MASTER’S THESIS

TOMI KANKAANPÄÄ

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Computer Science

MASTER'S THESIS
2.6.1999

Tomi Kankaanpää

DESIGN AND IMPLEMENTATION OF CONCEPTUAL NETWORK AND ONTOLOGY EDITOR

Supervisor: ma. prof. Jyrki Kontio

Instructor: Kuldar Taveter, M. Sc.
VTT Information Technology

HELSINKI UNIVERSITY OF TECHNOLOGY Department for Computer Science and Engineering	ABSTRACT OF MASTER'S THESIS

Author:	Tomi Kankaanpää
Name of the thesis:	Design and implementation of conceptual network and ontology editor
Translation in Finnish:	Käsiteverkkojen ja ontologioiden toimitinsovelluksen suunnittelu ja toteutus
Date:	2.6.1999 Number of pages: 74
Professorship:	Tik-76
Field of study:	Information Processing Science
Supervisor:	ma. prof. Jyrki Kontio
Instructor:	M. Sc. Kuldar Taveter VTT Information Technology

This thesis discusses the practical problems involved in creating and managing conceptual networks and ontologies. The key question of the study is: what kind of application and user interface features are beneficial to users working with knowledge repositories containing large amounts of interrelated information. The goal of this research is to come up with a core set of application functionalities, and to implement them in practice for tests and evaluation.

In order to describe the background of this study, and to illustrate complications involved in knowledge management, a compilation of various knowledge representation techniques and languages is presented. Benefits of visualisation and the use of visual languages are summarized, and two examples of visual methods of knowledge presentation are presented in greater detail. The role of ontology in knowledge representation and examples of ontology application areas are introduced to motivate the adoption of ontologies.

The main result of this study is the introduction of three key application features for ontological engineering: graphical representation of ontologies by visualising them as graphs; clustering to divide large model graphs into smaller dynamically loadable semantic fields; and parallel models to represent knowledge in multiple perspectives and to enable the creation of modular and reusable ontologies. Applicability and usability of these ideas are evaluated by implementing a prototype application called CONE, and by applying it in two different ontology management tasks: as a user terminal for accessing knowledge repository of a broker service application for electronic commerce, and as a knowledge engineer tool for creating models of corpus for purposes of multilingual information retrieval.

Keywords: ontology, knowledge representation, visualisation, Java

TEKNILLINEN KORKEAKOULU Tietotekniikan osasto	DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Tomi Kankaanpää
Työn nimi:	Design and implementation of conceptual network and ontology editor
Suomenkielinen nimi:	Käsiteverkkojen ja ontologioiden toimitinsovelluksen suunnittelu ja toteutus
Päivämäärä:	2.6.1999 Sivumäärä: 74
Professuuri:	Tik-76
Pääaine:	Ohjelmistojärjestelmät
Työn valvoja:	ma. prof. Jyrki Kontio
Työn ohjaaja:	M. Sc. Kuldar Taveter VTT Information Technology

Tämän diplomityön aiheena on tutkia ontologioiden ja käsiteverkkojen luomiseen ja hallintaan liittyviä ongelmia. Tutkimuksen avainkysymyksenä on, millaisilla sovelluksen ja käyttöliittymän toiminnoilla voidaan helpottaa työskentelyä kattavien ja paljon toisiinsa liittyvää tietoa sisältävien tietovarastojen kanssa? Tutkimuksen tavoitteena on löytää joukko sovelluksen ydintoimintoja, sekä toteuttaa niitä tukeva sovellusprototyyppi koekäyttöä ja arviointia varten.

Tutkimuksen taustoja ja tietämyksen hallintaan liittyviä ongelmia kuvaataan työn alussa, jossa esitellään joukko erilaisia tietämyksen esitystekniikoita ja kuvauskieliä. Tässä koosteessa kerrotaan visualisoinnin hyödyistä sekä erilaisten visuaalisten tiedon esitystapojen käytöstä tietämyksen esittämisessä. Kaksi visuaalista tiedonesitysmenetelmää käsitellään tarkemmin esimerkkeinä informaation esittämisestä kuvallisin menetelmin. Ontologioiden käyttöönottoa perustellaan kertomalla niiden sovellusalueista ja käytön tuomista eduista.

Työn tärkein tulos on kolme sovellusten perustoimintoa ontologioiden käsittelyyn: ontologioiden visualisointi graafeina, käsitteiden ryhmittely (klusterointi) pienemmiksi kokonaisuuksiksi tiedon käsittelyn nopeuttamiseksi ja helpottamiseksi, sekä rinnakkaisten mallien luominen eri näkökulmien kuvaamiseksi ja modulaaristen, uudelleenkäytön mahdollistavien, ontologioiden rakentamiseksi. Näiden toimintojen toimivuutta ja käytettävyyttä kokeillaan toteuttamalla niitä tukeva prototyyppi (CONE), jota sen jälkeen sovelletaan kahteen eri ontologioiden käyttötarkoitukseen: ensimmäisessä sovellus toimii loppukäyttäjän käyttöliittymänä sähköisen kaupankäynnin meklaripalvelun tietovarastoon, jälkimmäisessä itsenäisenä työkaluna eri kielten kieliaineiston mallintamiseen monikielisen tiedonhakujärjestelmän osana.

Avainsanat: ontologia, tietämyksen esittäminen, visualisointi, Java

Acknowledgements

This master's thesis research has been conducted while working at the knowledge engineering research group at the VTT Information Technology.

First of all, I'd like to express my gratitude to my instructor Kuldar Taveter and to my supervisor professor Jyrki Kontio for their guidance.

I would also like to thank my collegues of our research group, especially Aarno Lehtola for supporting the CONE project and for providing flexible working conditions, as well as Kristiina Jaaranen for her valuable feedback on CONE. I would also like to thank Marko Väisänen and Arto Laikari of Information Networks research group for engaging memories and companionship while working in the ABS project.

I am also very grateful to my family and friends who have helped me during this work. Thanks to my brothers Juha and Esa, and my father Pekka for their help for this work and my studies in general. Special thanks to Pia Venovirta for comments and corrections for improving the language.

Most of all I would like to thank my other half Maria. Without your support I would have never succeeded.

Espoo June 2nd, 1999

Tomi Kankaanpää

List of abbreviations

ABS	Architecture for Information Brokerage Services
AI	Artificial Intelligence
AWT	Abstract Windowing Toolkit
CG	Conceptual Graphs
CGIF	Conceptual Graph Interchange Form
CL	Controlled Languages
CLI	Command Line Interface
CONE	Conceptual Network and Ontology Editor
CORBA	Common Object Request Broker Architecture
EO	Enterprise Ontology
GFP	Generic Frame Protocol
GQM	Goal/Question/Metrics paradigm
IR	Information Retrieval
IS	Information System
JDBC	Java Database Connectivity
JOE	Java Ontology Editor
KQML	Knowledge Query and Manipulation Language
MVC	Model View Controller architecture
NCITS.T2	Technical Committee of Information Interchange and Interpretation of the National Comittee for Information Technology Standards
ODP-RM	Open Distributed Processing reference model
OML	Ontology Markup Language
PRE-CONE	Preliminary version of CONE
TINA	Telecommunications Information Networking Architecture
VoD	Video on demand
VTT	Technical Research Center of Finland (Valtion Teknillinen Tutkimuskeskus)
WWW	World Wide Web
XML	Extensible Markup Language

1 Introduction

1.1 Background

The fundamental problems of artificial intelligence research are knowledge representation and search. In order to create applications that are capable of intelligent reasoning, application developers must design means of representing information (knowledge representation, or data structures) in a way that can be utilised by the problem-solving computing process (search, or algorithms). It is evident that data structures and algorithms can be selected in a wide variety of ways, each of which has its own benefits and drawbacks. The traditional research on knowledge representation has approached the problem by describing the domain in such a form that it supports problem-solving process as well as possible. Unfortunately, this strategy entails that knowledge representation is very specific to the domain in question, and cannot be easily used outside the system it originates from. This is a major problem in today's networked world, where knowledge sharing is a priority.

Instead of looking at the domain from the specific problem-solving point of view, research on ontologies has adopted a slightly different approach — a modelling approach. By describing parts of the world "as they are", we may provide generic models that lay foundation for all activity concerning them. These models, referred as ontologies, are explicit specifications of objects and relationships existing in the domain under study. If formalised in a generic but rigorous way, ontologies may be easily adopted for many different purposes.

Building comprehensive ontologies is an arduous task. Finding relevant concepts and their interrelations requires thorough knowledge and understanding about the domain, and ability to extract the most essential characteristics of domain objects. Even one simple domain can easily yield hundreds of concepts. Various relationships between domain objects form very complex webs of dependencies, which may be difficult to track and manage. A huge amount of interlinked information is an integral part of the nature of ontologies, and efficient tools are needed to manage this complexity. Currently, such tools are few and lack of them has severely hindered the adoption of ontologies.

The purpose of this study is to examine methods that enable and ease the construction of ontologies, and to provide a generic ontology tool that is applicable to different environments and to various knowledge representation purposes. Currently, there is no standard or de facto procedure for formally encoding ontologies and thus most of the research projects working on ontologies have built their own domain specific implementations. However, this is in conflict with the fundamental principles of ontology — genericity, reusability and shareability of knowledge.

Research on ontologies is an ongoing process at VTT Information Technology. This work has been conducted during two different projects, each of them using ontologies for different purposes. The goal of the ABS (Architecture for Information Brokerage) project has been to develop an infrastructure for brokerage in context of electronic commerce occuring in the Internet, relying on ontology-based knowledge representation scheme of the content and service provider domains. Webtran, on the

other hand, uses ontologies in describing language-specific representations of concepts for purposes of automatic natural language translation. Both of these projects have had a great effect on this work by generating ideas on how ontologies eventually should be managed. Furthermore, both projects have been acting as active testbenches for different design and implementation details.

1.2 Research approach

The starting point of this work is the requirements set forth by ABS and Webtran projects. To start with, a survey of ontology support tools is performed to obtain a broader perspective on ontological engineering and basic problems involved with it. A set of applications are examined and heuristically evaluated considering their applicability to generic ontological engineering activity. The conclusions drawn from this evaluation combined with the requirements of the VTT projects define the guidelines for this research.

In the course of this work, a tool application with experimental ontology support features is implemented. The application, called "Conceptual Network and Ontology Editor" (CONE), is a combined browser-editor providing a graphical, graph-based visualisation of ontologies, and a set of tools for creating and managing ontologies. In addition to these features, the graphical user interface (GUI) contains multiple views to the ontology and an active grouping of concepts and relations, referred as clustering, to ease navigation in large and complex ontologies. Furthermore, bridges, special kind of relations, are introduced to represent multiple perspectives on the same information and to enable the creation and use of modular and reusable ontologies.

These features form the core functionality of CONE technology and are the main focus of assessment in this study. In order to validate the usability and applicability of these features, they are experimented in two small-scale empirical studies. In the first one, a subset of CONE features is embedded into ABS broker terminal application and evaluated as a part of ABS project's field trials. In this test, the focus of assessment is on graphical representation and its suitability as an end-user interface. The second evaluation on CONE is performed in the context of ontological engineering for automatic natural language translation. In this evaluation, the main focus is on features supporting creation and navigation of ontologies, and on the applicability to building ontologies with large number of concepts, and multiple parallel viewpoints. Finally, in the light of these experiences, some conclusions about the feasibility of CONE features can be made.

The scope of this work is the study of methods that make working with ontologies easier. The goal is to look for answers to questions like "What kind of application and user interface features can be beneficial to users working with extensive knowledge repositories?" and "What are the basic requirements for building reusable and sharable ontologies?". The nature of ontologies is highly interdisclipinary and is covered by many fields of scientific studies, such as philosophy, linguistics and cognitive science. Ontologies encompass many different aspects and many careful research efforts are required to cover ontologies in their entirety. To keep the focus of this work clear, low level formalisms for representing ontologies, techniques for deciding on domain conceptualisations, or using ontologies for knowledge acquisition are not discussed here.

1.3 Structure of the thesis

In addition to this introduction, the thesis consists of five chapters. In chapter 2 an introduction to the theory behind knowledge representation and ontologies, as well as a brief look on the problems and application areas involved in ontological engineering is presented. Chapter 3 begins with a review of some existing applications suitable for ontological engineering purposes, and continues with propositions on how to harness the power of ontologies into concrete tool application. After that, a description of CONE architecture accompanied with notes on the design and implementation are presented. Chapters 4 and 5 describe the use of CONE in two different contexts and present the results of two brief evaluations carried out to get ratification for the ideas presented in this thesis. Finally, chapter 6 concludes the discussion by summarising this work and its results, and gives some ideas for further work.

1.4 Related work

Research on ontologies is a relatively new branch of artificial intelligence. Most of the work has been directed to theoretical and representational aspects of ontologies (Gruber, 1993; Guarino, 1995; Uschold and Gruninger, 1996) and the practical considerations concerning difficulties in creating and managing ontologies have been laid aside. However, a few projects have tackled these problems: JOE project (Mahalingam and Huhns, 1995) at the Center of Information Technology in the University of Southern Carolina uses ontologies built by using JOE tool as a basis for semantic reconciliation for distributed heterogeneous information environments. Ontolingua (Gruber, 1993), CODE4 (Skuce, 1995), and GrIT (Eklund, Leane, and Novak, 1993) are ontology research projects that have developed their own tools for creating ontologies. Some of these projects are discussed in greater detail in chapter 3.

Ideas of using bridges and viewpoints as a means of describing multiple perspectives on knowledge have emerged in other research efforts as well. TROPES (Euzenat, 1993) is a multi-perspective representation system that allows knowledge to be viewed from different perspectives and new knowledge to be inferred using bridges connecting concepts in different viewpoints.

2 Knowledge representation and ontologies

The description of a domain under investigation, its real-world concepts and relations between them, is called the conceptual model of the domain. To build an application utilising the information on the domain, the conceptual model must be transferred into a form that can be modelled in the computer memory, and algorithms using this model must be developed. It is clear that the model can be constructed in a variety of ways, each having their own benefits and drawbacks. The approach used by the discipline of artificial intelligence is to formulate the conceptual model in terms of knowledge, to declaratively describe domain objects and their relations.

The way the conceptual model is represented is often referred to as a knowledge representation scheme and it is the foundation upon which intelligent applications are built. The following chapters describe the principles behind knowledge representation and theory of ontologies.

2.1 Knowledge representation

Knowledge engineering is a disclipine of creating knowledge-based systems — applications that encompass knowledge of some problem domain, and that are capable of solving problems related to that domain. The basic problems of designing such an application, and actually the fundamental problems of the entire artificial intelligence theory, are knowledge representation and search. The term "knowledge representation" is used to signify the way knowledge is formally described inside the machine, while term "search" refers to the way the information is utilised. The main concern of knowledge representation is to gather all essential information on the problem domain, and make that information easily accessible to the problem-solving process (Luger and Stubblefield, 1998).

In general, artificial intelligence is concerned with qualitative, rather than quantitative problem solving. As a basic building block of AI applications, knowledge representation scheme should also reflect this idea by supporting reasoning instead of calculation. This is done by organising the knowledge into easily processable form, classifying information by its properties, preprocessing the information, and so on. By using proper knowledge representation scheme and methods, we can provide much more effective tools for problem solving than traditional approaches, like a single well-defined algorithm working on arrays of bytes in computer memory.

2.1.1 Issues in knowledge representation

Representing knowledge inside the machine has proved to be a nontrivial task and involves several questions. What kind of knowledge can exactly be represented? Describing objects by their strict properties is a relatively easy task, but how can we include notions of comparison (e.g. how to capture knowledge of "Lucy is taller than Marty" in a way that supports answering the question "How tall is person X?"), or referral (e.g. "Our house is next to your aunt's house"). How could this kind of complex relations be rationally described? Is there possibly a simple set of symbols and rules with adequate representational capabilities that can be used for this purpose?

A related issue concerns with the precision and granularity of knowledge description. To effectively solve domain specific problems, how much of the domain knowledge needs to be described, and on what level should it be described? Some knowledge representation schemes use unrelated symbols to describe entities, while other schemes assemble pieces of information into structures that describe entities as a whole. The problem is closely affiliated with the problem of organising knowledge — can complexities of the representation be reduced by organising the information in hierarchical or other relational structures? Knowledge organisation and the granularity of knowledge representation have critical impact on how flexibly information with growing complexity can be managed, and how this information can be used for making deductions.

Luger and Stubblefield (1998) see the issue of distinguishing *intensional* and *extensional* knowledge as one of the key problems of knowledge representations. The term *extension* is used to refer to the set of all things denoted by a given concept; for example, "ball" has as its extension the set of all balls. Term *intension* determines the abstract meaning of a concept, such as definition of a ball in terms of its roundness, its ability to roll, bounce, be thrown and its use in games. Even though these two different definitions characterise the same concept, their role in understanding is very different. Describing extensional knowledge is quite easy, but how to describe intensional factors so that they can be used in a problem-solving process?

An extraordinary property of human brain that makes us more advanced than any other animals is the high level of consciousness — our well-developed ability to understand our own actions and thoughts. This self-awareness is a fundamental aspect of human mental behavior and is suspected (Lindsay and Norman, 1977) to be the basis of all intelligent choices, like learning and understanding. This kind of ability to "know about all things that we know", often referred as *meta-knowledge*, is also desireable for a knowledge representation scheme. Meta-knowledge is important if we wish to reason about knowledge itself, as in the sentence "Tom believes that Mary wants to marry a sailor". The representation scheme should be able to store knowledge about the objects and structures of the representation scheme itself.

In addition to these problems, concrete implementation requirements also greatly affect the design of a representation scheme. To be rich in expression is not adequate; the scheme must support efficient execution of resulting code, and it should provide a scheme for expressing the knowledge that is understandable and manageable by its human and machine users. Typically, the selection of the knowledge representation scheme is highly dependent on the nature of the problem at hand. The problem areas are usually very complex and diverse both in content and extent, making it very difficult to compromise between the detail level of problem description and efficiency requirements. Expressiveness and efficiency are the key factors that greatly affect the competence of a representational scheme. Many highly expressive representations are often too inefficient to be really used, and must be discarded in favor of less expressive counterparts.

2.1.2 Knowledge representation schemes

During the past 30 years, a wide variety of knowledge representation schemes have been developed, each of which have their own benefits and drawbacks. These schemes can be roughly classified into four categories (Luger and Stubblefield, 1998):

- *Logical representation schemes*. Expressions of formal logic are used to describe the knowledge and inference rules. First-order predicate calculus is the most commonly used logical representation scheme, and there is a variety of representional schemes based on it.

- *Procedural representation schemes*. The problem-solving knowledge is stored as a set of instructions that can be executed to obtain the results of the problem. Contrary to other scheme categories, procedural representation schemes concentrate on *how* to solve the problem in question rather than declaratively describe the problem domain.

- *Network representation schemes*. Network representation schemes employ visual graph notation to store knowledge: concepts are displayed as graph nodes, and relations as graph edges connecting nodes.

- *Structured representation schemes*. A modified form of network representation schemes, in which the use of concept nodes is extended by allowing more detailed data structures to be attached to the node. In addition to representing a certain concept, a node may contain values, references to other nodes, or even executable code for performing some predefined tasks.

The following chapters introduce two different basic knowledge representation schemes that are the foundations for most of the knowledge representation methods used today, i.e. predicate calculus and frame-based systems.

2.1.3 Predicate calculus

Traditional symbolic logic has two branches, propositional calculus and predicate calculus. Propositional calculus deals with statements (called *propositions*) such as "Lilian is the mother of Leslie.", composed of single atomic symbols (typically P, Q, etc.). There is no way of accessing the individual components of the assertion. However, predicate calculus provides this ability by anylyzing statements into finer components and representing them by means of predicates. A predicate is a symbol indicating specific relation between entities. To continue with our example, the proposition to describe the mother-child relationship between Leslie and Lilian would be expressed by the predicate mother(Lilian, Leslie).

Symbols of predicate calculus are used to denote either *truth values* (true and false), *variables*, *constants*, *functions*, or *predicates*. Constants name certain objects or properties, while variables are used to denote general classes of objects or properties, or unspecified objects in the domain. Functions in turn denote a mapping of one or more elements in a set (called the *domain* of the funcion) into a unique element of another set (the *range* of the function), elements of both sets being objects in the domain of discourse. Each function is associated with *arity*, or the number of domain elements mapped into single element in the range of a function.

Predicate symbol refers to a particular relation in the world of discourse. Unlike function symbols, which are used to refer to mapped objects without using their names, predicate symbols state that relations hold among certain objects. Predicates are defined by their name. For example:

```
cat(kati)
dog(jalo)
pals(kati, jalo)
```

In the example above, the first statement indicates that the predicate `cat` holds for a specific entity "Kati", denoted by the symbol `kati`. In similar way, the predicate `dog` indicating property "is a dog" is true for the entity represented by the symbol `jalo`. The third statements declares the relation that exists between these entities: entities denoted by symbols `kati` and `jalo` are known to be friends.

To specify unknown entities of the domain, or predicates holding for all objects of the domain, special symbols called *universal qualifier* (∀) and *existential qualifier* (∃) may be used. By combining these constraints for variables with standard logical connectives (∧,∨, ¬, ⇒, =) of propositional calculus, new sentences may be combined. For example, sentences

```
∀x ∀y (likes(x, y) ∧ likes(y, x) ⇒ pals(x, y))
∃y (human(y) ∧ master(y, kati) ⇒ master(y, jalo))
```

indicate that all objects that like each other are considered pals, and that if there exists some particular human that is master of `kati`, he is also the master of `jalo`.

