ensure a preservation of semantics: the user is free to define arbitrary mappings even if they do not make sense or even produce conflicts.

Lexical Relations An attempt to provide at least intuitive semantics for mappings between concepts in different ontologies is made in the OBSERVER system [MKSI96]. The approaches extend a common description logic model by quantified inter-ontology relationships borrowed from linguistics. In OBSERVER, relationships used are *synonym, hypernym, hyponym, overlap, covering* and *disjoint*. While these relations are similar to constructs used in description logics they do not have a formal semantics. Consequently, the subsumption algorithm is rather heuristic than formally grounded.

Top-Level Grounding In order to avoid a loss of semantics, one has to stay inside the formal representation language when defining mappings between different ontologies (e.g. DWQ [CGL01]). A straightforward way to stay inside the formalism is to relate all ontologies used to a single top-level ontology. This can be done by inheriting concepts from a common top-level ontology. This approach can be used to resolve conflicts and ambiguities (compare [HH00]). While this approach allows to establish connections between concepts from different ontologies in terms of common superclasses, it does not establish a direct correspondence. This might lead to problems when exact matches are required.

Semantic Correspondences An approach that tries to overcome the ambiguity that arises from an indirect mapping of concepts via a top-level grounding is the attempt to identify well-founded semantic correspondences between concepts from different ontologies. In order to avoid arbitrary mappings between concepts, these approaches have to rely on a common vocabulary for defining concepts across different ontologies. [Wac99] uses semantic labels in order to compute correspondences between database fields. [SvF$^+$00] build a description logic model of terms from different information sources and shows that subsumption reasoning can be used to establish relations between different terminologies. Approaches using formal concept analysis (see above) also fall into this category, because they define concepts on the basis of a common vocabulary to compute a common concept lattice. [iAHK02] proposes an approximate ontology translation framework. They provide semantics for mapping rules and which can be checked for consistency easily. The query evaluation is based on [CCHM01].

Chapter 7

Visualization

7.1 Idea and Goal

Visual techniques harness human perceptual capabilities to detect patterns and outliers in visual information. Finding adequate visualisations is not a trivial task, given the state of the art of the information visualisation field. Current visualisation techniques were all developed as solutions for certain problems. Therefore visualisation techniques are often re-adapted to suit new problems, rather then being used off-shelf.

To decide about visualisations for SWAP, the following issues are important:

- *What to visualise?* This determines the aspect to focus on. The novelty of the SWAP project consists in combining knowledge structures and P2P networks in a way that they benefit from each other. Several aspects are of interest:
  - Knowledge - knowledge related visualisations provide insight in the main knowledge structures, the ontologies that are available on each peer.
  - Network - network visualisations present the topology, structure, functionality and traffic flow through the network.
  - Community - the SWAP system is a new virtual media through which people interact. Visualising community related issues (intensity of communication, knowledge communities) gives an insight in the social impacts of the system.
- *Why to visualise?* Each visualisation has a well-defined goal: it aims to answer a question or to support a task. It is the task/question that determines the visualisation. It is therefore important to clearly define the questions/tasks that need support for each aspect.
- *How to visualise?* When the goal of the visualisation is clear one needs to select and to adapt one of the existing techniques. In some cases the choice for a certain

method/tool is straightforward, but often one has to chose, adapt or invent techniques that match a certain goal.

Each of the three following sections is devoted to one of the three aspects: knowledge, network, community. For each aspect we describe a (preliminary) set of questions/tasks that can be resolved by visualisations. Also we present tools explicitly developed for certain tasks, if any (ex. for visualising network topologies). The last section concludes with an overview of exiting tools which can be potentially used after being adapted to our needs.

7.2 Knowledge

The knowledge component in SWAP consists in the ontologies of the peers. Both the users of the SWAP system, as well as the developers need to perform certain ontology related tasks which can be supported by visualisation. We define three main task categories:

- **Analyzes** Simply visualising the ontology provides an overview of the vocabulary of the ontology and insight in the way the data set is partitioned by the ontology. The following analyzes tasks are possible:
  - *One data set, many ontologies.* Visualising how a given data set is partitioned by different ontologies provides a different view of the same data set. Suppose a user of a peer wants to see his data set through the ontology of another peer, and then decide if he wants to change his own ontology. Consider the case of people working with the same documents, but having different expertises and therefore classifying documents along different dimensions. For example the experts of Dresdner Bank.
  - *Many data sets, one ontology.* Visualising different data sets according to the same ontology allows comparison of the data sets. While this feature is still available, in the SWAP system it has a different applicability: for example when a peer wants to see the data on another peer (or just another data set), through his own ontology.
  - *Ontology monitoring.* A benefit of combining knowledge structures with P2P networks is the fact that knowledge can evolve over time adapting itself to the changing needs of the user and newly available knowledge. Monitoring how the ontology/knowledge evolves over time can simply be done by taking snapshots of the knowledge at different points in time. Transitions between the different stages of the monitoring have to be intuitive so that the user can understand the changes.
- **Query** The SWAP system takes advantage of the knowledge of the whole network in order to answer the queries of users. Visualisation techniques can solve major problems related to queries:

  - *Query formulation* - Query interfaces that contain a graphical presentation of the available knowledge (ontology), make the query formulation task much easier as the user is already acquainted with the used terminology.
  - *Query Result presentation* - inadequate presentation of query answers shadows the performance of many search engines as the user is overwhelmed with results without being able to pick the part that is of interest for him. Graphical presentations can explain the relation of the answer to the original terms of the query. The left part of Figure 7.1 shows a graphical presentation of the answer set for a query that wants to retrieve all vacation destinations on the French Loire, for 4 people, with 2 rooms and having the third quality class. The map was produced using the Cluster Map visualisation component.
  - *Query reformulation* - Very often the user is not satisfied by the returned answer: there are either too many items, or no items at all. Graphical presentation of results allows a user to perform:
    - query relaxation: in case of empty sets, by giving up some of the search terms and choosing the result items that come closer to the original query. In the case of our example query, as shown in the left part of Figure 7.1, there are no vacation destinations matching the terms of the queries. However, if the user ”relaxes” his requirements by giving up either the ”Loire” or the ”3 stars” query term, there are 2 items that could be of interest.
    - query broadening: in the case of an empty answer set some terms can be replaced by their super-classes broadening the scope of the query. Considering that the user is not willing to give up any of his requirements, however he accepts to go to a place different from Loire, but still in France, he can broaden his search. As shown in the right part of Figure 7.1, query broadening results in exactly one destination in France.
    - query narrowing: if there are too many items, the query can be repeated with more specialised sub-classes of certain query terms, narrowing the scope of the query.

- **Navigation** Visual presentations of ontologies can be employed as navigational structures for accessing data collections organised according to an ontology.

We conclude that knowledge structure visualisation techniques aid several user tasks, such as analysis, query and navigation. For SWAP, such methods would be interesting for

- the users of the system: enhancing the GUI of a peer;
- the developers of the system: embedded in a network monitoring tool.

In what regards tooling, the Cluster Map visualisation component has been applied and fine-tuned for most of the mentioned tasks [FSvH02].

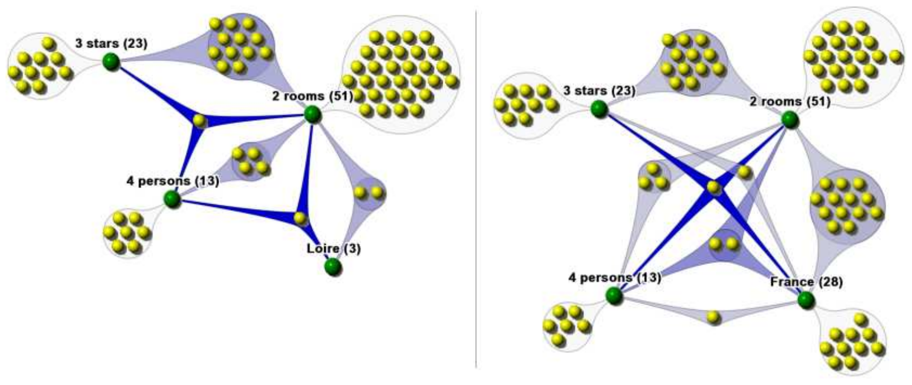


Figure 7.1: Query relaxation(left) and broadening(right)

7.3 Network

Visualisation has been successfully applied in network presentation and was widely used in activities related to the Internet and the Web. Visualisation aids our understanding of the structure, organisation, functioning and traffic of the network.