Predicate calculus semantics provides the basis of determining the truth value of expressions. The truth value of an expression dependes on the mapping of predicate calculus symbols into entities and relations of the domain of discourse. The truth of relationships in the domain determines the truth of the corresponding predicate calculus expression. If the truth value of a symbol or a sentence combined from other symbols is true, it is said to *satisfy* that sentence. A similar interpretation that makes a given sentence true for all sentences in a particular set of sentences, (i.e. satisfies all sentences of the set) indicates that the given sentence is a logical implication of the sentences in that set. This concept of "logically follows" forms the formal basis for defining *inference rules* — rules used for inferring the truth value of a sentence form a set existing assertions. An inference rule is basically a mechanical means of producing new predicate calculus sentences from other sentences. A simple but powerful rule of *modus ponens* is an example of an inference rule: if senteces P and P ⇒ Q are known to be true, then modus ponens lets us infer Q. The rule may also be applied to expressions containing variables, as can be seen in the previous example. Another inference rule, called *unification*, can be applied to find variable substitutions to make two expressions match, i.e. to infer domain instances that can be assigned to variables preserving the truth values of assertions.

2.1.4 Frame-based systems

Another basic example of knowledge representation scheme is frame-based systems (or schemas).

The basic idea of frames is simple: a frame embodies a concept or a physical object. Frames contain named attributes referred as slots, and slots have values assigned in them. A value can be anything from a numeric value to a reference to another frame. Each frame is a kind of a data structure, similar in many respects to the traditional record data structure, except that it can be customized to a specific individual and not only for a generic type. Slots of the frame typically contain information such as (Luger and Stubblefield, 1998):

1. Frame identification information to associate the frame with some specific entity or a type in the problem domain.

2. Relationship of this frame to other frames.

3. Values of entity properties. These values may also be used as a frame match criteria (i.e. specify a range of values) to check whether a certain entity matches the prototype defined by the frame.

4. Procedural information on use of the structure defined. Procedural code might be attached to a slot to dynamically evaluate the value of the slot.

5. Frame default information. Slot value indicates the default value that is used if no other value is available.

6. Unspecified. Many slots may be left empty to indicate that the value is not yet specified or irrelevant at the time.

Complex structures can be described by using a number of individual frames connected to each other by assigning references to other frames as values of frame slots. Large descriptions describing entire problem domains often form complex networks of frames. An example (from Luger and Stubblefield, 1998) of a frame description describing a hotel room is shown in Figure 2-1.

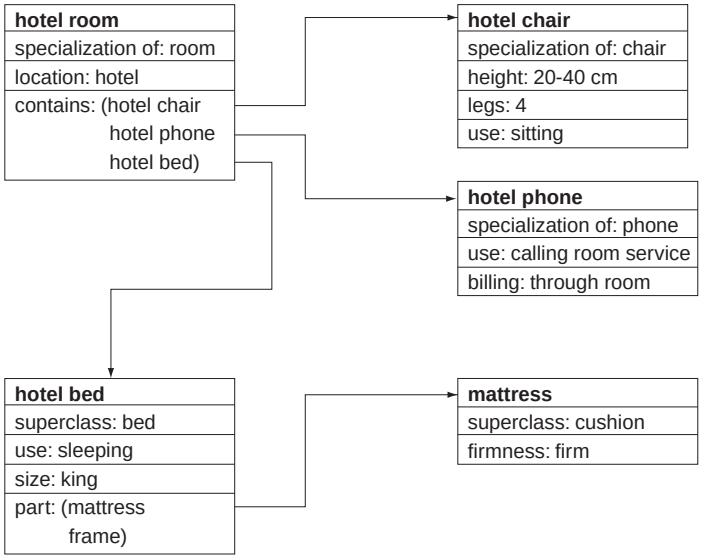


Figure 2-1

Frame-based systems allow knowledge to be organised hierarchically. It is often desirable to be able to consider objects as whole entities, and have their details hidden until explicitly requested. For example, understanding the functionality of a complex mechanical machine such as a car is not possible unless it is viewed at different abstraction levels.

Additional benefit of the hierarchic organisation is the support for inheritance. The slots and default values of a class frame are inherited across the class-subclass and class-member hierarchy. This aspect makes frame-based systems very similar to object-oriented models. However, the way frames are used differs from the traditional OO approach: frames are typically manipulated by an external algorithm, while object oriented models describes entities as classes with both properties and behavior. Furthermore, frames do not provide support for information hiding — all slots of a frame can be accessed outside the frame.

2.2 Knowledge representation languages

Knowledge representation schemes alone are not adequate to build knowledge-based applications. Representational schemes can be regarded as generic ideas or plans on how to represent knowledge inside the machine, but they do not provide any details on how the scheme is actually implemented. There is a clear difference between the *scheme* itself and the *medium* of its implementation: the scheme indicates the way of solving the problem (as data structures), the medium indicates how the problem-solving method is implemented (as programming languages).

To formally describe something, we need this kind of a medium (a language) to express things and ideas using symbols (for example, words in written language), sentences and clauses. Together, these small syntactic constructs form valid sentences with well-defined semantic meaning, presuming that they follow the rules of the language. A knowledge representation language is defined by two primary aspects:

- The *syntax* of a language describes the possible configurations that can constitute sentences. Sentences may appear as written words on paper, as well as a sequence of bytes in a computer memory.

- The *semantics* determines the facts referred by sentences. Without precise semantics, a sentence is just a sequence of letters on page or bytes in the computer memory. With semantics, sentences make claims about the world and thus provide explicit meaning for syntactic structures.

With a relatively small vocabulary and clearly defined semantic rules, we can easily describe enormous amounts of information. To apply this idea in field of intelligent applications, the rules of the language should be adequate to express knowledge about the domain: describe domain entities, their properties, and relations between them. In addition, it should be formalised in a way that emphasises and carries meaning, and enables the intelligent access to this information.

2.2.1 Requirements

Evidently, knowledge can be represented and transferred in a wide variety of ways, the most familiar one being the spoken words we exchange with each other every day. What makes a certain representation language better than the other? According to Luger and Stubblefield (1998), representing knowledge pose following requirements to representation languages. A good representation language should:

1. Handle qualitative knowledge: the knowledge of the problem should be described in a way that encompasses all relevant information in problem-solving point of view. Rather than simply describe objects and properties for them, the knowledge representation language should concentrate on relations between objects, intentions and consequences.

2. Support inference of new knowledge from a set of facts and rules: instead of describing and recording vast amount of loose, unconnected pieces of information, the language should subsidize knowledge representation by defining a sound set of facts, and a set of ground rules that can be used to infer new information.

3. Represent both generic principles as well as specific situations: to be as general as possible, knowledge representation language should include a method for

generalisation. If some fact or rule is valid for a certain object, it is often valid for other objects as well. Instead of defining the rule or the fact for each object separately, it makes more sense to generalise the rule using variables and qualifiers: variables of a rule can be assigned with values of the domain, keeping the rule still valid.

4. Capture complex semantic meaning: knowledge representation language should support inference and discovery of complex relations by capturing and representing knowledge in a form that is usable to its (human or machine) users. To describe an entity, it is not adequate to just list its properties and parts; a valid description of knowledge should be able to describe the ways in which parts of a whole are combined and what kind of interactions exist between them.

5. Allow meta-level reasoning: knowledge itself is negligible if there is no knowledge about the information itself, or knowledge of how to use it. This "knowledge of what you know" is refererred to as *meta-knowledge*, a higher level of understanding that is the basis of utilising the knowledge. To be truly usable, knowledge representation scheme should be able to store and handle meta-knowledge, and it should be able to use it to explain how a problem was solved or why certain decisions were made.

Next two examples of knowledge representation languages are represented. Neither of them have all of the above requirements built in, but both of them provide basic features that may be used to support them.

2.2.2 PROLOG

Even though PROLOG (PROgramming LOGic) is usually categorised as a logic programming language rather than a knowledge representation language, its introduction in context of knowledge representation is well justified. PROLOG is built upon first-order predicate calculus introduced in chapter 2.1.3. As opposed to many other programming languages, its declarative semantics allows programmer to describe the problem to be solved instead of describing the detailed steps required to solve the problem. A PROLOG program is a set of specifications describing the entities and relations in the problem domain. In this sense, PROLOG can be considered as a kind of a knowledge representation language.

A PROLOG intepreter functions by interpreting and executing statements expressed using the PROLOG syntax. The syntax[1] bears a great resemblance to that used to represent statements of the first-order predicate calculus. For example, to define the example predicate calculus statements of chapter 2.1.3, the following PROLOG clauses would be used (`?-` is the PROLOG intepreter prompt):

```
?- cat(kati).
?- dog(jalo).
?- pals(kati, jalo).
?- pals :- likes(X, Y), likes(Y, X).
```

Each interpreted statement is stored to a set of statements, referred as the database of the problem. The database is considered as an absolute and definite describtion of the

[1] There are numerous dialects of PROLOG. The syntax used here is the original Warren and Pereira C-PROLOG.

world, according to the principle of *closed world assumption* which states that everything that is not provably to be true is assumed to be false. To utilise the knowledge, PROLOG can be used to query the database. The interpreter processes the query by searching the database to see whether the query is a logical consequence of the specifications declared in the database.

2.2.3 Knowledge Interchange Format (KIF)

Knowledge Interchange Format (KIF) (Genesereth and Fikes, 1992) is a representation of first-order predicate calculus with a linear ASCII syntax. It includes a sublanguage for defining named functions, relations, object constants, and it supports reasoning about relations, functions, and expressions by including them in the domain of discourse. KIF notation is a LISP-like notation with prefix syntax; objects are denoted by object constants (LISP atoms), or term expressions, which are constructed from lists whose first element is a function constant. KIF sentences are formed from lists whose first element is a relation constant and remaining elements are terms, or by combining other sentences using basic logical operations (conjunction, disjunction, implication, and negation). KIF provides also support for existential and universal quantification with special ? prefix.

For example, the following KIF sentence represents the sentence “All writers are misunderstood by some reader” (Gruber, 1993).

```
(forall ?W
    (=> (writer ?W)
        (exists (?R ?D)
            (and (reader ?R)
                 (document ?D)
                 (writes ?W ?D)
                 (reads ?R ?D)
                 (not (understands ?R ?D))))
```

In this example, `W` is universally qualified, and `R` and `D` are existentially qualified variables. `writer` is a relation constant and the statement `(writer ?W)` that states the w is a writer. Symbol `=>` in turn indicates implication.

In addition to these basic atoms, KIF specification provides definitions for primitive object types such as sets, lists, relations, and functions.

2.3 Visual languages for knowledge representation

As stated in the previous section, knowledge representation is often based on some knowledge representation language. In general, languages build on a purely textual representation with strict syntactic and semantic rules. Domain concepts, their properties, relations and restrictions between them, are all represented by words and sentences of the representation language. Construction of large knowledge bases involves dealing with vast amounts of declarations in a particular representation language. Using this kind of method to describe knowledge is definitely an exact and exhaustive approach, but may prove to be unsuitable when it comes to human users and their ability to perceive, understand and manage large amounts of information.

By visualising language constructs — that is, using so called visual languages — some of these problems may be answered. But what is it actually that qualifies a language as "a visual" language? Kremer (1998) uses this term to refer to any form of

communication that relies on two- or three-dimensional graphics rather than simply linear text. Textual communication is often supplemented with visual properties (such as character types, styles, structure, or layout), but a knowledge representation language can be regarded as visual only if it is based on a pictorial expression.

The use of a visual language is compelling for many reasons. Their graphical nature can act as an analogical representation of the domain in a way that is not possible with approaches based on pure text. Graph, being able to explicitly represent relations between concepts by using nodes and arcs is an ideal method for displaying domain concepts and relations between them. Mapping between nodes and concepts, and edges and relations, provides a natural medium of representing relational structures.

Other benefits of using a visual language seem to be self-evident: while there is no evidence that abstract reasoning would be pictorial in nature, most users seem to prefer graphical representation to textual representation. Learning and execution times of actions involving larger amounts of information have been noticed to be faster with visual representation of the information than with representation based on linear text encoding (Nosek and Roth, 1990). The human short term memory is limited, but visual organisations enable brain's sensory information storage and ability to break down complex structures into more easily manageable chunks (Lindsay and Norman, 1977).

Furthermore, visual languages can be effectively applied in direct manipulation user interfaces (Shneiderman 1997), where the user handles the actual application objects by manipulating their visual representation on the computer screen with a mouse or other pointing device. The adoption of visual schemes opens new possibilities for building knowledge-based systems that require human interaction.

2.3.1 Semantic networks

Semantic networks have been used for knowledge representation since the early days of artificial intelligence. In fact, the earliest work on this area was made by Charles Sanders Peirce (1839-1914), a logician, mathematician, geodesist, and the first modern experimental psychologist in the Americas. He developed a graphical system of logic called existential graphs and used it to systematically record observations of the world around him. Contemporary semantic networks bear great resemblance to the Peirce's existential graphs, and his graphs have been the inspiration for many researchers in fields of artificial intelligence and philosophy.

In the field of psychology, graph structures have been used to represent structures of concepts and associations. Otto Selz (1881-1943), a german psychologist of Wuerzburg University used graphs to represent different concept hierarchies and the inheritance of properties. Lindsay and Norman (1977) conclude to the same idea of representing human brain and its information storage as a semantic network: concepts, generalisation, specialisation, defaults and exceptions and their properties can be described in a simple but yet expressive way. An example of a semantic network is displayed in Figure 2-2 (Lindsay and Norman, 1977).

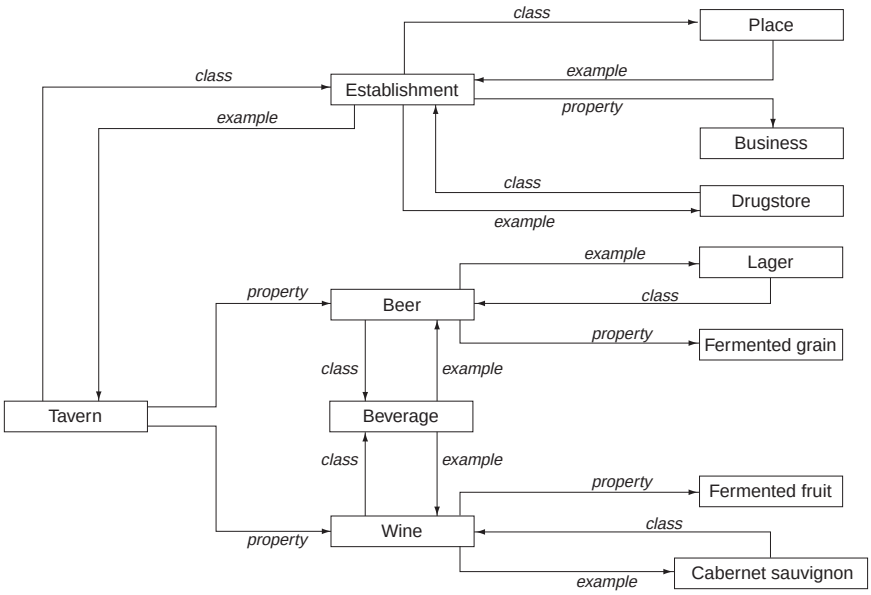


Figure 2-2

More recently, semantic networks have been subject to an interest motivated by the search for methods of organising and displaying larger and more complex knowledge bases. New interest in object-oriented programming and object-oriented databases has also focused attention on the object-centered aspects of semantic networks, especially type hierarchies and inheritance.

In general, the term "semantic network" encompasses an entire family of graph-based visual representations. They all share the basic idea of representing domain knowledge in the form of a graph, but there are some differences concerning notation, naming rules or inferences supported by the language. Term "semantic network" is also often used in a way that is almost synonymous for conceptual graphs. The following chapter describes the theory of conceptual graphs in detail and explains why this use of terminology is misleading.

2.3.2 Conceptual graphs

Conceptual graphs is a knowledge representation language introduced by John F. Sowa (Sowa, 1984), which uses graphical representation as a method of encoding knowledge. In a conceptual graph, concept nodes are used to represent entities, attributes, states and events, while relation nodes are used to show how these concepts are related to each other. Using a well-defined but yet expressive notation, conceptual graphs are capable of representing an extremely wide range of knowledge forms — from clauses of predicate logic to a natural language.

Sowa (1984), clearly distinguishes the ideas of conceptual graphs and semantic networks: each conceptual graph asserts a single proposition, while semantic networks are much larger. Sowa suggests that semantic networks are entities that embed conceptual graphs. Through a semantic network conceptual graph's concepts and relations are linked to context, language, emotion, and perception. Concepts may be associated with percepts for experiencing the world and motor mechanisms for acting upon it; they may be associated with the words and grammatic rules of a language;

conceptual graphs may be linked to some context or episode to which they are relevant. All these factors together form a semantic network, in which conceptual graphs may be used to describe assertions.

The abstract syntax of conceptual graphs is built upon a graphical notation. In this syntax, key elements are conceptual graphs, concept nodes, conceptual relations, and the type hierarchy.

A *conceptual graph* is arbitrarily complex, but a finite, connected, bipartite graph. They are finite in a sense that any human brain or computer memory can manage only a finite number of conceptual nodes and relations. Conceptual graphs are said to be connected, because two unconnected parts of a graph can be simply regarded as two conceptual graphs. They are also always bipartite, because by definition there are only two kinds of nodes — concepts and conceptual relations — and every arc must connect two nodes of different kinds.

An example of a conceptual graph is displayed in Figure 2-3.This graph displayes a graph for sentence "a person is between a rock and a hard place".

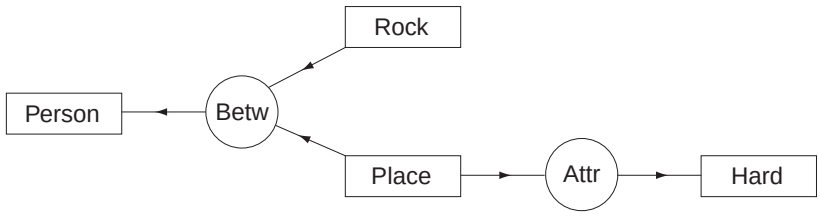


Figure 2-3

Conceptual graphs were originally designed as a graphical notation (display form), that definitely is the primary form of displaying conceptual graphs. However, to be more easily applicable to computers, a textual form (linear form) has been developed. This form uses a flattened text representation of a graph. As an example, the conceptual graph of Figure 2-3 may alternatively be represented as follows:

```
[Person]<-(Betw)- <-1-[Rock]
                  <-2-[Place]->(Attr)->[Hard]
```

In this form, concept names are surrounded by square brackets, and conceptual relation names by normal brackets. The hyphen after the relation indicates that its other arcs are continued on subsequent lines.

A *concept node* in a conceptual graph is a manifestation of some discrete real-life concept. On the one hand, node can represent concrete concept like a car, or a house (or a specific instance, like "John's house"); on the other hand it can represent an abstract concept like love, function or justice.

In a conceptual graph, every concept represents a unique individual of some type or a class defined in the type hierarchy (see below), and all concepts with the same type label represent distinct individuals of the type. To visually indicate this property, each concept node is labeled with a string consisting of identifier called the referent, separating character ":", and a type label. The *referent* of the concept is indicated by either an *individual* or *generic* marker. Individual marker, which is a serial number like #20894, is used to refer to a named or unnamed, but yet specified individual in the

world of discourse.On the other hand, generic marker * may be used to refer to an unspecified individual. To separate between multiple unspecified individuals, conceptual graphs allow the use of named variables, marked as *x, or *person. By convention, plain generic marker * is often omitted from the label, leaving only the bare type label. Table 2-1 contains some examples of naming scheme:

[emma:cat]	a cat named Emma
[#3287:cat]	a specified, but unnamed cat
[*:cat]	unspecified cat
[*x:cat]	unspecified cat referred by variable x
[cat]	unspecified cat

Table 2-1

A *conceptual relation* in a conceptual graph indicates a relation that involves one or more concepts. Conceptual relations are not restricted to any arity. Relation node may have any number of connecting arcs; the number of arcs belonging to a relation indicates relation's *valence*. A relation of valence n is said to be n-adic; term *monadic* is a synonym with 1-adic, *dyadic* with 2-adic, and *triadic* with 3-adic. This representation of relations has the benefit of not restricting relations to any specific valence. An example of triadic relation (between) is shown in Figure 2-3.

Type hierarchy connected with a conceptual graph defines types of concepts and conceptual relations. The type hierarchy is a lattice of is-a relations, forming a multiple inheritance system supporting multiple parents and children of types. The type hierachy is freely definable with restriction that every pair of types must have minimal common supertype and a maximal common supertype (for types s and u, v is a minimal common supertype if $s \leq v$, $u \leq v$, i.e. both s and u are subtypes of v; for any w, a common supertype of s and u, $v \leq w$. Maximal common subtype is has a similar definition). To keep the definition of lattice valid, two special types, the *universal type* (indicated by symbol $\top$) and the *absurd type* (indicated by symbol $\bot$) have been included in the type hierarchy.

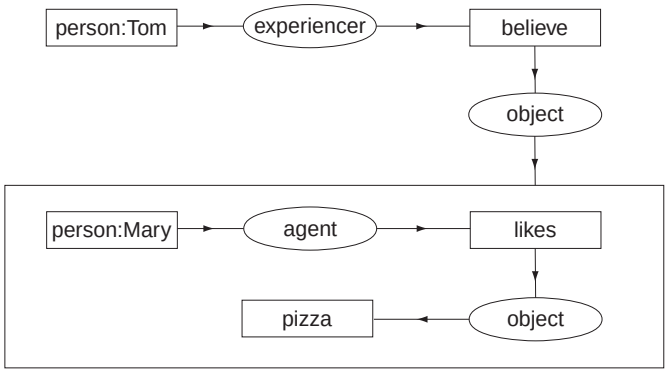


Figure 2-4

The theory of conceptual graphs allows entire conceptual graphs to be nested inside other concepts (in other graphs) to form a hierarchical structures of graphs. A concept with a nested conceptual graph that describes concept's referent is called a *context*. Use of contexts allows us to use conceptual graph theory to express different kinds of

propositions, for example the proposition "Tom believes that Mary likes pizza" shown in Figure 2-4.