7.3.1 Structure

Visualising the peers in the network, as well as the interconnections between them gives a global overview of the structure of the network. There is a link between two peers if they know about each other. Such a map provides answers to several questions: How many nodes exist? Which nodes does a particular node know about? Which nodes are currently active (on-line)? Monitoring the network structure over time shows the modifications in size (How many peers drop out? How many new-comers are?) and connectivity (Do links increase in number? How much? Or is their number constant? Why?).

There are two approaches to visualising a network structure:

- *A topological approach.* The infrastructure is mapped onto geographic space. This is to be considered when the physical location of the peers is important, so that facts of the form ”most peers are in Germany” can be deduced. Figure 7.4 presents a network mapped in the European geographic space.
- *An abstract approach.* The structure is drawn in an abstract space, without taking into account the geographic position of the nodes. In this cases the geographic location is not important, and the visualisations are meant to reveal other features of the network (connectivity, routing) which are location independent but would be difficult to see on a cluttered physical view.

Structure visualisation is useful for monitoring the complexity and evolution of the network. It can enhance the GUI of a peer by presenting the user with the local view of the network. Currently most P2P system interfaces do not allow the user to have a view on the neighboring peers. However it could be useful for a user to see which other users are online at a given moment in time: if he knows some peers (friends), he could directly instruct the routing of his queries. Gnucleus is an initiative to visualise the neighbouring nodes of a Gnutella client. A snapshot of a Gnutella network, created with Gnucleus (http://www.gnucleus.net/), is attached.

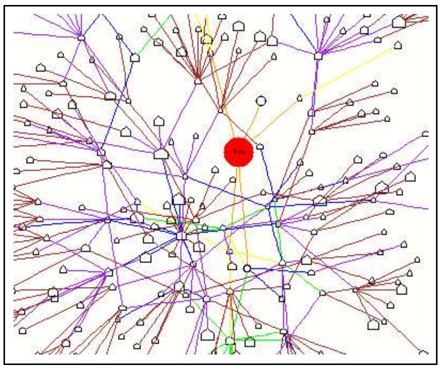
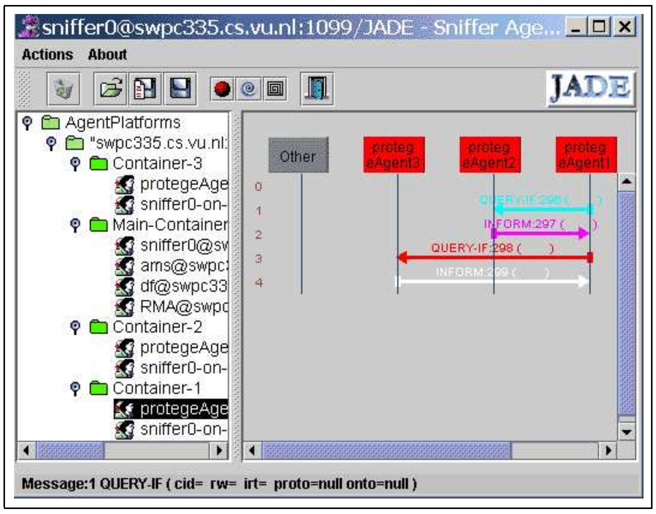


Figure 7.2: Visualising the neighborhood of a Gnutella client using Gnucleus.

7.3.2 Message routing/Operation

One of the motivations of SWAP is to enhance message routing in P2P networks by using the existing knowledge in peer selection. Visualising how a message is routed through the network helps understanding and modifying the message routing protocol. For example in the Jade platform offers visualisation of interaction between agents, Figure 7.3. This allows one to see how the message is routed, where it is answered and what is the answer.



Message:1 QUERY-IF (cid= rw= irt= proto=null onto=null)

Figure 7.3: Message routing in Jade.

The network developer gets an insight in the operation of the network and can fine-tune routing protocols if needed.

7.3.3 Traffic/Data Flow

and the volume of exchanged traffic [1].

Flooding is a frequent phenomena in P2P networks when the data flow cannot be controlled. Routing protocols try to solve this problem so that P2P networks can scale better. Visualising traffic intensity gives an insight in the parts of the network that could be potentially problematic. One can see where the most traffic is and then find out why? The map in Figure 7.4, depicts the volume of tele-communication between european countries. The map informs about the parties involved in the network, their geographic location

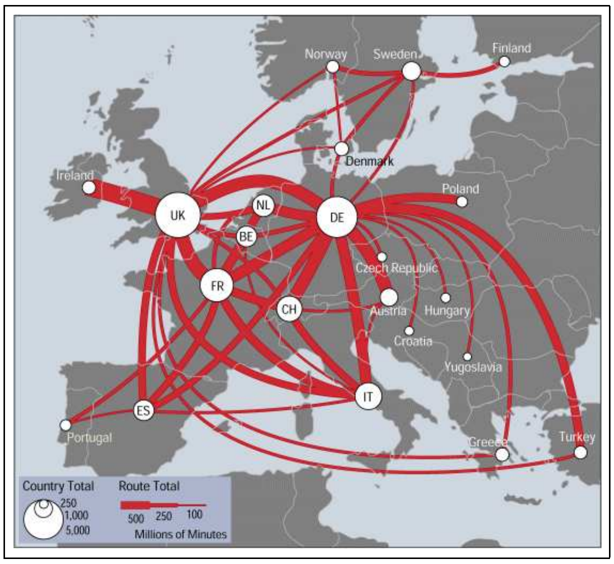


Figure 7.4: Traffic flow on a topological structure map.

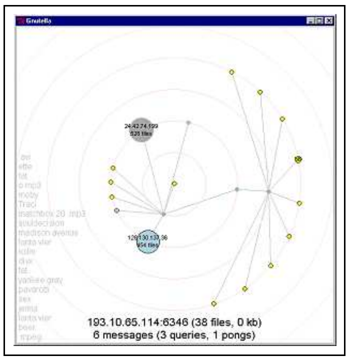

Gnutella
193.10.65.114:6346 (38 files, 0 kb)
6 messages (3 queries, 1 pongs)

Figure 7.5: Gnutella in action.

While static visualisations of the traffic volume are useful, they fail to give insight in the actual flow of data. This goal can be accomplished by using real time visualisation of the network traffic. A pioneer work to visualise interactions in P2P network is done in the GnuTellaVision initiative. This visualisation displays the operation of the Gnutella network in real time. The nodes of the network are presented on a circular layout. Their size indicates the volume of data they posses, while their color suggests their state: off or on-line. It is also possible to see the posted queries, which are present on the left side of the screen.

A demo is available for download at: http://www.sims.berkeley.edu/ ping/gtv/gtv.avi.

7.3.4 Knowledge distribution

The previous visualisations did not take into account the existing knowledge. Because one of the key motivations for SWAP is to see how knowledge evolves in a distributed environment, visualisation can help to answer the following questions:

- Who knows what - Giving a color code to each topic, one can adapt a simple network visualisation by coloring each peer according to his expertise. This will show who knows what and will give an insight in the distribution of topics in the network.

[1] http://www.telegeography.com

- Evolution of knowledge distribution - through interactions peers learn from other peers, therefore the knowledge distribution changes over time. Monitoring these changes will give us insight in the topic evolution: will more peers learn about the main topics, will some small topics disappear over time?
- Who knows more? - we can adapt a structure map, showing peers that have a larger ontology(or contain more items) as having a larger shape.
- Content communities - if the length of the links between peers is inversely proportional with the amount of interchanged messages then clusters of peers with closely related interests will form, revealing the emerging content communities.

7.4 Community

The success of SWAP depends on the degree the users will be satisfied with it. Often technically brilliant solutions fail due to low usability. Monitoring user behavior is useful for two reasons:

- *Increase the usability of the system.* Visualisations can reveal existing problems and suggest solutions. Many questions can be answered:
  - Do participants get involved? What is the frequency/amount of posed questions.
  - Is there a lot of interaction? Do people reply?
  - Do newcomers integrate easily? If they know very few peers at the start, can the system still provide answers in a reasonable time-frame?
  - Who are the experts? Who has been active for a long time? Who answers the most queries?
- *Test social metaphors.* Many methods underlying SWAP are based on social metaphors for communication. However, in the new environment participants are not conditioned any more by factors such as geographic locality, age or race. The SWAP system will establish a new on-line interactive environment where the identity of the participants is defined in terms of 1) knowledge and 2) interactions. It is interesting to see if the behavior of the peers is as expected.