Conceptual graphs are a notation for representing knowledge, but to serve as a basis for intelligent applications they must also support computation and automatic reasoning. The theory of conceptual graphs does this by including a number of operations that can be used to make deductions using existing graphs. These operations, so called *canonical formation rules* are (Sowa, 1984):

1. Copy: creates an exact copy of a given graph.

2. Restrict: allows a a concept to be replaced with a concept that represents its specialization: type of concept may be replaced with its subtype (as long as it is consistent with the referent of the concept), or if concept is a generic, its referent may be replaced with an individual marker.

3. Join: two conceptual graphs can be combined into a single graph. If there are two concept nodes (in different graphs) that are identical, the graphs may be linked together by removing one copy of the concept and relinking all of its adjacent conceptual relations to the remaining concept node.

4. Simplify: duplicate conceptual relations may be removed from the graph.

By applying these rules to a set of conceptual graphs, new graphs with deducted knowledge can be created. By means of join and restrict operations, an implementation of inheritance can be provided: replacement of a generic marker with individual marker implements inheritance of properties of a type by an individual; replacement of a type label by some of its subtypes defines inheritance between a class an its superclass.

It should be noted that canonical formation rules are not actually inference rules. The results of these operations are not guaranteed to always produce valid (i.e. truth-preserving) graphs. For example, restriction of conceptual graphs specifying two true sentences "Some girl is eating fast" and "Sue is eating pie" does not necessarily indicate that it is Sue who is eating pie fast. A graph derived using canonical formation rules may be logical and meaningful, but false. However, these rules have an important ability of preserving meaningfulness: no nonsensical graph (like the one specifying the sentence "Colorless green ideas sleep furiously", Chomsky's classical example (Chomsky, 1957) of grammatically correct, but meaningless sentence) can be produced from meaningful graphs using canonical formation rules. This is an important property when applying the theory of conceptual graphs to applications of natural language understanding.

The elements of the conceptual graph theory described above provide a relatively simple but expressive method of encoding knowledge in visual representation. The language has been adopted for many uses, and by many different communities. A proposition of conceptual graph standard (NCITS.T2/98-003, 1998) is currently under construction by a NCITS Committee on Information Interchange and Interpretation. The standard proposal includes the specification of syntax and semantics of conceptual graphs, and a specification of their representation as machine-readable character strings in the conceptual graph interchange form (CGIF).

2.3.3 Other visual knowledge representation languages

A wide variety of visual knowledge representation languages have been developed, primarily to provide an alternative visualization method to traditional textual grammar. Most of these languages follow the principle of *concept maps*, representing knowledge as a graph. Naturally, each language has its own notation and is designed for a certain purpose and with certain knowledge representation scheme in mind, but an unifying feature for them is to represent knowledge as nodes and edges of a graph. Some examples of visual KR languages are: KRS (Gaines, 1991) – a visual language based on textual knowledge representation language CLASSIC; KSM (Knowledge Structure Manager) (Cuena and Molina, 1996) – a knowledge modelling system which creates concept map output from a knowledge base structure; and DESIRE (DEsign and Specification of Interacting REasoning components) (Jonker et al., 1998), a knowledge-base and multi-agent system development method using an automatic translator to convert between textual and graphical representations.

2.4 Ontologies in knowledge represention

Research on ontology is becoming increasingly widespread in the community of computer science, and its importance is being recognized in a multiplicity of research areas and application areas, including knowledge engineering, agent-based systems, database design and integration, information retrieval and extraction. Like knowledge representation methods discussed earlier, the purpose of ontologies is to provide a way of describing knowledge. However, ontology approach represents a slightly different perspective: while traditional knowledge representation formalisms focus on describing and solving problems, ontologies concentrate on the description of the world around these problems.

2.4.1 What is ontology?

The term "ontology" originates from philosophy, indicating a systematic, empirical and largely pragmatic account of existence. In philosophical sense, ontology is referred as a particular system of categiories accounting for a certain vision of the world (Aristotelean Ontology), like a classification of species by their genus.

Despite of some terminological confusion and debate, artificial intelligence community has adopted the use of term "ontology" as a way of differentiating between knowledge representation schemes and the content of knowledge representation. In general, knowledge representation addresses *how* something is described, while an ontology is used to express *what* exactly is described. In the tradition of AI, knowledge is defined with a focus on functionality and reasoning (comparable to the discipline of epistemology in the field of philosophical sciences, indicating the study of nature and sources of knowledge). Ontology, on the other hand, represents the modelling view of knowledge representation: in addition to modelling problems and their solutions, ontologies are used for modelling *context* of problems and problem-solving (comparable to traditional philosophical Aristotelean Ontology, i.e. what exists and what is the nature of existence).

In enunciated words, ontology is a shared understanding of terminology of some domain of interest, based on *conceptualisation* of domain entities and relations between them. In their classic textbook of artificial intelligence, Genesereth and Nilsson (1987) define conceptualisation as:

> "A body of formally represented knowledge is based on a conceptualisation: the objects, concepts, and other entities that are assumed to exist in some area of interest and the relationships that hold among them".

Gruber (1993) extends this definition and indicates the meaning of ontology:

> "A conceptualisation is an abstract, simplified view of the world that we wish to represent for some purpose. Every knowledge base, knowledge-based system, or knowledge-level agent is committed to some conceptualisation, explicitly or implicitly. Ontology is an explicit specification of a conceptualisation."

In other words, ontology is a description of a domain of discourse in terms of some common representional vocabulary. By explicit commitment Gruber means that each entity in the ontology is well-defined in terms of this vocabulary. An ontology is said to *commit* to a specific conceptualisation if there exists a language (not necessarily a part of a logical language: for example, it may be a protocol used for communication between agents) to describe the ontology. Likewise, a language commits to certain ontology if it agrees with the ontology about the conceptualisation. It should be noted that this definition makes ontology language-dependent, while conceptualisation itself is language-independent.

An informal ontology may be specified by a catalog of types that are either undefined or defined only by statements in a natural language. A formal ontology refers to a particular engineering artifact constituted to describe a certain domain, and a specific set of vocabulary words with explicit intended meaning. This set of assumptions is often described in the form of first-order logic, where vocabulary words appear as unary (concepts) or binary (relations) predicate names (Guarino, 1998a). Thus, ontology can be viewed as a collection of concepts, their definitions, and relationships between them like e.g. inheritance and aggregation. This definition bears great resemblence to conceptual graphs introduced in the previous chapter.

The basic building blocks of ontologies are *concepts* and *relations*. Concepts can represent either *types* or *roles* (Guarino, 1998a). The basic difference between them is that types are said to be semantically rigid i.e. their instances are such that they always belong to the type, while an instance's roles may change. For example, person's gender is a type, because it cannot change during the lifetime of an individual. On the other hand, student is a individual's role as he ceases to be a student when he graduates. Under certain circumstances, a concept may be used to represent an *instance* as well (see the discussion of top-level, domain, and application ontologies below).

Like in the theory of conceptual graphs, ontological *relations* are used to connect one or more concepts to each other. There are no restrictions concerning the arity of these relations, but in general, unary and binary relations are adequate enough to build ontology. If desireable, relations of greater arity can be expressed simply by means of additional concept representing the relation itself.

The key difference between taxonomies and ontologies is that while they both describe a structure of knowledge using concepts and relations, taxonomies represent only one perspective, namely that of classification (e.g. inheritance). Ontologies, on the contrary, provide a richer representation of concepts and relations that does not restrict the structure of knowledge to a classification hierarchy, or in any other way: concepts in the ontology can be related by any kind and any number of different relations. However, it might be beneficial to organise ontologies by using *viewpoints*, selections of concepts and their interrelations that represent a certain point of view (for example, a taxonomy by inheritance. In this sense, taxonomies can be regarded as certain

viewpoints in ontology). For example, genes in biology can be classified differently from functional, chemical, and evolutionary viewpoints (Euzenat, 1993).

Organising ontologies by viewpoints requires the same concept to be available in multitude of viewpoints. This is accomplished by using *bridges*, special kind of relations that indicate that related concepts represent the same entity in the ontology. This allows the same concept to be viewed in different contexts, and different contexts to be combined to form deduced knowledge. Embodiment of a concept in different viewpoint is sometimes called a *shadow object*, like in the ABS project (see Chapter 4).

The idea can be taken further by allowing bridges to span the boundaries of ontologies. Use of bridge allows the same concept to exist in various ontologies, without forfeiting its integrity and indentity. A (partial) mapping of concepts and relations between two ontologies is called *alignment* (Sowa, 1997) of ontologies. It should be noted that alignment does not depend on the choice of names in either ontology. For example, an alignment of a Japanese ontology to an English ontology might map the Japanese term for number five "Go" to the English concept "Five" instead of the English verb "Go".

An example an ontology can be seen in Figure 2-5, a fragment of the ontological structure of SARI system (Taveter, 1998).

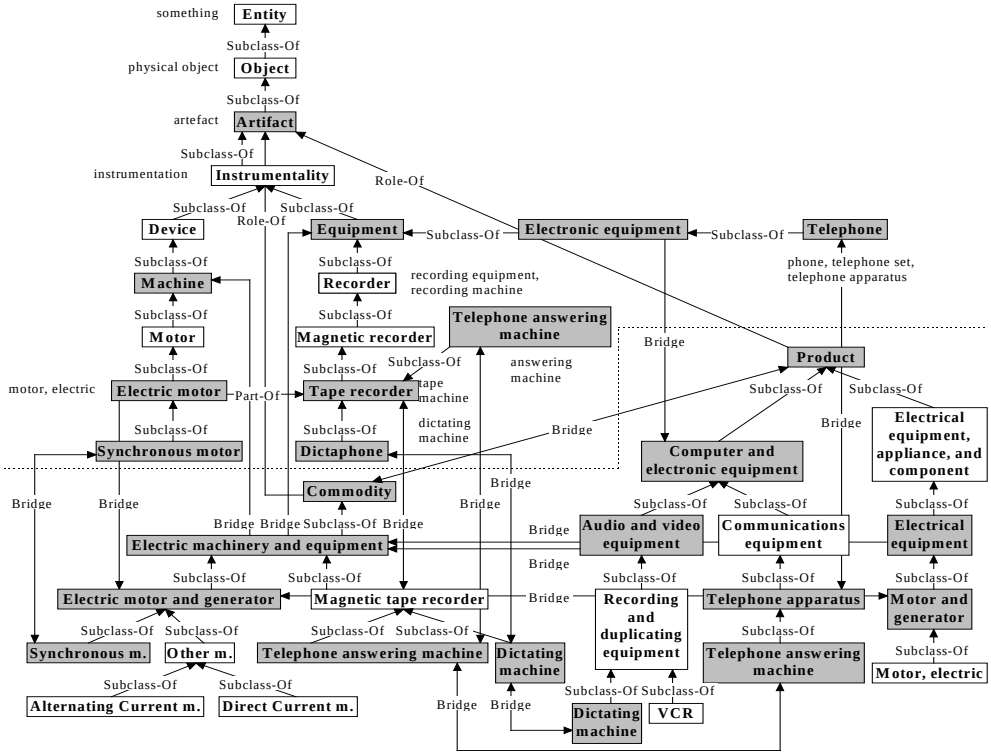


Figure 2-5

The major application area for ontologies is the integration of information, combining existing ontologies into new ones. If ontologies to be integrated share the same conceptualisation, they can be easily aligned by forming bridges between concepts in different ontologies. The situation is more complex if each ontology has its own conceptualisation, even if they would share a common vocabulary. There is no guarantee that they can agree on some information unless they explicitly commit to the same conceptualisation . In order to integrate ontologies, an agreement of mapping

between two different conceptualisations must be made. This kind of mapping is very difficult to create and manage, at least when more than two ontologies are being tried to be integrated.

According to Guarino (1998a), it is more convenient to agree on a single top-level ontology instead of relying on agreements based on the intersection of different ontologies (see Figure 2-6). The same considerations suggest the opportunity to develop different kinds of ontologies according to their generality, and to define new ontologies in terms of existing, higher-level ontologies.

- *Top-level ontologies* describe generic concepts that are independent of a particular problem or domain, such as space, time, matter, object, event, action, etc.

- *Domain ontologies* and *task ontologies* describe concepts related to a generic domain (like medicine, or forest industry) or a generic task or activity (like diagnosing or selling), by means of terms specified in the top-level ontology.

- *Application ontologies* describe concepts depending both on particular domain and task, which are often specializations of the corresponding ontologies. These concepts often correspond to roles played by domain entities, like "student" or "spare component". If the problem domain described by the application ontology is a detailed one, it may contain concepts representing specific *individuals* of the domain. For example, an ontology describing an enterprise may include concepts representing specific departments, business premises, etc.

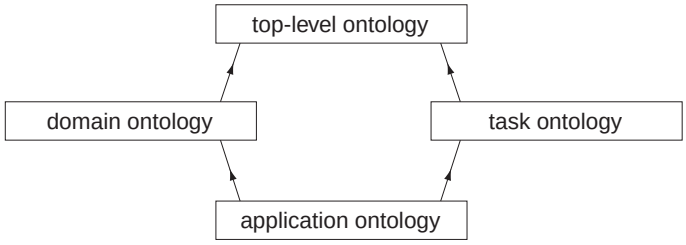


Figure 2-6

Adopting this kind of classification enables reusability and supports evolution of ontologies. Building ontologies based on well-known shareable base ontologies has several benefits. Describing a problem domain, even a limited one, with appropriate conceptualisations is usually a huge amount of work — so extensive that it shouldn't ever have to be replicated. If conceptual analysis is made carefully considering genericity and as many perspectives as possible, the results of this analysis can be encoded into a re-usable high-level ontology. This ontology can be shared with others who have similar needs for knowledge representation in that domain, saving a significant amount of labor in conceptual analysis. The consistency of information is guaranteed and changes or additions to upper level ontologies are automatically reflected in lower level definitions.

Any implemented ontology is just a starting point, which certainly is to be developed in the future. It is impossible to see all of the ramifications of the initial set of agreements, and this is why it should be possible to extend the ontology to cover new areas as they arise. By dividing knowledge into a hierarchy of modular ontologies, techniques for

extending and overriding existing domain descriptions can be defined. The ultimate goal of this approach is to build a library of ontologies which can be reused and easily adapted to different uses and environments.

2.4.2 Role of ontology in knowledge representation

Artificial inteligence research on ontologies is relatively new territory. Absence of formal definitions on ontology and differences between the traditional philosophical interpretation and the more recent AI interpretation has caused some misunderstanding. In some cases, the term "ontology" has been used just as a new name denoting the result of familiar activities like conceptual analysis and domain modelling, carried out by means of standard methodologies. Artificial intelligence intepretation of ontology is often obscured by the philosophical background of the term, causing ontologies to be equated with taxonomic hierarchies of types.

Much of the work on ontologies has focused on describing some aspect of reality: objects, relations, states of affairs, events, and processes in the world. As mentioned in the previous chapter, the current consensus is that the key difference of ontologies and other, so called "traditional" knowledge representation methods is that ontologies approach knowledge from the modelling point of view. Ontologies are not built with some specific domain or problem-solving process in mind, but as re-usable models that provide a well-defined basis for different problem solving tasks. The purpose of ontology is to be task- and problem-independent descriptions of entities, relations, functionality and events of the domain of discourse. Ontologies are quintessentially content theories.

The situation doesn't necessarily have to be that twofold. Rather than being completely separable from the traditional problem-specific approach to knowledge representation, ontologies can be seen as more extensive foundation for knowledge encoding that may enclose problem-specific knowledge as well. Chandrasekaran et al. (1998) propose ontologies of tasks and methods that contain knowledge of various problem-solving tasks, both in generic and in task-specific forms. If problem-solving knowledge is described in terms of entities defined in top ontologies, it could be shared and combined to form new knowledge just as easily as other content knowledge.

Due to the similarities between ontologies and the results of database design methodologies, such as ER modelling, database schemas are often incorrectly referred as ontologies. According to them, ontologies are like conceptual schemas in database systems; a database schema provides a description of shared data, allowing applications and other databases to interoperate with it without having to share data structures. However, these observations of "databases as ontologies" are misleading. Defining a database schema does not yet commit to any specific conceptualisation. Database entities and their interrelation provide a model of the domain in question, but do not give any indication about the meaning of the model objects. Entities external to the database can see only model objects, but have no way of associating them with their own correspondents. The situation is different if there are multiple databases that commit to certain ontology. Mahalingam and Huhns (1998) observe the similarities between ontologies and traditional Object-Oriented (OO) models and propose the use of ontologies as an extension to ER model to organise database concepts among tables and fields and to perform queries in relational databases. Using the terminology agreed in the common ontology, multiple distributed heterogeneous databases can be queried for information in parallel.

2.4.3 Ontology representation and markup

Basically, the theory of ontologies does not say anything about the markup or physical representation of the ontology: an *informal ontology* may be a description of the domain entities in form of written or even spoken natural language sentences. A more exact approach is to describe the ontology using some formal method, such as KIF, predicate logic, or conceptual graphs. This kind of ontology is referred as *formal ontology*.

Currently, there is no standard method or even common practice of formally encoding ontologies. Existing knowledge representation schemes have been adopted and new ones developed for this purpose. Ontology systems typically use their own private formats, and translators are built for communication and information exchange. Ontolingua (Gruber, 1993) was designed for this purpose. Its goal is to support the design and specification of ontologies in a form that is compatible with multiple representation languages, and to allow ontologies built in different projects and using different tools to be exchanged fluently. The language is based on extended form of Knowledge Interchange Format (KIF) (Genesareth and Fikes, 1992) to provide additional idioms frequently appearing in ontologies. It provides forms to describe classes, relations, functions, objects, etc. and is able to translate definitions based on these forms to input formats of variety of already implemented representation systems. However, Ontolingua is domain-independent translation tool, not a generic representation language for ontologies.

Ontology Markup Language (OML) (Kent, 1998) is a step towards a generic ontology markup format that can be easily exchanged in the distributed environment such as the Internet. It is a proposal of a framework for representing predicate logic using an object model and a language based on Extensible Markup Language (XML) (Bray et al., 1998). The language is currently under development, and four different versions of the language definition are under consideration. *Standard OML* is the basic form of OML, providing support for defining and extending ontologies in a distributed fashion. *Abbreviated OML* is abbreviated form of the standard form, providing a more compact format with some of the expressive features of standard OML left out. *Simple OML* is provided for compatibility with RDF/Schemas standard proposed by W3 Consortium (Brickley and Guha, 1999). Finally, *Core OML* is a version of standard OML which goal is to find a definition that is minimalist but logically equivalent in expressive power as the standard OML.

Combining experiences obtained from various existing knowledge representation schemes with a powerful XML standard, OML sounds like a promising candidate for representing ontologies in a formal way . However, it is still very young proposal and is continuously under very heavy changes. How it will succeed remains yet to be seen.

2.4.4 Issues in ontology

Although the benefits of ontologies are widely understood and the possibilities provided by large scale knowledge repositories are realised, it is fair to say that ontologies have not been used and the hopes for them have not been realised. The reasons for this are manyfold, but in general, lack of standard methods for encoding knowledge in sharable and easily extensible format has hindered the development of common top-level ontologies. Research projects have been mostly working with their own project-specific tools and methods, and efforts for creating foundations for

knowledge sharing in form of ontologies have been few. This far, creating an ontology has meant starting from a scratch.

The same fact has had its reflection on the economic side. Constructing ontologies is difficult and very time consuming, and requires hours of work of highly skilled professionals. The job is difficult to get done right at the first time, and often requires experimentation. Ontology development costs are very high, which in its own part has been a major barrier to building large scale intelligent systems based on ontologies.

Constructing and managing an ontology also involves many practical problems. The core of an ontology consists often of huge amounts of source code in a particular knowledge representation language. Without appropriate tools, the ontology is very difficult to understand and take into use. Tools designed specially with large ontologies in mind help users to browse the ontology to undestand its scope, structure and content. Search tools help users to find the terms of interest and things related to them. Swartout et al. (1996) see the lack of appropriate support tools for browsing ontologies as one of the main reasons for their slow adoption.

2.5 Applications for ontologies

The introduction of ontologies has been motivated by several factors, the primary one being the ability to enable the communication and the sharing of knowledge. In general, the goal of ontologies is to integrate models of different domains into a coherent framework in order to distribute information amongst its users. This need arises in modelling (for example, business process reengineering where an integrated model of the enterprise, its organisations, processes, goals and customers, is required), concurrent engineering and design, as well as in distributed multiagent architectures. By using ontologies, the information concerning the domain under discourse can be described in a manner that is able to serve a wide variety of tools and users.

An important direction in the utilisation of ontologies is to use them as knowledge representation mechanism for information retrieval purposes. In many situations, there is a lot of heterogenous information available, but finding the relevant fact is very difficult due to the large amount or the format of information. Various search mechanisms have been developed, but they are often unable to function in a desired way as there is no description of the actual content of the information. A prime example of this problem can be seen in the Internet search facilities.

A motivation for building an ontology may also arise from a need of collecting expertise into well-organised memory banks, to be utilised by intelligent applications as well as the people. This ontology can be used as an information source for problem solving, and its content may be accumulated as the expertise grows and new experiences are gained. This idea of "organisational memory" is especially important in corporate environments, where expertise is often tied to the personnel, and the experiences of the lessons learned are very valuable information.

2.5.1 Intelligent agents and communication

In order to be able to communicate, the basic requirement between agents is that they understand each other. An obvious use for ontologies is to use them as means of providing this kind of understanding. Agents communicate to each other via messages that contain expressions formulated in terms of an ontology. To understand the

meaning of these messages, the agents must have an access to the ontology they commit to, and to use this ontology to interpret messages. Guarino (1998a) refers this kind of information systems as *ontology-driven information systems*, systems that use ontologies to obtain some "higher" overall IS goal (another category of IS consists of so called *ontology-aware information systems*, systems that are aware of the existence of an ontology and use its vocabulary for some purpose, for example to formulate database queries to be performed in multiple heterogenous databases).

To enable this kind of knowledge-level communication, conventions at three different levels must be considered: representation language format, agent communication protocol, and specification of the content of shared knowledge (Gruber, 1993). Knowledge representation formats, such as KIF, and agent communication languages, such as KQML (Finin et al., 1992), are independent of the content of the knowledge being communicated. In a form of content descriptions, ontologies can be used for the conventions of the third kind. A common vocabulary of domain entities can be used to map different user models to a format understood by all parties. This format provided by the ontology is a language in which queries and assertions are exchanged between agents. Other approach is to design unique translator for communication between every two parties; however, this requires support for $O(n^2)$ translators for n different languages supported by users. Using ontologies as inter-lingua to support translation, only $O(n)$ translators are needed, as displayed in Figure 2-7 (Uschold and Gruninger, 1996).

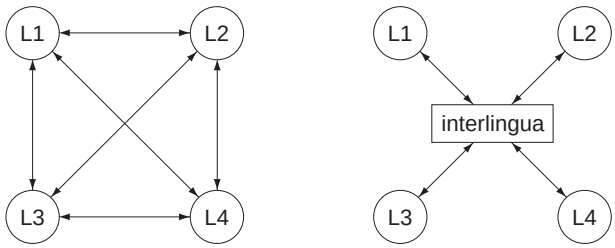


Figure 2-7

2.5.2 Information retrieval on the Internet

Since the introduction of world wide web (WWW) in mid 1990's, millions of web pages have been accumulated into this enormous web of hyperlinked information. Rapid growth has brought large numbers of new and existing documents online, and web as an information source has become very popular. Masses of documents can be searched in couple of seconds, and the results are often of high quality and from relatively reliable sources. However, the problems involved in Internet information retrieval are easy to see. Searching by using too generic key causes the search to fail miserably by resulting thousands of documents, which with a closer look seem to have nothing in common with the topic searched.

The cause of this kind of a behavior is the wrong approach to content-matching in majority of search engines used today. Current retrieval tools are inadequate because they try to match documents based on the information meant for human consumption (i.e. text in HTML page content), not the actual content of the page. The techniques currently adopted either rely on an encoding process describing documents according to certain perspective, or are based on full-text analysis and indexing of pages, and

search for a keyword or sentence specified by the user. In either of cases, correct content-matching is not guaranteed. The description analysed may reflect only part of the content (only text is analysed, pictures, graphs, etc. are ignored), and the occurrence of a particular word on a web page does not necessarily reflect the content of the document.

Introduction of ontologies for information retrieval can help to solve these problems. By expressing the web page content in terms of concepts and relations published by some common ontology, the page authors may describe their pages in form that is language independent and searchable using ontology-based content matching process. The content description may be embedded inside the web page in a machine-readable format that can be utilised by various agents and query engines.

SHOE (Simple HTML Ontological Extensions) (Heflin et al., 1996) is a knowledge representation language that allows ontologies to be designed and used directly on the World Wide Web. Using SHOE, web pages may declare ontologies to define rules of what kind of assertions may be made and what kinds of inferences may be drawn on them, or instances which make assertions based on ontologies. SHOE syntax is an extension to HTML, so statements can be embedded directly inside the web page. To apply SHOE in practice, a set of support tools have been developed: SHOE library is capable of parsing and manipulating SHOE declarations, and Knowledge Annotator is a tool for adding SHOE declarations into web pages. Éxpose robot application is used in looking for pages with SHOE markup and store parsed content representations into a knowledge base for search and inference processes. Same kind of approach have been adopted in OntoSeek project by Guarino et al.(Guarino, 1998b).

2.5.3 Enterprise ontologies

Enterprise modelling is concerned with the development of models of various aspects of an enterprise. Different models have evolved to model different activities: data models to model the information managed by the enterprise, process models to model various processes occuring in the enterprise, activity models to describe different business activities performed within and by the enterprise, and so on. By combining these various models, a better understanding of the enterprise and its actions can be achieved. Origins of the problems faced by the enterprise may be tracked down more easily, enterprise's operations can be more easily monitored and controlled, and alternative problem solutions may be tried out by simulation. The integration of these models is the description of the enterprise and its operation.

The overall goal of enterprise modelling is to take an enterprise-wide view of an organisation which can then be used as a basis for taking decisions. The results can be seen as improved business planning, greater flexibility and adaptability to changes in the business environment, more effective communication within the organisation, and better integration and interoperability of tools and practices used in the enterprise.

The Enterprise Ontology (Uschold et al., 1997) is a collection of terms and definitions relevant to business enterprises, to be used as a basis of building enterprise ontologies. It includes a wide variety of carefully defined concepts which are widely used for describing enterprises in general. The idea is not to force different enterprises to conform to a same pattern, but rather to provide one set of definitions which adequately and accurately covers the relevant concepts in the enterprise modelling domain, to be modified and adjusted to each particular business' conditions.

Use of ontology to describe, for example, a business or organisational knowledge may consequentially enhance communication between people in the organisation. Normative model provided by the ontology allows its users to speak the "same language", which is not obscured by ambiguities caused by different perspectives, assumptions or opinions. This practice has also a conductive effect on development of standards within a community, as all participants use standardised terminology for domain objects and relations.

2.5.4 Ontology and information systems

A shared understanding of the task at hand can also assist in the specification of software systems. Guarino (1998a) suggests the idea of using ontologies in information systems (IS), both in the development time and run time.

During the development, the semantic content expressed by appropriate ontology in ontological library can be transformed into an IS, reducing the amount of work required for conceptual analysis. An informal approach would utilise knowledge stored in ontology to facilitate the process of identifying the requirements of the system and understanding the relationships among the components of the system, while more formal approach would use ontology as a declarative specification of a software system. By using this method, application domain knowledge can be more easily shared across different projects and heterogenous platforms in form of common vocabulary, and the use of ontologies allows domain knowledge to be reused in a manner that provides substantially higher level of reuse than software reuse normally practised in software engineering. Even if the quantity of ontological knowledge is too modest to be directly adapted as a basis of information system, it could present developers a very powerful tool that can help in many ways during requirement and conceptual analysis phases.

At run time, the most obvious use for ontologies is in connection with the database component. The availability of ontologies describing information resources is at the core of mediation-based approach to information integration. By using them as generic descriptions of the domain to be queried, ontologies can be used to support dynamic construction of queries that are targeted to multiple heterogenous databases.

Another, perhaps not so obvious, run-time use for ontologies is their utilisation in conjuction with user interfaces. Since ontologies embody the semantic information of the constraints and relations of a domain or a task, they can be used to automatically create form-based interfaces and to check for constraints violation in them. In case the user is aware of the existence of ontology in IS, he may also be presented with a view to it in order to be queried, browsed or manipulated.

3 Tool support for ontological engineering

Describing a problem domain in a form of an ontology is difficult and hard work. Domains to be represented can be very large, or the amount of details contained can be overhelming. Figuring out the relevant concepts and their interrelations requires thorough knowledge of the domain, as well as experience in knowledge engineering.

Even if difficulties involved in conceptualisation and selection of correct formalism were entirely omitted, there are still major barriers to be crossed over. Most of them are simply practical: how the user can input concepts and relations; in what way the contents of ontology should be displayed on the computer screen so that the user can understand what he sees; how can he cope with extensive amounts of information; how the existing information is used, and how new information is accumulated? It is clear that the goal of ontological engineering, modular, shareable and reusable ontologies, cannot be reached without answering these questions and providing proper tools designed with these problems in mind.

3.1 Background

The problems presented above have been realised and some research projects have already started solving them. Several different tool applications have been created, each of which has its own perspective to the process of ontological engineering. They vary greatly in features and details, and perhaps have different conception of ontological engineering. However, they serve their purposes relatively well and are good sources of ideas when considering new tool application for ontological engineering.

As a starting point of this study, a survey of existing ontology editors was performed. A total of six applications were selected for closer inspection and evaluated to get a more definitive picture of problems, requirements and existing solutions involved. The focus of this survey was on following issues:

- *Design model*. What are the key entities used to represent knowledge, and how are they organised? How are these entities related to each other and what is the motivation for them?

- *Graphical representation*. How is the knowledge represented? Which objects of the design model are visible, and how are they displayed? Does the application employ some kind of visual methods to represent types, instances, exceptions, etc.?

- *Information management*. How does the application cope with large amounts of data? Are there methods for abstraction or viewing only parts of the model?

- *User interaction*. How does the user interact with the model?

- *Editing*. Does the application provide any tools to accumulate the knowledge?

The survey was based on two primary information sources: scientific articles (or marketing material) discussing the tools and their application areas, and the test use performed by the author of this thesis. Most of the tools or their demo versions were freely available for test use, and could be used to browse pre-built knowledge bases or to create new ones. Others (GKB-Editor and GrIT) were evaluated based on the description of their features.

The goal of this survey was not to rank these applications, but to describe their features to help out forming the ideal set of features to be implemented in CONE project, and to prevent making the mistakes made by other tool developers. The following is a brief introduction to each tool in survey, including a short commentary of their benefits and drawbacks.

3.1.1 CODE4

CODE4 (Conceptually Oriented Description Environment) is a general-purpose knowledge management tool developed in Artificial Intelligence Laboratory of University of Ottawa (Skuce, 1995). It can be used for analyzing, storing and retrieving conceptual knowledge about some domain and its main objectives are learnability and adaptability for various applications, such as natural language processing and expert systems.

Knowledge representation formalism used in CODE4 is called CODE4-KR. It is based on ideas adopted from frame-based inheritance systems, conceptual graphs, object-orientation and description logic systems. Motivation for CODE4-KR is increased expressiveness over the ability to perform complex inferencing, and suitability for non-AI specialist use.

CODE4 treats knowledge as a collection of *concepts*, each of which represents a "thing". A concept represents either the collective idea of a collection of similar things (such as "car"), or a one particular thing (such as an instance of a "car", "my red Honda"). Concepts may be abstract or concrete, real or imagined. They may represent actions, states or in general, anything one can think about. Concepts can contain other concepts as *properties*, units of knowledge that are inherited by other concepts. Properties are connected to concepts via statements, which are basically yet another primitive concept in CODE4 knowledge representation.

The underlying ideas of CODE4 user interface are *knowledge maps* and *browsers*. Knowledge map is a software abstraction that defines a network in terms of a starting concept and relations that connect it to other concepts. The user interface of CODE4 consists of a set of browsers. In order to display definitions of the ontology the user has to select a knowledge map and browser type: outline browsers, graphical browsers and matrix browsers are the basic browser types supported. Each browser type has a different view to knowledge structures.

- *Outline browser* displays the hierarchies as indented lists. Each line of text represents a concept and its subconcepts are displayed on lines below it. Typical outline processor commands can be applied to the hierarchy: nodes can be collapsed and expanded to display only the desired information.

- *Graphical browser* displays the hierarchic knowledge map as a directed graph with concepts being represented by nodes with different shapes and relations between them as connected arcs. The user may work directly on a graph, adding, deleting, forming connections etc.

- *Property matrix browser* is used to display concept properties. A group of concepts can be opened to same property matrix browser, which makes it very easy to compare several concepts, to see how they differ.

In addition to browsing tools described above, CODE4 provides another tool for managing the knowledge map: a notion of masks. A mask is a set of conditions that is applied to all concepts in the knowledge map when considered for display. Evaluating the mask condition returns a truth value that determines whether the concept is displayed or not. The masks are useful for several purposes, most notably to focus the display and to reduce screen clutter, or to perform database-like retrieval.

Clearly, CODE4's main innovations are consistent set of tools, different browser types, and the powerful mask tool. Different views (hierarchical, graphical, property matrix) to same knowledge source give users a chance to always select the best tool suited for a task at hand. Masks may be applied to display different perspectives on the information, or to reduce the amount of visible information.

3.1.2 IKARUS

IKARUS (Intelligent Knowledge Acquisition and Retrieval Universal System) is a web-based successor for CODE4 knowledge management environment. The main principle behind IKARUS is to combine traditional knowledge management to cooperative capabilities of World Wide Web. In IKARUS, any user, anywhere in the world, can access the knowledge base and add to it.

To organise knowledge in the ontology, IKARUS uses a hierarchical representation that resembles classes in object-oriented programming or frames in traditional AI systems. Frames store knowledge about the *subject* in form of one or more *statements*, and they are organised hierarchically. Each subject may have multiple parents and any number of children, and they can be used to represent types (classes) as well as instances of a particular type. Statements contain the information about subjects either as *predicates* with well-defined syntax and semantics, or as unstructured fragments of information (such as text, URLs to web documents etc.).

IKARUS user interface consists of the two different views to the knowledge base: HTML and WWW browser based interface for traversing through the hierarchies, and a Java-based graphical presentation of the user-selectable part of the hierarchy. The top part of the IKARUS main window (Figure 3-1) contains the control panel that allows the user to select starting parameters for browsing. The left part contains the hierarchical (or alphabetic) list of all subjects in the knowledge base. Selecting a subject link displays the subject in the right part of the window. Hierarchy Viewer Window can be used to display a graphical representation of the subject is-a hierarchy. Subjects are displayed as nodes of the graph and are related by directed arrows to reveal multiple inheritance relations of subjects.

A flexible coloring scheme is employed to ease viewing and understanding complex internode relations. Colors are used to indicate specific concept types, or to make inheritance relations more easily observable. For example, all parent concepts of the currently selected concept are highlighted with red color as the selection changes. Based on HTTP protocol, HTML pages and request-response nature of its operation, the user interface of IKARUS knowledge management system is limited. Editing knowledge bases is cumbersome via HTML form submitting process. No graphical or direct manipulation tools are provided for editing, neither is textual representation available.

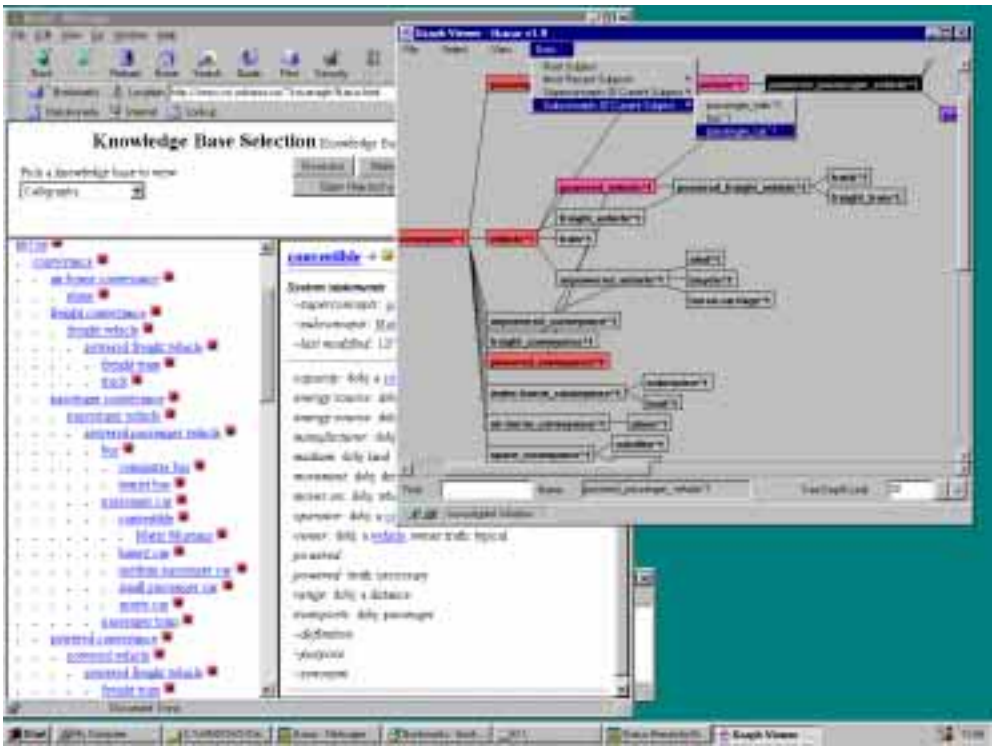


Figure 3-1

3.1.3 JOE – Java Ontology Editor

Java Ontology Editor (JOE) is an ontology construction and viewing tool developed in Center for Information Technology, University of South Carolina (Mahalingam and Huhns, 1998). The basic idea behind JOE is to provide a knowledge management tool that supports multiple simultaneous users and distributed, heterogeneous operating environments. Basically, JOE is the graphical viewer/editor that allows the ontology to be viewed in different formats and with different accuracy.

Ontologies, from JOE's point of view, are Entity-Relationship (ER) or Frame-Slot models of a given knowledge base. The ontology can be viewed in three different formats: as an ER diagram, as a hierarchy similar to Microsoft Windows file manager, or as a graphical tree structure.

JOE application consists of two different user interface components: ontology viewer and the query generator. The viewer can be used not only to browse ontologies but also to create and modify them: using editor dialogs, new *entities* (concepts) can be added and old ones removed, and entity *attributes* defined. New *relations* can be added to connect two or more entities. To ease navigation among a large number of concepts, following features are available in JOE:

- *Selective viewing*. The display method can be controlled by the user to show only desired information. The ontology can be viewed in three modes: entities only, entities and relations, or entities and attributes.

- *Searching*. Concepts of the currently visible ontology can easily be searched using their names.

- *Zooming and magnification*. The ontology can be viewed at different zoom levels, from entire ontology to a detail view. When entire ontology is viewed, a special magnification window can be opened to display a magnification of a selected part of the view.

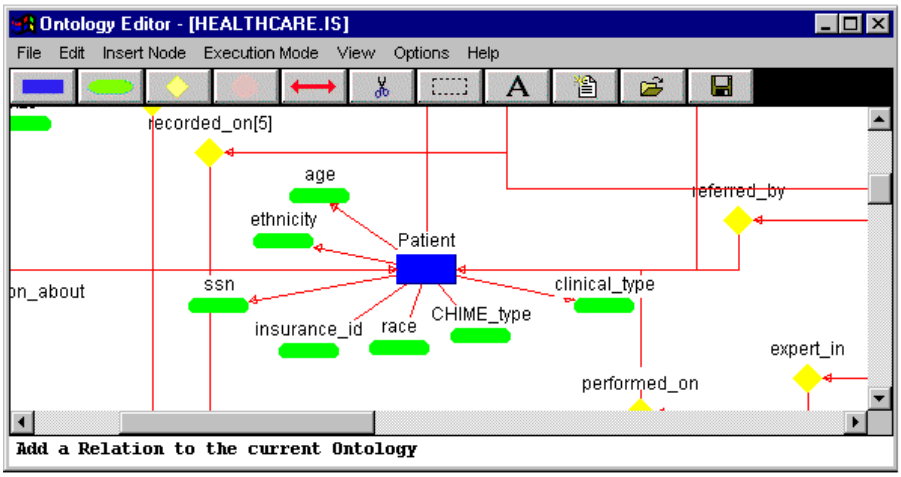


Figure 3-2

In addition to various viewing features, JOE supports most of the basic editing functionality such as selecting, cutting and moving. JOE editor toolbar buttons (see Figure 3-2) is used to insert new entities, attributes and relations. The same editing view is used when querying. Queries are formulated by using the visual representation of the ontology and a point-and-click approach of adding query conditions. Clicking relations and/or attributes displays dialogs to set values for query parameters and adds them to the current query. After all conditions are added, the query is transformed into SQL statement and submitted to the server for processing.

JOE is a complete and well-thought application for managing and querying ontologies. JOE's features are at their best when viewing ontologies: using the selective viewing option objects in the ontology can be effortlessly located. Zooming combined with magnification allows the user to see the entire ontology at once without losing any details. An intuitive feature of JOE's is the way how queries are generated: a simple point-and-click approach enables users with lesser skills in SQL to perform queries using the ontology and provides an easy way to understand the connection between ER diagrams and SQL statements.

The main problems of JOE are its inflexible editing tools and the lack of large ontology support. Usage of modal tools to add, remove, select, name and move objects in the ontology is clumsy and does not work well. When the size of the ontology increases, the graph becomes very cluttered and screen updates are very slow.

3.1.4 Santiago/VT

Santiago/VT is a part of a Santiago Knowledge Management System from Peirce Holdings International Pty. Ltd. The knowledge representation format used by Santiago/VT is Conceptual Graphs.

Santiago Knowledge Management System consists of two accompanying products: Santiago/KR (Knowledge Retrieval) which is based on experiences of company's

previous product Peirce, and Santiago/VT, a tool that provides a graphical user interface for the Santiago knowledge retrieval system. The browser enables the user to visualise sets of tree-structured conceptual graphs, and provides a text-based interface for defining new graphs.

1. (exists (?p) (and (person ?p) (forall (?t) (and (time ?t) (fool ?p ?t)))))
2. (forall (?p) (and (person ?p) (exists (?t) (and (time ?t) (fool ?p ?t)))))
3. (not (forall (?p) (and (person ?p) (forall (?t) (and (time ?t) (fool ?p ?t))))))

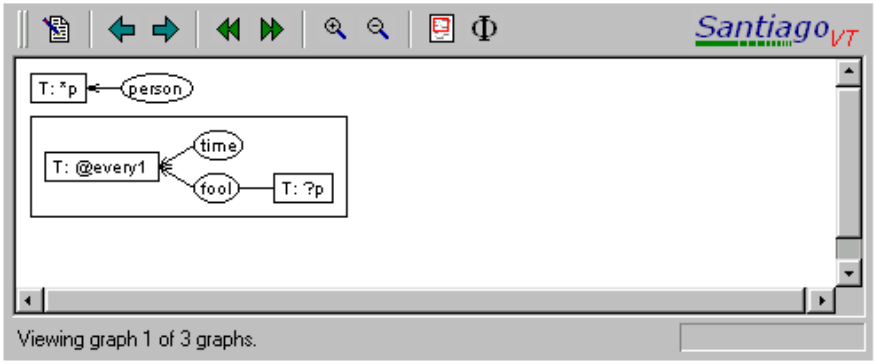


Figure 3-3

The Santiago/VT browser interface is displayed in Figure 3-3. The application is implemented as a Java applet, so it can easily be embedded in HTML page. The conceptual graph is displayed in scrollable portion of the window. The toolbar contains simple tools for navigating the set of statements. The view can be zoomed, and an overview map displaying the entire graph as a miniature can be opened.