There a lot of work in the field of social maps which are based on human behavior in electronic media such as email, chat, mailing lists or MUDs. A remarkable initiative is People Garden, a visualisation that presents portraits of user behavior in mailing lists using the flower paradigm. Each individual is represented by a flower, and each petal of the flower corresponds to a message. The colors of the petals shade with time. A response to a message is indicated by placing a small yellow circle at the top of the petal. This visualisation proves powerful when visualising whole communities, as gardens of flowers. The height of the flower is proportional with the time spent in the garden. Such gardens give an insight in the characteristic of the group. Figure 7.6 shows a lively, democratic group, where most participants post messages, no matter if they are present for a long time, or if they have recently joined. Such visualisations for SWAP would allow estimating the social climate in the system.

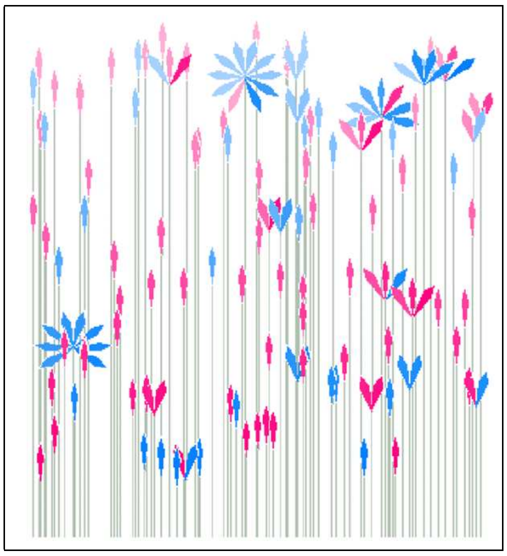


Figure 7.6: People Garden: a democratic group.

7.5 Tools

In the previous chapters we have mentioned some tools that target certain type of issues. Further, we give a list of other tools that could be adapted/re-used for our purposes.

- Inxigh's Star Tree uses a hyperbolic browser interface to view Star Trees: shared interactive maps that help users navigate information.
- Hyperprof is a hyperbolic browser that allows to view a Java profile in a hyperbolic view.
- H3Viewer is a tree viewer that allows an intuitive exploration of hierarchical graphs. It can be also viewed in 3D.
- Ontobroker had a simple hyperbolic browser included.
- OntoRama is a Java client that browses a knowledge base (ontology) structure in a hyperbolic layout. It especially allows browsing of RDF.[EGR02]

Final remark. This study proves that visualisation is a powerful tool to give insight in different aspects related to SWAP such as knowledge, network and community. Also, there is significant work in all three fields. However existing methods will have to be chosen and adapted in such a way that they serve the specific needs of SWAP users and developers.

Chapter 8

Summary

The SWAP project has a wide scope. This deliverable provided a summary of methods related to the goals of our project i.e. methodologies to deal with questions in the field of peer-to-peer and in the field of ontology processing. Having the results of this survey in mind we are ready for the next step of designing and implementing our own methods for the SWAP project. Even though in some of the areas a lot of research has been done already, we still have enough freedom to continue with new ideas.

Bibliography

[AS01] Rakesh Agrawal and Ramakrishnan Srikant. On integrating catalogs. In *World Wide Web*, pages 603–612, 2001.

[BCW01] Christopher Brewster, Fabio Ciravegna, and Yorick Wilks. Knowledge acquisition for knowledge management: Position paper, 2001.

[BF02] N. Bertoldi and M. Federico. Itc-irst at clef 2001: Monolingual and bilingual tracks. In Carol Peters, Martin Braschler, Julio Gonzalo, and Michael Kluck, editors, *CLEF*, volume 2406 of *Lecture Notes in Computer Science*, pages 9–26. Springer, 2002.

[BKM00] F. Baader, R. Küsters, and R. Molitor. Rewriting concepts using terminologies. In A. G. Cohn, F. Giunchiglia, and B. Selman, editors, *Proceedings of the Seventh International Conference on Knowledge Representation and Reasoning (KR200)*, pages 297–308, San Francisco, CA, 2000. Morgan Kaufmann Publishers.

[BLR97] Catriel Beeri, Alon Y. Levy, and Marie-Christine Rousset. Rewriting queries using views in description logics. In *roc. of the 16th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-97)*, pages 99–108, 1997.