Graph is edited using a separate editor window, which displays a textual representation of the graph in Santiago's BCG (Basic Conceptual Graphs) syntax. This is the biggest shortcoming of Santiago/VT. Forming relations, adding and deleting concepts is difficult using only a text editor and thus managing a large knowledge base using Santiago/VT is practically impossible.

3.1.5 GKB-Editor

The GKB-Editor (Generic Knowledge Base Editor) is a tool for browsing and editing knowledge bases across multiple Frame Representation Systems (FRSs) in a uniform manner. It offers an intuitive, graph-based user interface, in which users can edit a knowledge base through direct manipulation and selectively display the region of a knowledge base that is currently of interest. GKB-Editor has been developed at Artificial Intelligence Center (AIC) in SRI (Stanford Research Institute) International.

GKB-Editor consists of four different interactive browsers. For this study, the most interesting one is the class hierarchy viewer. Other views are modifications of it or provide a non-visual view to the knowledge base.

The class hierarchy viewer (Figure 3-4) can be used to inspect class-instance taxonomy. The class hierarchy is drawn as a tree-structured graph in which each frame is represented by a node of the graph, and directed edges are drawn from classes to subclasses or instances. Multiple inheritance is shown by drawing edges from each superclass. Classes are distinguished from instances by color codes: classes are drawn in red, and instances in blue.

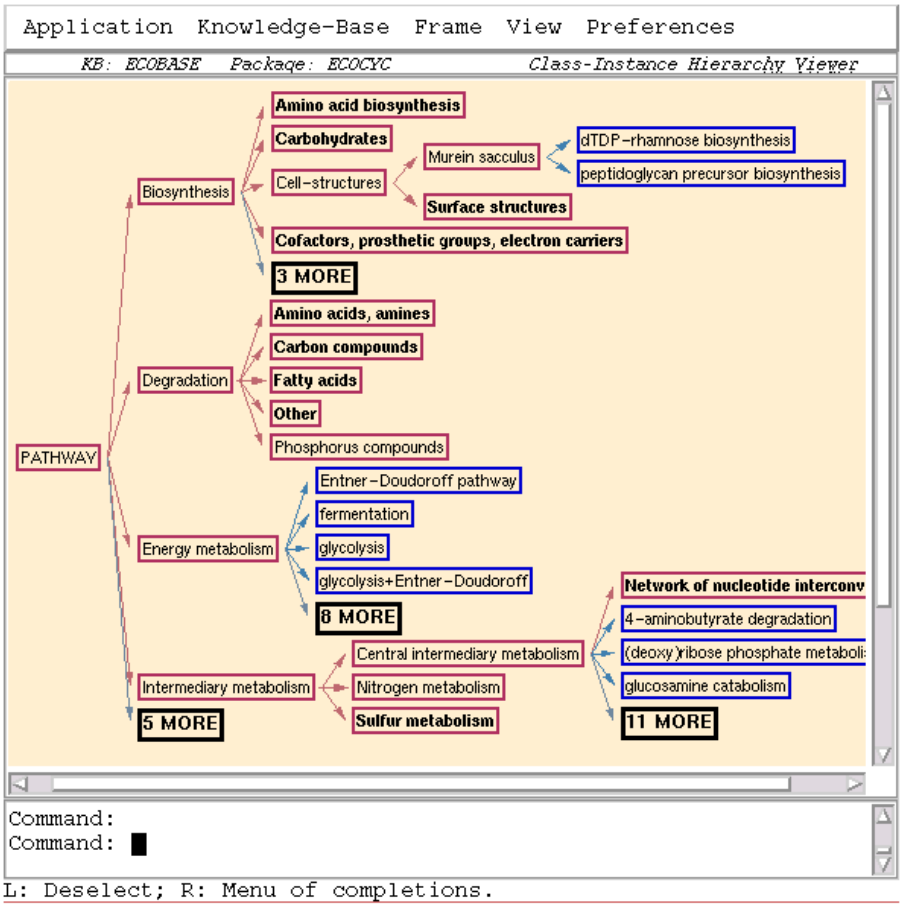


Figure 3-4

The class hierarchy viewer facilitates browsing by introducing features called incremental browsing and incremental expansion. Incremental browsing allows the user to open any node as a root of a new display hierarchy and view only a part of the class-instance hierarchy. Visual clutter is reduced by incremental expansion feature which automatically hides complex subtrees (collapsed node is called depth cutoff node and it is indicated by bold face type in node name) Also, multiple descendants of a node can be reduced to a breadth cutoff node (indicated by a bold face label "n MORE" in node name). By clicking a depth or breadth cutoff node with a mouse, an entire subtree or any additional siblings can easily be revealed. Editing class hierarchy (add a new parent, change parent, add a new class or instance etc.) is done via menu commands.

GKB-Editor is characterised by its complete set of graphical browsing tools. GKB-Editor allows the user to view the knowledge base from various viewpoints and display only desired portions of the knowledge base at time. Incremental browsing allows the user to control what is displayed, and in what level of detail. Unnecessary information can be hidden with a single click of mouse.

3.1.6 GrIT

The aim of the GRIP (Eklund, Leane, and Nowak, 1993) group was to produce a user interface for conceptual graphs that supports the graphical manipulation of conceptual graphs. The result of this project was GrIT: a software package for managing conceptual graphs.

GrIT provides a graph-based graphical representation and a set of basic tools for managing conceptual graphs: moving, deleting, resizing are all handled using the mouse by drag and drop. Capabilities of GrIT are quite minimal and it does not provide anything new compared to other applications in this survey. However, the GrIT paper (Eklund, Leane, and Novak, 1993) comes up with some interesting ideas concerning the usage of three-dimensional UI in conceptual graph interfaces. According to GRIP researchers, complex conceptual graphs could be displayed as a three-dimensional model on overlapping planes, each filtering information being passed to the topmost plane. Different planes, combined with color coding, could be reorganised by the user to get different perspective to the graph.

3.1.7 Summary

The survey presented above does not entirely cover the field of ontology support tools, but it provides a good number of ideas that can be used as a starting point for developing a new tool application. These applications have many advanced and powerful features, but what is needed is an integration of the best features to a single product. To summarise, the main innovations of these applications are:

- *Graphical representation*. Graphical and nonmodal direct manipulation user interface is a desirable way of displaying and editing conceptual networks and ontologies. The graph-based graphical representation of knowledge structures provides a natural and easily understandable metaphor, and is an excellent basis for various user interface features that help users to manage and navigate complex data.

- *Multiple formats*. Displaying the information in different formats simultaneously can help the user to understand the modelled information by showing the semantic connection between concept in different syntactic forms (for example, nodes in a graph and the corresponding KIF statement).

- *Selective viewing*. Providing the user with the ability to hide and show parts of the graph is a powerful way of managing graph layout. By hiding unnecessary portions of the model by collapsing subgraphs or subtrees, the chaos on the computer screen can be effectively eliminated, yet preserving the ability to display rich and detailed content. However, in order to not confuse the user, expansion state should be clearly visible, and the context should be preserved when the expand or collapse action is performed.

- *Abstraction methods*. Filters or masks can be useful when the user tries to find a particular piece of information from a large ontology. Filtering may be used to temporarily hide unnecessary information or to reduce the level of detail. Automatic filtering might be considered, but the user should be the final judge of what is displayed and in what accuracy. There should be an easy way to apply filters to the current view and the filtering should be indicated visually in the interface

- *Visual helper methods*. By using colors, concepts may be grouped to make detecting particular relationships easier, or to just divide concepts into categories. However, coloring schemes should be used carefully to prevent cognitional overload.

3.2 Support methods of ontological engineering

As shown by the summary presented in the previous chapter, introducing few relatively simple features can make working with ontologies significantly easier and more effective. Graphical representation for displaying ontologies and direct manipulation interaction style for navigating and editing ontologies are essential features. By defining ontologies in terms of discrete models and allowing linking between the models by using bridges, modularity and reusability of knowledge is enhanced and a special mechanism for inference is enabled. Furthermore, by allowing the user to dynamically change the visible part of the model to show only those parts currently relevant, both the user's cognitional load, as well as the computer's load of rendering complex networks, are significantly reduced.

Based on the survey of existing tools presented in the previous chapter, and the ideas aroused by the ABS and other VTT projects, a set of support methods for ontological engineering is proposed. The following chapters introduce these support methods and discuss each of them in detail.

3.2.1 Graphical representation and direct manipulation

Benefits of the graphical user interface (GUI) are widely known and GUI is by far the most common type of user interface in use today. Users are accustomed to WIMP (Windows, Icons, Mouse, Pointer) metaphor as a means of working with computers. Traditional command line interfaces (CLI) are disliked and not approved as a solution for everyday use of computers. However, the definition of GUI is vague: many applications use windows, menus, mouse interaction, etc. only to allow the user to work with application content information that is represented in textual or numeric form. In this setting, strengths of the graphical user interface are wasted and do not provide added value to help the user to accomplish his task — the application is not remarkably easier to use than correspoding CLI version of the same application would be.

More can be gained by visualising the data handled by the application in a form that helps understanding and managing it. Large datasets can possibly be displayed as graphs and diagrams, instead of simple numeric representation. Colors, patterns, shapes, etc. may be used to encode or classify items, or to draw user's attention to some specific detail. Three-dimensional graphics may be utilised to give different perspectives on the same data, or to manage level of detail in natural and easily understandable way (e.g. "to see more details, move closer!"). More information can be fitted on the computer screen while still preserving understandability and readability. This kind of supportive representation methods help users to find out the most relevant piece of information and to see such interrelations that otherwise would be very difficult to notice. Graphical representation is extremely important in situations in which information to be displayed is abstract or otherwise difficult to comprehend. This is the case when working with ontologies.

Graphical graph-based representation is a natural way of representing ontologies and conceptual networks. Displaying the conceptual network as a graph with nodes as concepts and arcs as relations reflects the user's mental model of knowledge — worlds of things connected to each other. Natural navigational skills, such as directional sense and cartographical memory can easily be utilised. Accumulation of knowledge can be implemented as natural, constructive process of broadening the graph by adding new nodes and arcs.

Errors that are caused by not considering the context of an action are easy to make when the context is not clearly visible. In graph representation the context is always visible and within the reach, while in other representations (for example, pure text or concept-relation lists) the connections to other concepts are obliterated by the sheer mass of visually almost identical objects (such as letters or words).

An important benefit of graphical representation is also that it reflects the complexity of the ontology as a complexity of graphical representation. Clusters of concepts and relations are displayed as groups of lines and graphical shapes, while unrelated concepts are shown as isolated nodes with few connecting arcs. Using graph representation, it is easy to see how the ontology is balanced, while in the other display methods the visual representation is homogenous and evenly distributed and not able to convey concepts of balance or focus. Zooming out can be applied to see the miniaturized version of the same data, or zooming in to see more details. Other representation methods like pure text cannot be scaled in a similar way, and are more vulnerable to the physical constraints of the computer screen: only a relatively small amount of text can be visible at the time, and seeing how much and what kind of information is left out is not possible.

Graphical representation of conceptual networks lends itself well to a direct manipulation interface because there are clear interconnections between concepts of application and user domains. Direct manipulation works especially well with graphical representations: it releases the control of the interaction to the user (Schneiderman, 1997). The user can use natural gestures such as clicking, dragging and dropping, drawing lines etc. to edit or navigate the conceptual model.

3.2.2 Large view management using clustering

Even though graphical representation of ontologies and conceptual networks has several benefits noted above, there are some practical problems that severely affect its usability. Any representation method is likely to fail when the amount of information increases massively, but for a graph-based representation rapid increase in the amount of information amount will propably paralyze both the user and the machine. Relations displayed as edges in the graph will create a complex web of lines, in which relation edges may cross each other. Some edges may connect concepts at very long distance from each other so no source, target or either one of the nodes are visible. Associations between concepts can no longer be made by a single glimpse, but by explicitly following relation edges through a complex maze of lines.

Similarly, a complex graph is a heavy load for the computer. Machine representation of a concept or a relation requires a small amount of dynamic memory to be allocated for each object. If ontology contains thousands of concepts and relations, it is not feasible to consider them all to be available in memory for display. Each concept and relation in a view must also be rendered as changes occur in the model, or as the user changes viewing position to the new part of the model. Concepts may carry details that must be rendered with the concept, and geometrical calculations must be performed when edges are drawn. If rendering process must be performed frequently, the result is likely to be a sluggish and inresponsive user interface.

To cope with these practical limits, some kind of dynamic loading of concepts and relations is required. When building ontologies, it quickly becomes apparent that concepts representing domain objects typically arrange themselves into discrete groups. Models form into smaller submodels that are connected to other submodels. In

graphical representation, this can be seen as *clusters* of nodes. Nodes inside the clusters are arranged close to each other in order to keep relation edges shorter, while concept nodes in different logical groupings are kept away from each other to indicate their unrelatedness or to classify them visually. This division is based on the logical properties of the domain model or some practical criteria developed by the user, and decisions about it cannot be made by the computer (unless some kind of metadata and an algorithm is introduced for this purpose).

By using the concept clusters as a basis of dynamic loading, the amount of information needed and processed at the time can be effectively decreased. Clusters can be seen as basic building blocks for models: when ontology is created, ontology creators arrange concepts and relations into logical, easily manageable units. When the same ontology is browsed, users receive the model incrementally in the same units.

3.2.3 Multiple perspectives

Things tend to appear different depending on the perspective of the viewer. Describing a problem domain as an ontology never produces explicit, predictable result. The result is highly dependent on its creator, his abilities and interests, and the emphasis chosen for the conceptualisation. Domain under discourse yields as many descriptions as there are persons willing to describe it and conditions under which the conceptualisation is done.

This idea can be turned into a concrete benefit by using different perpectives as a foundation of classifying knowledge. Instead of handling ontology as a single, continuous representational structure that tries to be as complete as possible, knowledge can be organised into different *viewpoints* (or *models*). A viewpoint is a complete conceptualisation of the problem domain from a certain point of view, containing all relevant domain objects and describing all necessary relationships. Several viewpoints can be defined in parallel to represent different aspects of knowledge. For example, the *employee* concept concerns all the entities which are employees. Its instances can be seen under several viewpoints: as *employees* from the *accounting* viewpoint, as *researchers* from the *functional* viewpoint, as *customers* from the *restaurant* viewpoint, and so on (Euzenat, 1993). To combine various viewpoints together, concepts denoting same things are interlinked by using knowledge structures called *bridges* to provide a semantic connections between different perspectives. An schematic example of the use of multiple perspectives in ontology can be seen in Figure 3-5, where the same concept appears in three different viewpoints.

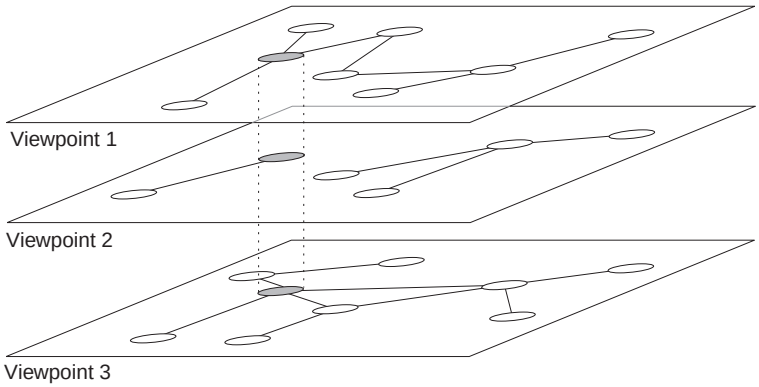


Figure 3-5

From the user interface design point of view, the goal is to provide the user with as easily understandable representation of the content information as possible. Viewpoints and bridges turn out to be ideal tool for this: each viewpoint can be displayed in its own view, with clear indications of relational connections between concepts in different viewpoints. The user can select a viewpoint most suitable for the current task, and ignore all others. If more details are desired, the same information can be viewed simultaneously from different viewpoints, each providing different level or quantity of details. Navigation is enhanced by use of bridges that allow user to swap to a different viewpoint simply by selecting a concept and traversing through its bridges.

The introduction of viewpoints and bridges as means of organising ontologies is motivated by other factors as well. A common problem in ontological engineering is that describing a problem domain on an adequate level involves too much work, unless predefined top-level ontologies can be utilised as foundations of the new ontology. Taking the idea of bridges few steps further: to be able to span different ontologies instead of just viewpoints, we may use them to connect definitions made in different ontologies. Creating a new ontology involves just defining the set of new concepts and relations between them, and linking the new concepts to ones already existing in library ontologies. This way, we may create reusable and modular top-level and domain ontologies that can be harnessed to use later on.

3.3 Design and implementation of CONE

3.3.1 CONE development project

Based on the experiences obtained from the survey of existing tool applications (see chapter 3.1), the requirements set by the ABS project and other VTT projects, and the initial ideas of support methods for ontological engineering, actual CONE application development work was initiated in September 1998. Earlier, a prototype of graphical representation (PRE-CONE) had been developed for the ABS project, and successfully integrated into the ABS Broker version 2.0 at the end of August 1998. The goal of the project was to continue the work done during the ABS project and to build an application suitable for building generic ontologies, and to further develop and evaluate the applicability and usability of the ontological engineering support features presented in this thesis.

The project adopted the iterative model as its process model of software development. The development started with a quick design phase to plan the editor's object model and the basic functionality of the persistent storage backend and continued immediately with an implementation effort for the first prototype. The goal was to experiment with the basic design in practice as soon as possible to correct major design flaws and evaluate its feasibility when working with ontologies stored in a relational database. After evaluation, the development proceeded with adding features and correcting errors by repeating the same *design-implement-evaluate* cycle. New ideas of features and implementation arisen from the concurrent research work were added in a similar way. Table 3-1 lists some milestones of ABS and CONE development projects.

Primary software engineering tools and languages used in the CONE project were adopted from those used in the ABS project. Primarily, Unified Modeling Language (UML) (Fowler and Scott, 1997), Java Programming Language, and Rational Rose 98 application were used as development tools. Throughout the project, Rational Rose modelling tool was utilised to draw class and interaction charts to model various design

concepts, or to reverse engineer the current codebase to display portions of the architecture in the UML notation.

Java was chosen as the implementation language, primarily because of good experience obtained during the ABS project. Java is a clean and easy-to-use programming language, which greatly speeds programming by relieving the developer from laborious and error-prone tasks, such as application memory management. Java's advantages also include high-level application frameworks that enable easy integration with other systems, for example, with relational databases (JDBC). Due to popularity of Java, a number of feature-packed integrated development environments (such as Symantec Visual Cafe) that enable rapid application development and user interface prototyping have become available.

JavaSoft's Java Foundation Classes[2] (JFC) library was used as an application framework to ease user interface coding and quickly prototype different user interface ideas. JFC is a 100% native Java application framework built upon standard Java AWT, providing support for various user interface widgets (windowing, menus, toolbars, dialogs, switchable themes, etc.) and advanced application features (command objects and listeners, undo, etc.).

Date	ABS (PRE-CONE)	CONE
25.6.1998	development of PRE-CONE starts	
10.7.1998	first proto finished	
20.7.1998	meeting in Paris, integrated to ABS Broker proto user terminal	
11.9.1999	PRE-CONE finished	
15.9.1998		development of CONE starts
30.9.1998	meeting in Helsinki, integrated to ABS Broker v. 2.0 user terminal	
15.10.1998		first proto of CONE, loads models from the backend
11.11.1998		first user interface finished
30.11.1998	ABS trials	
1.12.1998		started redesign of the backend, moved to a stricter MVC architecture
8.1.1999	ABS evaluation starts	
10.1.1999		new MVC architecture finished
30.1.1999	ABS evaluation done	multiple views, zooming, cluster expand/collapse functional
15.2.1999		enhanced interface, introduced new tool model
18.3.1999		CONE in Webtran tests begin
31.3.1999	ABS project finished	

Table 3-1

[2] Java Foundation Classes are also known as "Swing".

3.3.2 Requirements

Since CONE was never intended to be a downright product development project, plenty of latitude was left available in deciding what kinds of features should be implemented. A software requirements' specification was initially written to discover potential features and to guide the development work, but it was never meant to be strictly followed. Instead, freedom of selection was left to the project to come up with the most useful support features of ontological engineering. This brainstorming work resulted the ideas presented in chapter 3.2.

However, some initial guidelines were formulated to start with. The following requirements were considered as minimum requirements to shape up among existing conceptual network and ontology editing applications.

- *Clear and aesthetic graphical representation.* The tool should provide a clear representation of conceptual data in the form of a graph. The amount of screen clutter should be minimized, colors and other visual cues may be used to display properties and categorise concepts.

- *Elegant design model*. The application should be built upon a coherent design model that clearly represents the key ideas of the theory of ontologies. The model should be as simple as possible, and should provide only elements that are truly essential, without sacrificing key expressive capabilities (see chapter 3.3.3).

- *Easy to use direct manipulation editing tools*. Pointing and clicking, mouse gestures, and drag and drop should be utilised maximally to provide natural and quick user interface.

- *Incremental navigation.* The application should be able to display only selected portions of the conceptual model, and incrementally expand the graph to show new, undiscovered areas.

- *Multiple views*. Ability to display different views of the same data simultaneously, possibly in different formats (for example, as a graph and text encoded in KIF).

- *Different zoom levels*. Provide a possibility to change the scale of the model display to allow miniaturized views or enlarged views with more details.

- *Modular organisation for ontologies.* Using the application, it should be possible to create libraries of ontologies that can be reused and referred by new ontologies.

To be applicable to various existing and forthcoming VTT projects, the following ideas were emphasised in designing parts of the application not visible to the user.

- *Embeddable*. The editor/browser portion of the CONE should be embeddable in other applications, such as web browsers or applications written in Java.

- *Independence of knowledge representation and persistent storage methods*. The application backend should be implemented in a way that would enable new backends to be built and plugged in to support different storage devices and knowledge representation formats.

3.3.3 Design model

The discipline of human-computer interaction uses generic term *conceptual models* (Preece, 1994) to refer to ways different people conceptualise and understand different systems. Whether interacting with devices, machines, computer programs, other people or the physical world, people use their prior knowledge and experience in developing mental models to understand system's functionality and to predict its behavior. This kind of a view of system's functionality is called the *user model* and it is the basis upon which users start to build their knowledge of interacting with the system in question. Another primary kind of conceptual models is the *design model*, the way the designer conceptualises and views the system.

An important consideration of conceptual models is the relationship between these two models. In an ideal situation, a well-designed system provides mappings of various user models onto design model, i.e. that users and designers "speak the same language", or think in terms of the same concepts and relations existing in the system. If this occurs, the users are able to use the system's full capacity as intended by its designers and the system is easier to learn and understand. In practice, users are seldom able to do this. The design model (and the system built upon it) can be inappropriate or ambiguous for the task the user is trying to accomplish, or it is not well communicated to the user through the user interface, documentation, and instructions of use, collectively known as the *system image*.

The goal of user-interface design is to develop system images that help users to form accurate mental models of the system. The key to this is to design the system so that it follows a consistent, coherent conceptualisation (design model) and to design the user interface, documentation, help systems, etc. (system image) in a way that it conveys the design model in a clear and obvious way.

This goal was kept in mind when creating the design model of CONE. The design model of CONE is built upon *ontologies*. Ontology is an explicit description of some particular domain, consisting of *models* describing different parts of it. In CONE, there is no strict distinction between models and *viewpoints*: a new model may be created to reflect another perspective on some other existing model, or simply to describe some relevant aspects of a domain not modelled before. Adding one or more *bridges* to a model implicitly makes it a viewpoint.

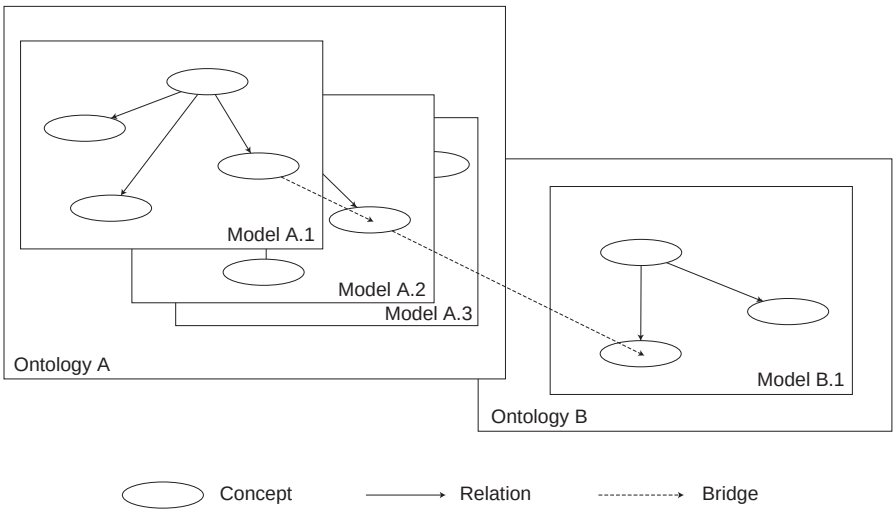


Figure 3-6

Each model consists of any number of *concepts*, representing various domain entities. A concept may be used to represent a generic idea as well as a specific individual, and are described by *properties* and has a defined *type*. Relationships between various concepts are represented using *relations* and *bridges*. A figure describing these relations is shown in Figure 3-6.

From the user's point of view, ontologies are fairly transparent. They are simply collections of models that together describe a domain from one or more perpspectives. Ontologies are also repositories for *types*, user-definable objects to define semantics of concept nodes or relation links[3]. The motivation for arrangement is mostly organisational: by dividing model and type definitions into different ontologies we may define generic top-level or domain ontologies that may be used later when constructing certain task or application ontologies. Each ontology in CONE has a name and a description to indicate the purpose of the ontology. Name of the ontology acts as an identifier for the user, but internally each ontology is identified by unique label called *descriptor*.

From implementation point of view, ontology is always associated with some persistent storage capable of keeping track of entities defined in the ontology. The storage may be for example a file, database, or even a server application on a remote host. In its initial implemenation, CONE supports a local backend using Oracle Lite database.

The basic entitities that CONE user manipulates are *models*, *concepts*, *relations* and *bridges*. Models are descriptions of domains built from concepts and relations. Like in case of ontologies, the user distinguishes models by their names and may attach a free-form description to describe the indicated purpose of the model. New models can be freely added to any existing ontology, and unnecessary ones can be removed.

Concepts and relations are basic building blocks of a model. Each concept belongs to one and only one model, and a relation belongs to the same model its starting and ending concepts belong to. A concept has a defined type and can have a name, which is called *referent* according to the theory of conceptual graphs. Referent determines the entity or set of entities the concept refers to (leaving referent empty indicates existential qualifier), making it possible to describe either distinct individuals or generic concepts. An unlimited set of *properties* can be attached to any concept. A property is a key-value pair describing some specific attribute of a concept. CONE does not set any constraints on the use of properties: it is user's responsibility to decide which properties should be described using properties, and which as new concepts and links.

Relations are used to define connections between concepts inside the same model. A relation link itself does not initially have any semantics. It does not turn into a concrete relation[4] until the user assigns the relation link with a particular type ("subclass-of", for example). Bridges are special kind of links that allow linking of concepts defined in different models (even between models defined in different ontologies). Bridges have

[3] The graph nodes and edges created using CONE toolset (see chapter 3.3.4) do not have any semantics until they are assigned with a type. At this point, nodes are said to turn into concepts, and edges into relations.

[4] CONE currently supports only binary links (and relations). To describe unary relations or relations with arity greater than two, a concept representing the relation should be used.

fixed semantics: concepts connected by the bridge represent the same generic concept, but in different models (or viewpoints).

To support dynamic loading of models, they are further divided into discrete collections of concepts, called clusters. By default, each model contains only one cluster: the default cluster into which all concepts are initially placed. Any number of additional clusters can be defined by the user, and concepts may be moved from a cluster to another. When the model is loaded, only the initial cluster is loaded and displayed. This division allows models to be loaded incrementally cluster by cluster instead of loading all model definitions at the same time.

3.3.4 User interface

The user interface of CONE application consists of the desktop, toolbar, any number of model windows, property and information dialogs, and the type window. A screenshot of CONE in action is displayed in Figure 3-7.

Each model in available ontologies can be opened in its own window. The window displays the model as a graph, with graph nodes representing concepts, and graph edges as directed links connecting concepts. Each concept node is drawn as a rectangle with the concept title in it, and a stacked appearance indicates the existence of one or more bridges connected to that concept. Nodes can be freely moved and resized, and connected links and enclosing cluster border are updated automatically. Double clicking graph items brings up the corresponding information dialog to display details about the item. The same dialog can be invoked using a conceptual menu by right-clicking the graph item with a mouse. Multiple views may be opened to display different portions of the same model, and the zooming level may be adjusted for each view.

By default, when a model view is opened, only the initial cluster of a model is displayed — all other clusters are initially hidden. Graph edges that span cluster boundaries have cluster expansion control (a small handle marked with + sign) attached to it, and clicking the control brings up the hidden cluster. Simultaneously, cluster expansion controls turn into cluster hide controls (two small handles on the edge with – signs) which can be used to hide either one of the clusters. By using this mechanism, the user can selectively browse through the graph displaying only those clusters he is interested in. Cluster expansion state is maintained per window basis, so the user can have two windows displaying different portions of the same model.

To manipulate graph objects, CONE user interface allows user to apply different tools to the graph. The default *selection tool* can be used to select (by pointing and clicking, or by using a selection marquee), move, resize, or delete graph nodes, while *zoom tool* changes the scale of the model view. Object creation tools, *create concept tool*, *create edge tool*, and *create bridge tool* may be applied to add new objects to the ontology. The current tool is selectable by using the global toolbar located at the top of the CONE desktop (see Figure 3-7).

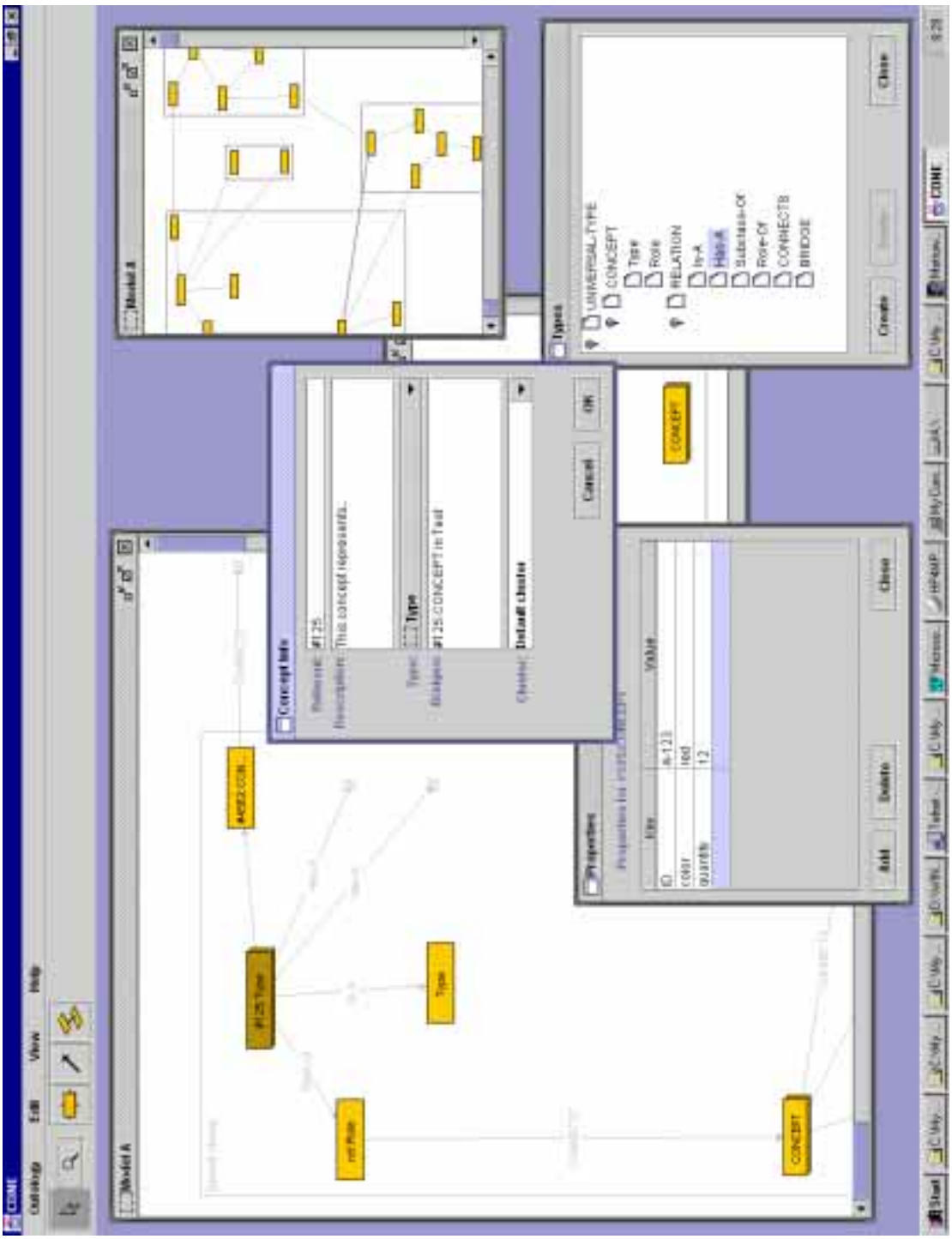


Figure 3-7

3.3.5 CONE architecture

In order to support embedding of CONE into other applications, the architecture (Figure 3-8) divides the implementation into two separate entities, *core* and *application* classes. The former contains the primary classes implemented to support graphical representation and desired editing features, while the latter, application classes, contains a wrapper around the core to provide the default embedding environment. The core module may be detached from the application wrapper and moved to the new one

that makes it possible to use CONE in some other context, for example as a part of another application or as an applet in a web browser.

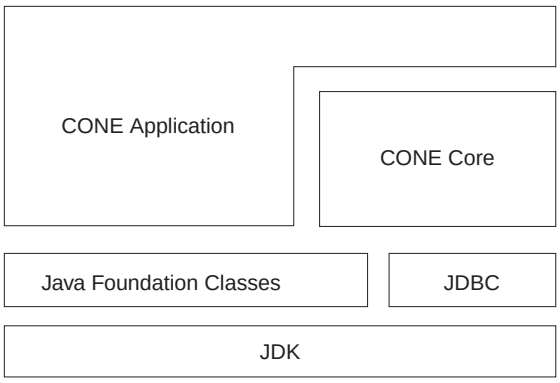


Figure 3-8

CONE application classes are a set of JFC-based classes that provide desktop, windowing, menus, toolbars, and dialogs for the CONE application. Their main purpose is simply to instantiate relevant core classes, embed them inside JFC components to display them and to receive events generated by the user input.

The actual editor functionality is implemented by the set of classes referred as the core. The main purpose of the core classes is to load models from a persistent storage attached to an ontology, display these models to the user, and interpret events propagated by the user and transform them into changes that manipulate the models. The implementation of CONE core classes follows the approach of *model-view-controller* (MVC) architecture, separating the application model object from the view(s) representing it on the computer screen. An object called controller connects these objects, reacting on the user input events occuring in the view and transforming them to changes in the model.

Model classes

Objects representing application objects (ontologies, models[5], clusters, concepts, properties, links, bridges, types) are represented by corresponding model classes (`Ontology, Model, Cluster, Concept, Property, Link, Bridge, Type`, respectively). Each instance of some model class represents some specific object in the ontology, and provides an interface for accessing object's properties. To represent relations between model objects, model class instances refer to other model class instances, forming a relatively complex webs of relationships. Using object references to represent this kind of a structure would not be feasible — referring to a single object for inspection would propagate and require all related objects to be recursively loaded. Instead of object references, CONE model classes use *identifiers* to identify and refer to ontology object instances. The purpose of an identifier is to provide a handle to an object in ontology, without loading the object into memory. An identifier is an unique

[5] Term "model" is used here in two different meanings: primarily, "model" refers to the user-defined description of some problem domain, as described in chapter 3.3.3. As this kind of a description is one of the most central entities in ontologies, CONE defines the corresponding application object called "model". On the other hand, in context of model-view-controller architecture the same term is used to collectively indicate all application objects.

string consisting of *ontology descriptor* and free-form *object identifier*. Ontology descriptor names[6] the ontology in which the object belongs to, and object identifier uniquely identifies the object inside that ontology.

One of the design goals of CONE was to encapsulate the persistent storage of model objects and to support ontologies bound to different storage devices. To enforce this idea, interfaces to model objects are defined in terms of abstract classes that define the protocol for other classes to access them. For each storage system supported, a set of implementation classes inheriting model classes' protocol is defined. Other, non-model classes manipulate model objects only throught these abstract base classes, and are not aware of the implementing subclasses. Using this method, different storage methods (for example, a local file containing model encoded in KIF, or a distributed ontology storage system based on CORBA) may be implemented by defining new subclasses to abstract model classes. Initial CONE version implements the persistent storage using relational database (Personal Oracle Lite v.3.0) via Java Database Connectivity classes (JDBC).

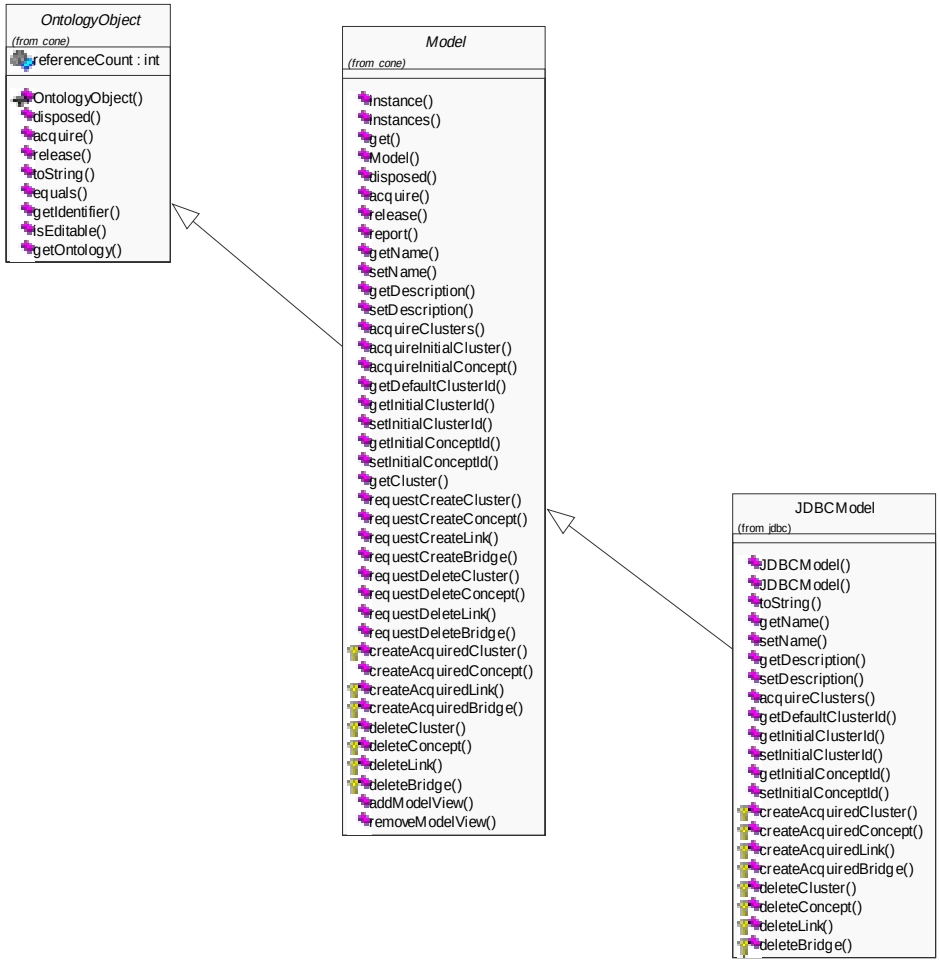


Figure 3-9

[6] In the current implementation, names like "cone-app", "cone-domain" and "webtran" are used. If CONE ontologies were adopted in a distributed environment, a hierarchical naming scheme similar to Java packages should be used (for example, "fi.vtt.tte.webtran").

Figure 3-9 displays abstract `Model` base class and `JDBCModel` implementation class. Similar class hierarchies are defined for all model classes. The complete model class hierarchy is in Appendix A.

Model objects are loaded from the backend on demand. In case of a JDBC-based default backend, request for certain object (e.g. concept) results a SQL query to the database. Query result is packaged as an object (e.g. instance of class `JDBCConcept`) which is returned to the requesting party. If there are multiple requesting parties, a question of integrity surfaces: there should always be only one instance representing certain object in the ontology. CONE solves this problem by maintaining a simple reference counting scheme and a cache of loaded instances as a class member variable in each model class. When an object representing certain ontology object is requested, the class instance cache is first looked up to see if instance for the object in question has already been created. If an instance exists, its reference count is increased and the instance is returned directly from the cache. If not, instance is created by the backend, placed to the cache, and finally returned with its reference count incremented. It is requesting object's responsibility to release the object by calling its `release()` method when it is no longer needed. To indicate this reponsibility, methods that increment model object's reference count are named `acquire` prefix. This scheme guarantees that there is always one and only one instance in memory per certain ontology object.

View classes

To display ontologies and models for the user, a set of view classes has been defined. Each view class object implements a visual display of the corresponding model class object. Any time the model object changes, each view object is notified so that the visual representation of the object can be updated. Multiple view objects can be attached to the model object to display the object simultaneously in multiple windows.

Each model is visualised in a model view (defined by class `ModelView`). The model view does not actually have any visual representation (except the white background). Its main purpose is to manage creation and deletion of other view objects and arrange them as a graph. Class `ModelView` inherits from the generic `GraphPanel` class, that is a part of a separate graph drawing package[7]. Among other things, this class provides a support for scaling and node/edge selection.

Concept view (class `ConceptView`) provides a visual representation of a single concept in the ontology. The view object has a reference to the corresponding model object and uses it to obtain the size, location, and title of the concept. Each concept view draws itself according to instructions given by the enclosing model view: layout and level of details is dependent on the current scale and drawing settings managed by the model view. Concept views inherit the abstract `GraphNode` (defined in the graph package) class to operate as nodes of a graph.

Link views (class `LinkView`) draw themselves in a similar manner as the concept views do: the information needed to draw the link (type title) is obtained from the corresponding `Link` object. To make drawing fast enough, the link edge maintains a cached value for start and end points which are recalculated when a connected node is

[7] This package was developed initially as a part of CONE, but later on separated into its own Java package. The package encapsulates graph drawing into set of classes that can be used by other applications needing generic graph drawing abilities.

moved or resized. As concept views, link views inherit from graph package (class `GraphEdge`) to function as edges of a graph.

Cluster view (class `ClusterView`) represents a cluster enclosing a set of concepts. The cluster view keeps track of concept views enclosed in it and draws a border rectangle around the concepts.

Controller classes

A *controller* object binds each model view (`ModelView`) to a certain model object (`Model`). Its purpose is to interpret UI events received by the view, and convert them to higher level actions that manipulate the model. The controller invokes change method in the model, which records the changes into the persistent storage. After that, it sends notification messages to the view objects to reflect the changes in the user interface. In CONE, the controller part of the architecture is not a single object but a group of objects: the input is processed by one or more tool objects, and then passed to the model controller object. The processing is started by the current tool which converts raw input events received from the model view to an instance of the class `GraphPanelEvent` according to tool's type and the type of the input event. The current tool and the global tool state are managed by a state machine in order to automatically provide the correct tool based on user actions. A class hierarchy of tool classes implemented by CONE is shown in Figure 3-10.