[Bra02] Martin Braschler. Clef 2001 - overview of results. In Carol Peters, Martin Braschler, Julio Gonzalo, and Michael Kluck, editors, *CLEF*, volume 2406 of *Lecture Notes in Computer Science*, pages 9–26. Springer, 2002.

[BW01] Sherif Botros and Steve Waterhouse. Search in jxta and other distributed networks. In *Proceedings to The First International Conference on Peer-to-Peer Computing (P2P'01)*, August 2001.

[Car02] Jeremy J. Carroll. Matching rdf graphs. Technical report, Hewlett-Packard Laboratories Bristol, UK, 2002.

[CCHM01] K. C.-C.Chang and H.Garcia-Molina. Approximate query mapping:accounting for translation closeness. *The VLDB Journal*, 10(2-3):155–181, September 2001.

[CGL00] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Answering queries using views over description logics knowledge bases. In *Proceedings of the National Conference on Artificial Intelligence AAAI/IAAI 2000*, pages 399–404, 2000.

[CGL01] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Description logics for information integration. In *Computational Logic: From Logic Programming into the Future (In honour of Bob Kowalski)*, Lecture Notes in Computer Science. Springer-Verlag, 2001.

[Cod02] Coderman. Decentralized discovery, 2002.

[CW02] Antonio Carzaniga and Alexander L. Wolf. Content-based networking: A new communication infrastructure, 2002.

[DMDH02] AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. Learning to map between ontologies on the semantic web. In *Proceedings to the Eleventh International World Wide Web Conference*, Honolulu, Hawaii, USA, May 2002.

[EGR02] Peter Eklund, Steve Green, and Natalyia Roberts. Ontorama: Brwosing rdf ontologies using a hyperbolic browser. Technical report, Distributed Systems Technology Cetre (DSTC), Australia, 2002.

[FFMM94] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an Agent Communication Language. In N. Adam, B. Bhargava, and Y. Yesha, editors, *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, pages 456–463, Gaithersburg, MD, USA, 1994. ACM Press.

[FHvH$^+$01] Dieter Fensel, Ian Horrocks, Frank van Harmelen, Deborah L. McGuinness, and Peter F. Patel-Schneider. Oil: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2), 2001.

[FRV96] Daniela Florescu, Louiqa Raschid, and Patrick Valduriez. A methodology for query reformulation in CIS using semantic knowledge. *International Journal of Cooperative Information Systems*, 5(4):431–468, 1996.

[FSvH02] C. Fluit, M. Sabou, and F. van Harmelen. Ontology-based Information Visualisation. In Vladimir Geroimenko, editor, *Visualising the Semantic Web*. Springer Verlag, 2002.

[GLR00] Francois Goasdoue, Veronique Lattes, and Marie-Christine Rousset. The use of CARIN language and algorithms for information integration: The PICSEL system. *International Journal of Cooperative Information Systems*, 9(4):383–401, 2000.

[GLY98] D.A. Gachot, E. Lange, and J. Yang. The systran nlp browser:an application of machine translation technology. In G. Grefenstette, editor, *Cross-Language Information Retrieval*, pages 105–118. Kluwer, Bosten, 1998.

[GR02] Franois Goasdou and Marie-Christine Rousset. Compilation and approximation of conjunctive queries by concept descriptions. In *roceedings of European Conference on Artificial Intelligence (ECAI'02)*, Lyon, France, 2002.

[Gru93] Tom R. Gruber. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993. Kluwer Academic Publishers.

[Hal00] Alon Y. Halevy. Theory of answering queries using views. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 29(4):40–47, 2000.

[Hal01] Alon Y. Halevy. Answering queries using views - a survey. *The VLDB Journal*, 10(4):270–294, 2001.

[HH00] Jeff Heflin and James Hendler. Semantic interoperability on the web. In *Extreme Markup Languages*, 2000.

[HHH02] Matthew Harren, Joseph M. Hellerstein, and Ryan Huebsch. Complex queries un dht-based peer-to-peer networks. In *Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, Cambridge, MA, USA, March 2002.

[Hon99] Theodore Hong. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, chapter Performance, pages 203–241. O'Reilly, 1999.

[Hor02] Ian Horrocks. DAML+OIL: a reason-able web ontology language. In *Proceedings of EDBT 2002*, March 2002.

[HT00] Ian Horrocks and Sergio Tessaris. A conjunctive query language for description logic aboxes. In *Proceedings of the National Conference on Artificial Intelligence AAAI/IAAI 2000*, 2000.