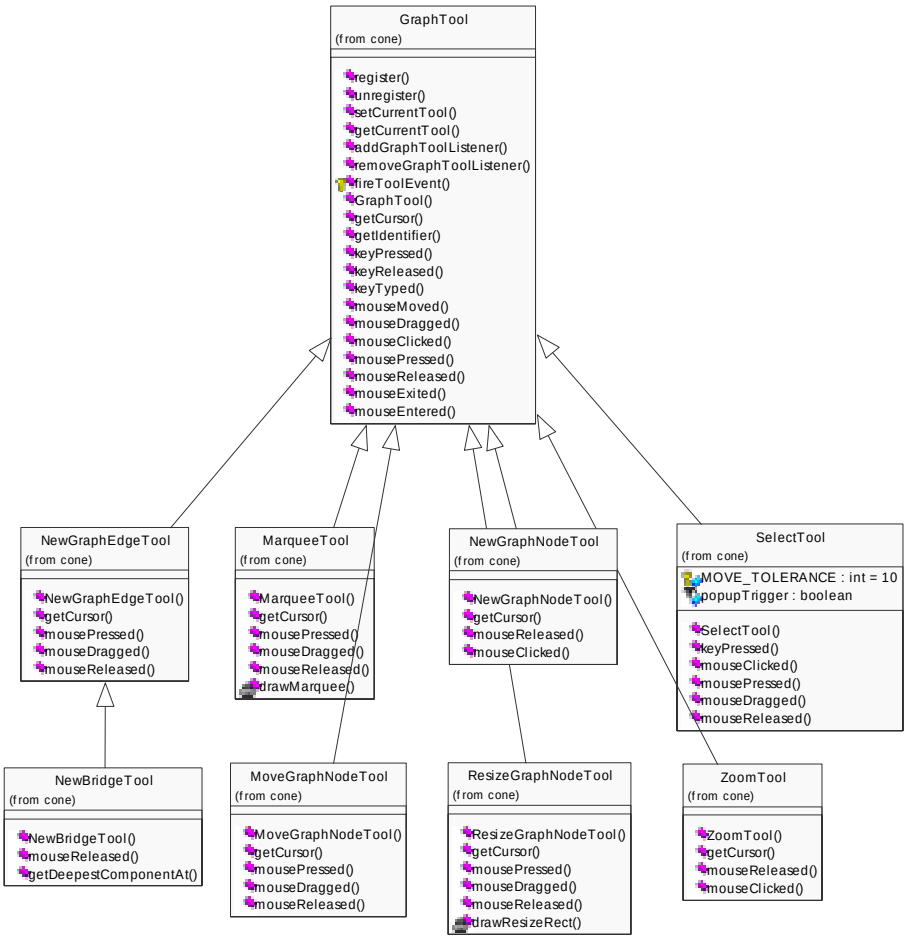


Figure 3-10

MVC operation

The components described above form the basis of the main functionality of CONE. To render objects on the screen, view objects request corresponding model objects for the information needed. To carry out actions invoked by the user, view objects forward user events to the controller classes for further processing. Controller determines the type of the event, and sends the change request to the model object. Model object, being the last in the chain, processes the event by manipulating the actual model and notifies all views about the change. For example, mouse drag to move a node (`ConceptNode`) is first processed by the selection tool (`SelectTool`) which determines that the user tries to move node(s) in the current selection. It grabs all user events until a mouse up event occurs, and creates a new graph event (`GraphPanelEvent`) with event identifier `NODES_MOVED`, and sends it to the model controller (`ModelController`). Model controller determines the model (`Model`) for which the message is targeted to, and requests the model to move the desired nodes. Model records the node positions to the persistent storage, and sends node position change message to each concept view (`ConceptView`) affected. This kind of a view-controller-model-view loop is always carried out when events are invoked in the model view. Views never update themselves directly.

The use of MVC architecture has several benefits: responsibilites are divided into clearly defined modules, which may be modified without causing anomalies in the others (e.g. new view types may be introduced without changes in the controller or in the model). Support for multiple views (even of different types) is easy to implement, and error handling and recovery fits naturally into the architecture.

Another example of MVC operation in CONE is given in Appendix B, which shows a UML sequence diagram of actions that occur as a response to user clicking the model view with concept creation tool selected.

4 Case study 1: Applying CONE to electronic commerce

4.1 ABS: Architecture for Information Brokerage Services

VTT Information Technology has been a partner in the Architecture for Information Brokerage Services (ABS) project, which was a part of the ACTS (Advanced Communications Technologies and Services) programme within the European Union's 4th Framework of scientific research and development (1994-1998). The main objective of ABS has been to define and specify an open information brokerage service architecture to permit efficient provision of online information services in the context of electronic commerce.

To allow commercial transactions to take place, either in traditional or in electronic commerce, the customer and the vendor must first come across and agree with the terms of the bargain. In the Internet it is often a problem for the users to locate the "right" service providers, especially when they have only a vague idea of what they are actually looking for. The concept of information brokerage presents numerous ideas to help end-users in accessing services and information more easily and to allow service providers to publish and advertise their offers. The purpose of ABS project was to study problems associated with designing and implementing this kind of a mediation platform.

ABS project, which started in July 1996 and ended in March 1999, involved ten partners from six different EU countries (Table 4-1).

France Telecom CNET	France
Degriftour	France
SEMA Group Ltd.	France
Deutsche Telekom Berkom GmbH	Germany
Point of Presence GmbH	Germany
VTT Information Technology	Finland
Sonera Ltd.	Finland
National Technical University of Athens	Greece
Universidad Politécnica de Madrid	Spain
Portugal Telecom CET	Portugal

Table 4-1

The main objectives of the project were to define and specify an information brokerage service architecture based on ODP-RM and TINA concepts, to design and implement prototypes of the design using Java and CORBA technologies, and to validate the service by organising and conducting regional and international trials in real life electronic commerce environments.

The project consisted of six working packages involving preliminary studies, service definition, system specification, architectural design, prototype implementation, trials,

and evaluation. VTT Information Technology was an active contributor during the entire project and took part in all working packages. Most of the efforts were directed to design and prototype implementation, in which VTT was in the key role.

4.1.1 ABS Enterprise Model

The Enterprise Model defined by ABS summarises all actors, their domains and their relationships and policies. ABS identified the following set of actors applicable in brokerage.

The *user*, which can be a human, machine, or an application, has a need to locate information or product supplied by a *content provider*. To make this possible, communication services transferring the information from the originator to the recipient are required. *Brokerage service provider* offers information on services and content available in the network; services which are subject of the brokerage (e.g. VoD, video conferences), or supporting services (e.g. security, payment, certificate, communication). The *broker* may mediate information about the trading objects, but it does not own the services itself. It is the *retailer* that is supposed to have the rights to sell, and hence does not only mediate information about the goods, but also trades them. In this way the retailer acts also as a content or service provider towards the user domain. The ABS vision of a retailer is displayed in Figure 4-1, where it is overlapping with Broker, Service Provider and Content Provider domains. All these activities take place in a network where telecommunication services as well as supporting services (security, certificate, and payment) are offered by Service Providers. Networking connections can be supplied by *network providers*.(Casulli et al., 1998)

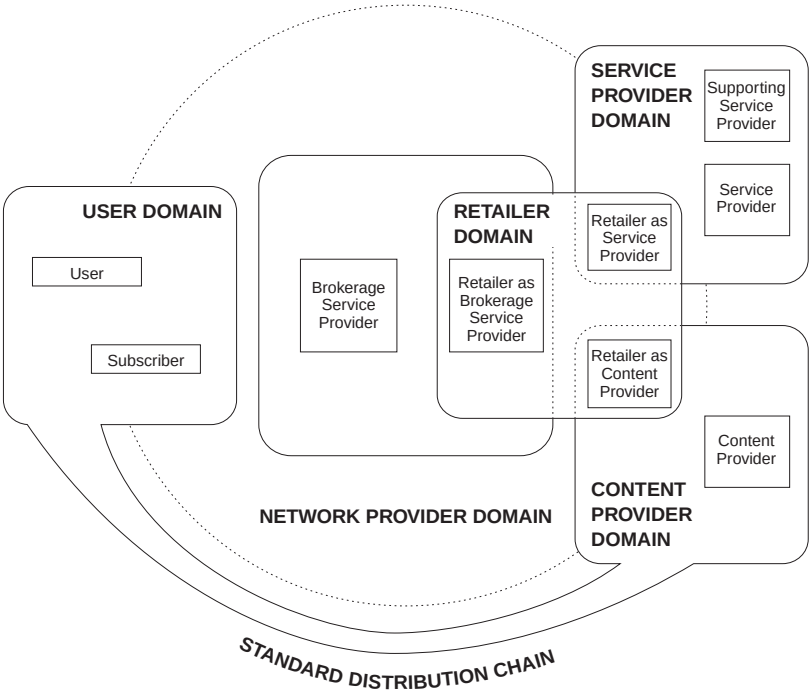


Figure 4-1

The key issues in this model are the distinction between the broker and retailer actors, and the introduction of support service providing parties like payment, billing, and security service providers, as actors from separate domains.

4.1.2 ABS Information Model

The Enterprise Model defined by ABS is reflected in the ABS Information Model. This model describes the entities inside the system and information involved in its information processing. The main requirements of the information model are (Casulli et al., 1998):

- to create a projection of the complex, heterogeneous, and unstructured provider domain into an internal structured representation
- to give the users the appropriate view of the provider domain and enable the access to information supplied by providers
- allow content providers to target their offers towards potential customers

To manage complex information, ABS uses ontologies as means of structuring knowledge of the brokerage domain. The information is represented using a *conceptual network*, a multi-relational and multi-hierarchical structure of nodes and relationships between them. It forms a knowledge-based conceptual representation of the relevant aspects of the domain subject to brokerage service, representing domain entities as concept nodes connected to each other by relation links. The manifestation of a particular concept associated with a certain perspective, or *viewpoint*, is called a *shadow object* of that concept. Various shadow objects of a concept are interconnected using constructs called *bridges*. The conceptual network forms thus an interconnected web of concepts arranged into a multi-dimensional space, allowing the same information to be viewed from multiple perspectives.

The knowledge-based model provided by the conceptual network is used to associate content provider related data, called *resources*, with concepts comprising the conceptual model of the domain. A resource is a composed entity that encapsulates model items from the content provider's domain. It contains a description of the content and interfaces for querying and accessing the data of a resource.

As an example of the use of ABS information model, a broker providing services for traveling business would use its conceptual network to describe catalogs of information about accomodation, restaurants, activities, car rental, attractions, city maps, etc. Different perspectives, such as business, sports, tourism, nightlife, etc. would be provided as separate viewpoints, and descriptions of the various services would be encoded as resources attached to shadow objects in those viewpoints.

4.2 CONE and ABS

In ABS, the motivation for developing graphical representation of ontologies was the need for an end-user terminal component that could display conceptual network entities and relations. The initial idea was to develop a graphical browser/editor for conceptual networks that could be used in both end-user and content provider user interfaces. However, due to the project schedule of ABS, building graphical application with complete editing features was not possible, and the project settled for a simple

graphical representation (here referred as PRE-CONE, or preliminary CONE) for end-user UI coupled with a simple text-based editor for creating ontologies.

The requirements imposed by ABS were rather straightforward. The main concern in using graph-based representation was the large size of viewpoints to be used. A typical viewpoint could consist of hundreds of concepts connected to each other. For this purpose, a clustering scheme was developed to allow a client to incrementally load conceptual network data from the broker, and graph representation was built to display one cluster at a time and allow user to move from one cluster to another. Other basic requirements were the ability to display arbitrary concepts and binary relations, ability to automatically layout graph nodes that do not have position information, to allow user to pick a concept for closer inspection, and to use bridges to move from a viewpoint to another.

For these purposes, PRE-CONE was implemented. The goal was to provide a module that could be embedded in the user terminal application developed by the Portugal Telecom/CET partner. Figure 4-2 displays a screenshot of ABS user terminal that displays a part of the conceptual network used in the Finnish trial.

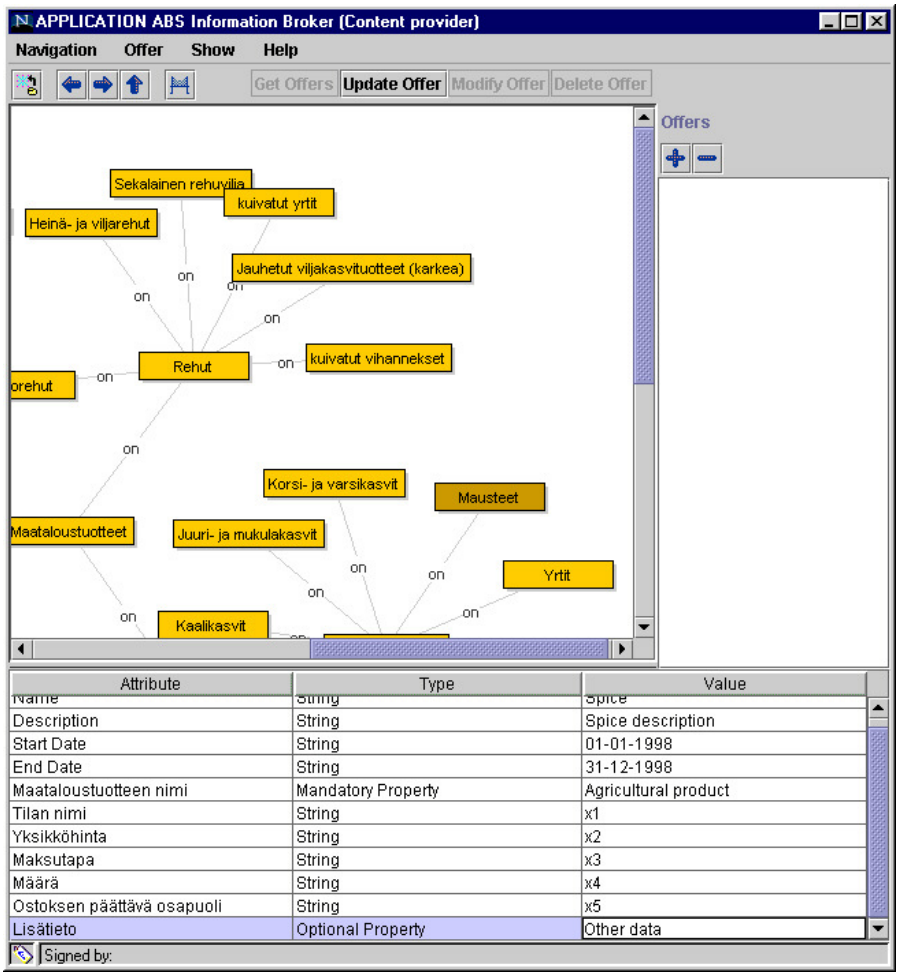


Figure 4-2

A lot of ideas and implementation details figured out during the PRE-CONE development were later used as a starting point for the actual CONE development work. Initially, both projects used the same rendering code, but later on CONE

implemented its own scheme to render complex views more effectively. Event handling was also the same in the beginning, but PRE-CONE approach was later on discarded in favor of more flexible MVC paradigm. PRE-CONE also gained from the CONE development: just before Finnish trials, some implementation ideas discovered during CONE design phase were incorporated into PRE-CONE.

4.3 Trials and evaluation

As a part of the ABS project, two series of trials were arranged to validate various architectural solutions and get some feedback from end users. The first trial that was arranged in October 1997 was internal to the project and its main purpose was to validate the kernel functionalities of the system. The second trial, arranged in November 1998 simultaneously in Finland, France, and Germany involved real users, and its purpose was to evaluate the broker as a whole. As a part of the project's working package 5 (2nd trial), VTT Information Technology participated in conducting Finnish trials and evaluating trial results.

4.3.1 Evaluation approach

To balance the load of evaluating the results of a relatively large project, each trial site focused on particular subset of ABS broker functionality. The main focus of evaluation in Finnish trial was on the broker's ability to allow dynamic accumulation and maintenance of service offers. The secondary goals were to evaluate the usability of the broker user interface and to assess the users' acceptance of ABS broker as a service mediator platform. This trial, or more specifically the part of it concentrating on usability issues, was an excellent opportunity to evaluate PRE-CONE and get some feedback on graphical representation of ontologies from the so-called "naive" users.

Considering PRE-CONE, the main goals of the evaluation were to test and get feedback on its:

- *Applicability*: Is the graph-based representation of ontologies applicable to services like ABS?
- *Understandability*: Is the visual representation used in PRE-CONE understandable? Does it help in finding information? Are the graphical primitives adequate?
- *Functionality*: Are the technical solutions adequate what comes to performance?

The trial was arranged at VTT premises as a one full-day session. The trial audience was selected from a related project: a group of seven farming and rural travel entrepreneurs[8] was invited to spend a day getting acquainted with ABS, trying out its various components, and discussing broker details and electronic commerce in general. For trial purposes, a small local network had been built: four desktop clients running

[8] Test users selected for the trial were members of an active Internet community called Agronet (http://www.agronet.fi). They all were familiar with the Internet and the possibilities of electronic commerce. This audience proved to be ideal for ABS project purposes, because they already had some hands-on experience on brokerage and were aware of problems involved with commerce and information retrieval in the web.

Windows 95 and Netscape Navigator web browser were connected to a high-performance UNIX server hosting a web server and various ABS broker servers. Client machines were set up in a classroom to enable group discussion and demos, and to make it possible to capture the events of the day on videotapes.

The trial event started with a brief introduction to the ABS project, followed by a short demo to introduce the users with the ABS client and to demonstrate the key functionalities of the broker. Due to the prototype status of the ABS broker, this kind of an introduction was considered imperative, even though it was understood to have bias on the test results. Otherwise, the results would have been spoiled entirely by test users stucking into unimplemented or erroneous features, or simply by not knowing enough background information to get into issues under evaluation.

The evaluation proceeded with a hands-on experiment. The test users were divided in groups of two to three persons, and each group was accompanied by two or more evaluators (ABS project employees) acting as observers and instructors. Each user was given some time to get familiar with the ABS client, and to try out actions shown in the demo. The users were then given some example tasks (for example, "try to find information about prices for potatoes and locate farmers selling them") to be accomplished and were asked to "think aloud" — to tell about their ideas and assumptions while performing the task. In case the user did not know what to do or strayed into wrong direction, the instructor helped the user by giving hints or showed how the task was supposed to be carried out. After few of these tasks, the test users were asked to adopt the role of a content provider and in a similar way to carry out tasks involved with adding and updating information.

The trial day concluded with a free-form discussion chaired by evaluators, and a questionnaire the users filled out before leaving. The purpose of the free-form discussion was to broaden the scope of the evaluation from specific ABS features to a more generic assessment of the ABS system — how would it possibly help the test users in their everyday lives, and what are the main problems of the brokerage from the test users' point of view. The questionnaire was used to collect background information about the users and to provide them a possibility to express ideas they otherwise would not have stated in public, or to list down issues that hadn't crossed their minds earlier.

To collect the data of this one-off event, four different methods of capturing the user actions and feedback were used (Poulain et al., 1999):

- *User observation*. A video camera was used to record the events of the entire trial day. During hands-on trial, video camera was aimed to a single group and was used to capture user actions, comments, and events on the computer screen. In addition to that, one observer in each test user group made notes about the user actions as they occured. The other observer interviewed idle group members while one of them was working with the ABS client.

- *Interviews*. During the hands-on trials, observers interviewed the test users, asking questions about their assumptions and expectations concerning the broker, their feelings about the way information is represented, etc.

- *Questionnaires***.** Used to collect background data and free-form comments.

The first two methods were the primary means used to collect user comments concerning PRE-CONE.

4.3.2 Results and commentary

In order to obtain a clear picture of the feedback received during the trial, user comments were collected from the various data sources, and classified according to their topic. The comments were summarised and finally presented in ABS final evaluation report (Poulain et al., 1999). The same approach was used to evaluate PRE-CONE. The following is a summary of issues involved with the graphical representation part of the ABS system.

During the trial and evaluation it became quite clear that ontologies, no matter whether they are represented graphically or by some other means, are not for end-users. The reason is not that the contents of the ontology were not understood, but rather that the reason why such an explicit and detailed information was displayed was not clear to the users. Users characterised ontologies as "unfamiliar", "strange", "techie-style", or "very database-like" way of displaying information: correct and very detailed — yes, but helpful for locating services and service providers — no. The reason for this is that plain ontologies do not provide any added value. They are just very detailed descriptions of domains, formalised and perhaps displayed using some representation scheme. In information services intended for information retrieval, users require inferred and preprocessed information: classifications, indices, ways of automatically finding related data, etc.

In that sense, the approach chosen by ABS was not adequate. Building a broker service based on ontologies can bring several important benefits for the user, but the user interface should also be designed to allow the user to fully take advantage of rich knowledge content stored inside the broker. Rather than providing the user with a view to this extensive knowledge repository, he should be provided with a familiar interface (such as hierarchical classification used in many internet information services) that is dynamically constructed using inference processes by extracting relevant information from ontology. The existence of an ontology allows inference processes to discover relationships that otherwise would be very difficult or even impossible to find.

After the initial astonishment had been passed and some background information on ontologies been given, a plenty of good user comments concerning graphical representation were received from the trial users. When separating PRE-CONE from the context of ABS, the users agreed that the graphical represention of ontologies as graphs is a clear and illustrative technique. As further improvements, they suggested the use of colors and pictures (icons to replace rectangular nodes) to make graphs more pictorial. Free-form explanatory texts could be used to display notes, comments, etc.

An interesting finding was that when provided with the graphical view of conceptual network, the majority of users immediately started to reorganise graph node positions by hand. Some users adjusted only a few node positions, while some others spent a whole lot of time to make the graph layout match their personal desires. It seems that different people tend to have different view on how the graph should be laid out, and ability to do this manually is of great importance. Typically, users tried to move concepts to groups of closely related concepts, which supports the clustering approach chosen in CONE.

Even though the ABS broker development and evaluation was mostly focused on other topics than those central to CONE, the process brought up several important ideas that had an effect on the way in which CONE was eventually shaped up. For example, clusters were initially developed in ABS to manage large viewpoints. ABS project acted as a launch pad for CONE design and development.