[HT02] Ian Horrocks and Sergio Tessaris. Querying the semantic web: A formal approach. In Ian Horrocks and James A. Hendler, editors, *International Semantic Web Conference*, volume 2342 of *Lecture Notes in Computer Science*, pages 177–191. Springer, 2002.

[iAHK02] Jun ichi Akahani, Kaoru Hiramatsu, and Kiyoshi Kogure. Coordinating heterogeneous information services based on approximate ontology translation. In *to appear in Autonomous Agents & Multiagent Systems*, http://www.agentcities.org/Challenge02/Proc/Papers/ch02_37_akahani.pdf, 2002.

[KAC$^+$02] Gregory Karvounarakis, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, and Michel Scholl. Rql: A declarative query language for rdf. In *Proceedings of the Eleventh International World-Wide-Web Conference*, Honolulu, Hawaii, USA, 2002.

[KKH00] P. Kanerva, J. Kristofersson, and A. Holst. Random indexing of text samples for latent semantic analysis. In L.R. Gleitman and A.K. Josh, editors, *Proceedings of the 22nd Annual Conference of the Cognitive Science Society*, volume 1036, Erlbaum, New Jersey, 2000.

[KLW95] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42, 1995.

[Lan99] Adam Langley. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, chapter Freenet, pages 123–132. O'Reilly, 1999.

[LG90] D. B. Lenat and R. V. Guha. *Building large knowledge-based systems. Representation and inference in the Cyc project.* Addison-Wesley, Massachusetts, 1990.

[LVC$^+$99] W. Li, Q. Vu, E. Chang, D. Agrawal, Y. Hara, and H. Takano. Powerbookmarks: A system for personalizable web information organization, sharing, and management. In *Proceedings of the Eighth International World-Wide Web Conference*, May 1999.

[Mae02] Alexander Maedche. Emergent semantics for ontologies. *IEEE Intelligent Systems*, January/February 2002.

[MDS02] Frank van Harmelen James Hendler Ian Horrocks Deborah L. McGuinness Peter F. Patel-Schneider Mike Dean, Dan Connolly and Lynn Andrea Stein. Owl web ontology language 1.0 reference. Technical report, W3C Working Draft, July 2002.

[MGJ01] Giovanni Modica, Avigdor Gal, and Hasan M. Jamil. The use of machine-generated ontologies in dynamic information seeking. In *Proceedings of the Ninth International Conference on Cooperative Information Systems (CoopIS 2001)*, Trento, Italy, September 2001.

[MGMR01] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm. Technical report, University of Stanford, 2001.

[Mit97] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[MKSI96] E. Mena, V. Kashyap, A. Sheth, and A. Illarramendi. Observer: An approach for query processing in global information systems based on interoperability between pre-existing ontologies. In *Proceedings 1st IFCIS International Conference on Cooperative Information Systems (CoopIS '96)*, Brussels, 1996.

[MNS01] Alexander Maedche, Gnter Neumann, and Steffen Staab. Bootstrapping an ontology-based information extraction system, 2001.

[NBB$^+$97] G. Neumann, R. Backofen, J. Baur, M. Becker, and C. Braun. An information extraction core system for real world german text processing. In *In Proceedings of ANLP-97*, pages 208–215, Washington, USA, March 1997.

[NK02] Natalya F. Noy and Michael Klein. Ontology evolution: Not the same as schema evolution. Technical report, Stanford University, 2002.

[NM00] Natalya Fridman Noy and Mark A. Musen. Prompt: Algorithm and tool for automated ontology merging and alignment. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, Austin, TX, USA, 2000.

[NM01] Natalya F. Noy and Mark A. Musen. Anchor-prompt: Using non-local context for semantic matching. In *Workshop on Ontologies and Information Sharing at the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*, Seattle, WA, 2001.

[NS02a] Cheuk Hang Ng and Ka Cheung Sia. Peer clustering and firework query model. Technical report, The Chinese University of Hong Kong, 2002.

[NS02b] J.-Y. Nie and M. Simard. Using statistical translation models for bilingual ir. In Carol Peters, Martin Braschler, Julio Gonzalo, and Michael Kluck, editors, *CLEF*, volume 2406 of *Lecture Notes in Computer Science*, pages 9–26. Springer, 2002.

[NWQ$^+$02] Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjörn Naeve, Mikael Nilsson, Matthias Palmér, and Tore Risch. Edutella: A P2P networking infrastructure based on rdf. In *Proceedings to the Eleventh International World Wide Web Conference*, Honolulu, Hawaii, USA, May 2002.

[Ome00] Borys Omelayenko. Machine learning for ontology learning, 2000.

[Ome01a] Borys Omelayenko. Learning of ontologies for the web: the analysis of existent approaches, 2001.

[Ome01b] Borys Omelayenko. Syntactic-level ontology integration rules for e-commerce. In *Proceedings of The 14th International FLAIRS Conference (FLAIRS-2001)*, May 2001.

[PHG$^+$99] A.D. Preece, K.-J. Hui, W.A. Gray, P. Marti, T.J.M. Bench-Capon, D.M. Jones, and Z. Cui. The kraft architecture for knowledge fusion and transformation. In *Proceedings of the 19th SGES International Conference on Knowledge-Based Systems and Applied Artificial Intelligence (ES'99)*. Springer, 1999.

[PV99] Y. Papakonstantinou and V. Vassalos. Query rewriting using semistructured views. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 455–466, Philadelphia, PA, USA, 1999.

[Rab90] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In Alex Waibel and Kai-Fu Lee, editors, *Readings in Speech Recognition*. Morgan Kaufmann Publishers, Inc., 1990.

[RD01] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings to IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Heidelberg, Germany, November 2001.

[RK02] Sean C. Rhea and John Kubiatowicz. Probabilistic location and routing. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)*, June 2002.

[Sav02] Jacques Savoy. Report on clef-2001 experiments: Effective combined query-translation approach. In Carol Peters, Martin Braschler, Julio Gonzalo, and Michael Kluck, editors, *CLEF*, volume 2406 of *Lecture Notes in Computer Science*, pages 27–43. Springer, 2002.

[SD01] Michael Sintek and Stefan Decker. TRIPLE - An RDF Query, Inference and Transformation Language. Technical report, DFKI, 2001.

[SEA$^+$02] York Sure, Michael Erdmann, Juergen Angele, Steffen Staab, Rudi Studer, and Dirk Wenke. OntoEdit: Collaborative Ontology Development for the Semantic Web. In *Proceedings of the First Semantic Web Conference*, June 2002.

[SSDN02] Mario Schlosser, Michael Sintek, Stefan Decker, and Wolfgang Nejdl. HyperCuP - hypercubes, ontologies and efficient search on P2P networks. In *Proceedings to the Eleventh International World Wide Web Conference*, Honolulu, Hawaii, USA, May 2002.

[Ste02] Luc Steels. Language games for emergent semantics. *IEEE Intelligent Systems*, January/February 2002.

[Stu01] Gert Stumme. Using ontologies and formal concept analysis for organizing business knowledge. Technical report, Insitut AIFB, University of Karlsruhe, 2001.

[Stu02] H. Stuckenschmidt. Approximate information filtering with multiple classification hierarchies. *International Journal on Computational Intelligence Applications*, 2002. Accepted for publication.

[SvF$^+$00] H. Stuckenschmidt, F. van Harmelen, D. Fensel, M. Klein, and I. Horrocks. Catalogue integration: A case study in ontology-based semantic translation. Technical report, vreije universiteit amsterdam, 2000.

[SWKL02] Katia Sycara, Seth Widoff, Matthias Klusch, and Jianguo Lu. *LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace*, chapter 5, pages 173–203. Kluwer, 2002.

[Wac99] Holger Wache. Towards rule-based context transformation in mediators. In S. Conrad, W. Hasselbring, and G. Saake, editors, *International Workshop on Engineering Federated Information Systems (EFIS 99)*, Kühlungsborn, Germany, 1999. Infix-Verlag.

[WVV$^+$01] H. Wache, T. Voegele, T. Visser, H. Stuckenschmidt, H. Schuster, G. Neumann, and S. Huebner. Ontology-based integration of information - a survey of existing approaches. In H. Stuckenschmidt, editor, *IJCAI-01 Workshop: Ontologies and Information Sharing*, pages 108–117, 2001.

[YGM02] Berly Yang and Hector Garcia-Molina. Designing a super-peer network. Technical report, Standford, 2002.

[ZKJ01] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, April 2001.