5 Case study 2: CONE and multilingual information retrieval

5.1 Webtran: WWW-based machine translation for controlled languages

Nowadays, information on European legislation and intergovernmental agreements is scattered in distributed repositories in hetereogeneous formats and is available in many languages. This information is technically accessible through information networks, but it is extremely difficult even for professionals to use it because of differences in document structures and languages. This is a common problem in cross-lingual information retrieval (IR) systems where queries are made in one language to a document collection in several different languages and the goal is to retrieve only those documents relevant to the query. Before retrieval can be performed, deep linguistic analysis and translation of the query appears to be necessary.

Webtran is a machine translation system for controlled languages (CL) to be embedded in WWW-based information service systems (Lehtola et al. 1998). It is designed to provide a full support for translation in online WWW services, such as an online mail order catalogue or information retrieval from cross-lingual databases. The framework in which Webtran is involved consists of a user interface through which the user can make queries in his own language to search for legislative texts from different EU databases of EU regulatory information. The query is translated by Webtran Translator into the language of the target documents before being forwarded to the multilingual databases. Retrieved documents will be displayed in their original languages. In the domain of legislation, users usually prefer to have texts in their original languages so that the interpretation would be more reliable.

5.2 CONE in Webtran

For low-cost services of the access to the legislative databases through WWW, it is necessary that a fully automatic translation would achieve a reasonable performance. To do so, the approach adopted by Webtran is based on controlled vocabulary. This helps to relate terms in one language to a common set of language dependent concept[9] identifiers. At the language level a concept can be expressed by a term and its synonyms which can be single words or longer surface expressions. The term is the most obvious or most widely agreed expression of the synonyms. Furthermore, there can be semantically close expressions that are not accurate but approximately reflect the meaning of the term. For example, the official term for the concept "avoiding payroll tax" in Finnish is "ennakonpidätysvelvollisuudesta vapauttaminen", and one way of expressing it may be "ennakonpidätyksen välttäminen". Expressions of a term in different languages can also be viewed as synonyms. Table 5-1 shows an example of Finnish and Swedish surface expressions of a concept.

[9] In Webtran, the word "concept" is used to refer to interrelated items in a conceptual model, that have been defined by humans for a particular target domain.

Finnish	ennakonpidätysvelvollisuudesta vapauttaminen
Swedish	befrielse från skyldighet att verkställa förskottsinnehållning
English (approx.)	acquitting from the responsibility of paying payroll tax

Table 5-1

The use of concepts for query translation can enhance the retrieval performance. With a plain string based boolean matching, synonyms or otherwise semantically related words are not included in the result and thus the precision of retrieval is decreased. To achieve an IR scheme that takes word semantics into account, the system requires experts in legislation to define the conceptual models and relationships to surface expressions in the covered languages.

To build a information retrieval system based on this idea, a model of the searchable information must be defined. Each term occuring in the corpus must be mapped to the corresponding concept, and each synonym of the term must be associated with its base term. If multiple languages are supported, these relationships must be formed for each language separately, and the terms in different languages should be linked to represent their equality in different languages. The result of this modelling work is a multilingual ontology of the searchable information.

The Webtran project uses CONE application to model various concepts of legal terminology in context of travelling business. In these models, concepts are used for representing various terms of the corpus. To describe the structure of the terminology, relations (for example, *uses-for*, *broader-term*, *narrower-term*, or *related-to*) are established between concepts to associate terms with each other. Synononyms for terms are described as properties of the base term concept. Laws in different languages but with the same content use different terminology and that is why each language is described in its own viewpoint. However, different languages have equivalents on the concept level, and bridges can be used to provide an alignment between languages. Initially, only Finnish terms are to be described, other languages (Swedish, English) to be added later on.

5.3 Evaluation and commentary

Upon writing of this thesis, the Webtran project has proceeded to a stage in which the initial tests of using CONE have been carried out, but the application has not yet been tried out with real data and real use cases. The exact method of using ontologies in Webtran is still under consideration, and some research is to be done before proceeding to building extensive multilingual ontologies. A complete assessment of CONE is not yet reasonable because the number of active users is too small and there is not enough test case material to be analysed.

Webtran project has been acting as a pilot project for CONE. In order to test CONE's suitability for the needs of building ontologies for multilingual information retrieval and to guide the development into the desired direction, the prototype was taken into test use relatively early during the development process. This test use provided a framework for exercising CONE with real data and making a small-scale evaluation of the suitability of its features. However, since this evaluation is based only on a brief interview with a single test user, it should not be considered as any kind of complete proof of the ideas presented in this thesis.

5.3.1 Evaluation approach

The motivation for this evaluation consists of two primary issues:

- *Validation of the basic functionality*. At the point in which the main planned functionality has been implemented and the application can already be used with real data, affirmation of the current ideas for the user interface and major technical solutions is in place.

- *Feedback and ideas for further development*. End-users are experts of their domain and are a very valuable source of ideas. They have general insight of the task at hand and typically are well aware of the problems involved with it. Arranging test use provides developers with plenty of feedback and ideas for new features.

Since mid March 1999, one Webtran project employee has been studying the use of CONE and using it to build small ontologies modelling various legislation concepts. The user has been working with CONE independently, using it in ways best suitable for the tasks required. The goal of this work has been to initiate the modelling work in Webtran project and test-drive CONE with real data.

The evaluation was initiated by arranging an interview with the test user. To find the questions most suitable for the evaluation interview, a generic method based on the Goal Question Metrics paradigm (GQM) (Basili, 1992) was utilised. The idea behind GQM is to use a systematic mechanism for defining, measuring, and evaluating software engineering processes. Goals of the organisation and its projects are defined and refined into a set of questions characterising the objects of measurement (e.g. products, processes, or resources). Each question tries to characterise the object of measurement with respect to a selected quality issue and to determine its quality from the selected viewpoint. Questions in turn are refined into metrics that can be used for measuring the issues involved (whether they be objective or subjective to the viewpoint taken) in a quantitative way.

At the current stage of the Webtran project, applying GQM paradigm in its entirety to CONE evaluation would not be feasible, and not even desirable. Most of the metrics would be very subjective and would reflect only the views of two different perspectives, namely those of a developer and a single test user. There is no "organisation" that could answer the questions and give a broader view to the applicability of CONE. However, the GQM approach can be used to define goals of the evaluation and refine them into generic questions about the feasibility of various features. These questions can then be refined to be presented in the interview. At this stage of evaluating CONE, the main problem is not to find reliable and accurate metrics for measuring the software, but to ask the right questions to get as detailed comments as possible, and to discover problems not anticipated by the developer. This is why the approach adopted in this evaluation limits the use of GQM to its two higher levels, namely those of defining goals and questions, thus applying GQM in a very generic manner. Goals define the main focus of the evaluation, and questions form the guidelines the interview is based on. Results of the interview combined with the developer's subjective evaluation of CONE can then be used as a basis of further development work.

Adopting GQM in an early stage of the project has several benefits, even though it cannot be entirely utilised: abstract problems (e.g. "is the approach chosen by the project reasonable?") can be divided into subproblems that are easier to answer and

evaluate, the systematic approach guarantees that evaluation is not limited to a single viewpoint, and the purpose of all evaluation actions is clearly defined. Furthermore, the groundwork initiated early in the project eases the accomplishment of more complete measurement and evaluation to be carried out later on.

The following is a set of GQM templates used for constructing questions concerning main CONE features. Each template describes the goal it represents, and a set of questions to be answered by the evaluation. The same questions were also presented to the user, even though in a slightly modified form. Appendix C is a summary of questions presented to the user during the interview. Commentary of user responses is presented after each template.

5.3.2 Commentary on graphical representation

Goal	
Analyse	CONE graphical representation
in order to	evaluate its usability and understandability
with respect to	user friendliness, correctness
from perspective of	user
in context of	CONE development project, CONE applicability to Webtran as ontology support tool
Questions	
1. Is CONE's way of representing ontologies clear and easy to understand?	
2. Is the meaning of all objects in the model view obvious? Are their purposes understood and is the connection between them and the theory of ontologies clear?	
3. Is the graphic design of the model view good? Is the model view pleasant and ergonomic to look at? How could it be improved?	
4. Are there any benefits in using different zoom levels when viewing ontologies?	
5. Are multiple views used to display parts of a model? If not, why?	
6. Is the amount of details adequate? Is there something to be added or removed?	

Table 5-2

The interview revealed CONE's strengths to be clear and easy to understand visual representation of ontologies, and its well-thought visual appearance. Even though the interviewee did not have previous experience with ontologies, CONE's way of visualising concepts and relations was easy to adopt and helped in learning the theory of ontologies. Zooming and multiple views to display different parts of an ontology were considered to be worthwhile ideas that greatly help working with large ontologies.

As main improvements, model view rendering should be optimised to make working with complex models more efficient, and example models should be provided to make learning CONE easier. More user-defineable options for visualisation (e.g. colors, shapes for concept nodes, different thicknesses and arrowheads for relation edges) should also be provided to allow markup and customisation.

5.3.3 Commentary on editing tools

Goal	
Analyse	CONE editing tools
in order to	evaluate their efficiency and ease of use
with respect to	efficiency, intuitiveness
from perspective of	user
in context of	CONE development project, CONE applicability to Webtran as ontology support tool
Questions	
1. Are editing tools available in CONE easy to use? Is it easy to move or resize nodes?	
2. Is selection marquee used to select multiple items (for example, to move them)?	
3. Are the tools provided by CONE relevant?	
4. How about properties and attributes? Are they relevant and correctly handled?	
5. Is the tool state managed correctly? How often errors do occur?	
6. Are CONE tools efficient? How long does it take to create a model?	
7. Is there a need for more tools, for example to automatically reorganise graph layout?	

Table 5-3

According to test user's experiences, CONE was very easy to learn because of its toolset similar to many Windows-based applications. Basic editing tools (moving nodes, creating and deleting concepts, relations and bridges) are intuitive, and enable quick graph editing and manipulation. Tool state is managed in a coherent way and is error-free in general. Concept node resizing tool seems to have an inadequate affordance: the user was not even aware of the possibility to resize nodes, because this action is not visible in the user interface.

More freedom of use is needed in CONE. Properties and attributes are possibly too fixed, and cannot be used for all purposes required by Webtran. The user should be able to more flexibly adjust CONE capabilities to different uses. Automation tools could be considered (i.e. to manage clusters or to layout graphs automatically[10]) as extensions to the current CONE toolset.

[10] PRE-CONE had this functionality provided by a simple spring layout algorithm. However, graph layout algorithms are typically very time-consuming, and the results are often not good enough. To apply such an algorithm in CONE, more advanced method taking cluster, node and relation semantics into account should be used.

5.3.4 Commentary on clusters

Goal	
Analyse	use of clustering feature
in order to	evaluate its usability and see if it helps managing large ontologies
with respect to	ability to reduce user's cognitional load and model loading time, and ability to allow user to concentrate to relevant parts of the model
from perspective of	user
in context of	CONE development project, CONE applicability to Webtran as ontology support tool
Questions	
1. Are clusters used? Do users see benefits in the use of clusters?	
2. How large models user typically create? Approximately, how many concepts do they contain? How many clusters?	
3. If clusters were not available, how many concepts would be contained in a model?	
4. Is the load time of a cluster adequate? Is the response fast enough when user clicks at the cluster expansion control to display a new cluster?	
5. Does the use of clusters have effect on users' ability to find particular concepts from the model? How does it show?	

Table 5-4

Though some major problems with the clustering feature of CONE still exist, the interviewee considered clustering feature to be very useful. Clusters are used to represent various semantic fields inside a model, mainly as means of organising knowledge. This far, number of clusters and number of concepts inside them has been relatively small (approx. maximum 50 concepts) and reliable results of using clusters for navigation cannot be determined. However, by using clusters, model load times have been reduced to acceptable levels.

A major problem of using clusters is their reachability. By default, all other clusters except the initial cluster are collapsed and not visible. In this situation, it is difficult for the user to navigate to a certain concept if the only indication of the existence of other clusters is the cluster expansion controls on relation edges crossing cluster boundaries. Using some abstraction method (i.e. cluster proxy) to indicate the existence of a cluster should be considered. Another problem, caused by a design flaw in the current implementation, is the existence of stray clusters that cannot be reached unless they have relations or bridges to some other cluster.

5.3.5 Commentary on viewpoints and bridges

Goal	
Analyse	use of viewpoints and bridges as means of organising knowledge inside ontology
in order to	evaluate their usability and ability to represent parallel models
with respect to	benefits of use, correctness
from perspective of	user
in context of	CONE development project, CONE applicability to Webtran as ontology support tool
Questions	
1. Are parallel models and bridges used to represent different viewpoints of some ontology? What is the basis of the classification method used to divide concepts into models?	
2. Is the graphical representation of bridges adequate?	
3. Are bridges used primarily as means of organisation, classification, or navigation?	

Table 5-5

Using viewpoints and bridges is an essential feature of applying CONE in Webtran. In initial test cases performed by the interviewee, she used viewpoints to describe the same concepts in different languages, and bridges to combine the concepts that have the same meaning or function in their own contexts. Bridges were also found beneficial when navigating through large models, as translations of terms in different languages are available as separate viewpoints linked together via bridges.

Graphical representation of bridges has still some room for improvement. The current bridge implemention is one-way: the bridge is visible only in the starting concept of the bridge. For Webtran purposes, bridges should be automatically two-way and visible in both source and target concepts of the bridge. Furthermore, some other visually more affording representation than the current stack-of-boxes should be considered.

6 Conclusions and future work

The goal of this study has been to develop practical techniques for creating and maintaining ontologies. The benefits of the theory of ontologies are well established and ideas of their uses widely spread, but not much research has been done to bring these benefits into practice. Working with large ontologies is difficult, mainly due to their large size and complexity. Tools for creating and maintaining ontologies are few.

This research effort has tackled the described problems by concentrating on finding user interface and application features to help ontological engineering. As the initial step of this research work, a survey of existing tools was made to acquire an overview of existing solutions and to find the basic features required for constructing ontologies. Combined with requirements set forth by two different VTT projects, a core set of editor features was eventually designed. In order to try out these ideas in practice, and to provide a tool for other projects' use, a prototype application called CONE was developed. The first incarnation of CONE technology was introduced in the form of an ontology graphical representation component (referred as PRE-CONE) used in the ABS project. The second, more advanced prototype was aimed at generic ontology engineering work, and was test-driven in context of natural language understanding and multilingual information retrieval.

The contribution of this study to a general understanding of how to create and manage extensive ontologies are the following support method ideas:

- *Graphical representation*. Visualisation of ontologies in the form of a graph is a natural metaphor for ontologies, and can make the complex structures easier to understand and manipulate.

- *Direct manipulation editing tools*. The user is in control of interaction: the model can be accessed directly and the effects of the action can be immediately seen. The process of creating an ontology is as easy and intuitive as assembling a graph with a mouse.

- *Clustering*. Extensive ontologies are a heavy load for the user to look at and understand, as well as for the computer to load from a persistent storage and render on the screen. By dividing concepts into discrete clusters, several benefits may be achieved: large models may be loaded incrementally, saving machine processing time and memory, as well as reducing user's cognitional load.

- *Multiple perspectives*. Multiple viewpoints connected with bridges can be used to provide different perspectives (level of detail, emphasis, classification, etc.) to the same information.

Even though a lot of work has been made for this research, the results are still more or less preliminary. The two case studies presented in this thesis have been good developmental aids, and have provided a plenty of valuable ideas concerning technical implementation, user interface, and the feature set of CONE. However, neither of them has been complete enough to thoroughly validate the support methods proposed. Both studies were performed in an uncontrolled way, and with limited test audience.

The major problems of the case study performed in the context of the ABS project are the following:

- *Different focus*. The case study was performed within the ABS broker evaluation which focus was the entire ABS system, not only the graphical representation method. Majority of the comments received were related to the ABS functionality, and the amount of comments directly concerning PRE-CONE was too small to draw any certain conclusions. It was also difficult to determine the exact target of user comments (e.g. "Locating particular item on the broker view is sometimes difficult". Is it because of the possible defect in the graphical representation or is it because the content provided by the broker is ambiguous?).

- *Limited test cases*. The test cases were too generic, and did not measure any particular feature of the application. The tasks presented to the users should have been more carefully planned to measure one particular feature, and there should have been more cases that specifically target CONE features.

- *Only subset of the functionality was tested*. The evaluation applied only to the graphical representation which is a single feature of CONE.

- *Small amount of data*. The conceptual network used in the ABS test was too small to effectively measure how the graphical representation methods work with large graphs.

The second case study, covering the use of CONE in Webtran project, was more exhaustive what comes to features. However, it also had its limitations:

- *Limited scope*. There was only one person using and commenting CONE. It is difficult to distinct personal working methods, likes and dislikes, wishes, and ideas from errors or design flaws. To get feedback that can be used as a base for improvements that serve a larger user community, more test users are needed.

- *Uncontrolled test*. The test was based on the test user's independent use of CONE. The feedback received from the user is subjective, and reflects only those problems discovered and realised by the user herself. By planning and controlling the test situation, the test can be made more complete to cover all features desired.

- *No observation*. The evaluation was based solely on the test user's experiences with CONE. The interview did reveal some major issues, but cannot possibly be used to compensate user observation. By observing the user as she is working, the evaluator may discover such problems that would have never been realised by the user, or flaws that the user would not even consider as problems.

- *No anonymous feedback*. All feedback was given directly from the test user to the developer. This typically emphasises positive issues and limits the amount of negative feedback. This problem can be alleviated by having more test users and a method for anonymous feedback, or a non-developer personnel carrying out the evaluation.

Considering these issues, no final conclusion about the feasibility of the support methods can be drawn. In order to test and evaluate them more carefully, complete user testing with more users, larger amount of data, and more carefully planned test cases should be organised. Separate usability tests could be organised to validate the user interface, and technical tests to measure performance and functionality.

Continuing research on ontologies has brought up several new ideas to be considered. Practical implementation work of CONE has revealed existing problems that need more attention or polishing, and evaluation has brought up problems that may require some of the CONE foundations to be reconsidered.

Most of the concerns in current implementation are affiliated with the application design model and the way it is represented by the program. One of the key goals in developing CONE was to provide as simple design model as possible, without sacrificing any capabilities relevant to the theory of ontologies. The current design model can be regarded as simple and elegant, but there is a question whether its expressive power is adequate. For example, it is quite difficult to combine all concepts linked together using bridges to a supeconcept that represents all existing perspectives. On the other way around, it is difficult for the user to determine if the concept he has in mind already exists in some other viewpoint.

As seen in CONE evaluation in Webtran project, there are still some problems involved with the use of clusters. Using the current implementation, it is possible that the user has problems in finding desired concepts, and sometimes clusters are strayed so that they are impossible to retrieve. To allow the user to find certain entities more easily, a method that abstracts clusters rather than hides them should be considered.

To take CONE one step further, some kinds of inference capabilities could be implemented. The current implementation treats entities in the ontology as plain objects, mostly ignoring their semantics. This deficiency can be seen in the way relations are handled — they are all are treated in uniform manner, even though they often represent clear, well-specified ideas. For example, is-a relation connecting parent concept to a child concept could automatically cause properties of a parent concept to be inherited by the child concept. For the user, and for the utilisation of ontologies created with CONE, this kind of automatic inference would provide new possibilities.

To continue this research and to validate the applicability and usability of support methods of ontological engineering, some further actions must yet be taken. Large-scale user trials with real users, real use cases, and real data must be carried out and the results must be carefully analysed and evaluated. This is the most appropriate way to assess ideas presented in this thesis as a whole. Fortunately, VTT Information Technology's continuing interest towards ontologies provides an excellent opportunity for this.

7 References

Basili, V. R. 1992. *Software Modeling and Measurement: The Goal/Question/Metric Paradigm*. University of Maryland, College Park, MD. Computer Science Technical Report Series, CS-TR-2956 (UMIACS-TR-92-96).

Bray, T., Paoli, J., Sperberg-McQueen, C. M. 1998. *Extensible Markup Language (XML) 1.0*. W3C Recommendation, February 10th 1998. Available at: http://www.w3.org/TR/1998/REC-xml-19980210.

Brickley, D., Guha, R. 1999. *Resource Description Framework (RDF) Schema Specification.* W3C Proposed Recommendation, March 3rd 1999, Available at: http://www.w3.org/TR/PR-rdf-schema/.

Casulli, S., Charbilas, C., Kankaanpää, T., Karetsos, G., Katsoulas, D., Laikari, A., Moreira, M., Medrouk, L., Tothezan, I., Väisänen, M., Villagra, V. 1998. *D43 Broker version 2 prototype documentation and user guide*. ABS project documentation, CEC deliverable number A206TELPJEDSI043b0.

Chandrasekaran, B., Josephson, J. R., Benjamins, V. R. 1998. *The Ontology of Tasks and Methods*. Proceedings of KAW'98: Eleventh Workshop on Knowledge Acquisition, Modeling and Management, Alberta, Canada, April 18-23, 1998.

Chomsky, N., 1957. *Syntactic Structures*. Mouton. The Hague.

Cuena, J., Molina, M., 1996. *Building Knowledge Models using KSM*. KAW'96: Tenth Workshop on Knowledge Acquisition, Modeling and Management, Gaines, B., Musen, M. (ed.), Alberta, Canada, November 8-14, 1996.

NCITS.T2/98-003. 1998. *Conceptual Graphs: A presentation language for knowledge in conceptual models*. Proposed Draft. National Committee for Information Technology Standards (NCITS). Available at: http://concept.cs.uah.edu/CG/Standard.html

Eklund, P., Leane, J., Nowak, C. 1993. *GRIP, Toward a Standard GUI for Conceptual Structures*. Proceedings of PEIRCE Workshop (ICCS'93), Ellis, G., Levinson, B. (ed.), Quebec City, Canada, August 1993.

Euzenat, J. 1993. *On a purely taxonomic and descriptive meaning for classes.* Proceedings of the IJCAI workshop on Object-based representation systems. Chambéry, France, 1993.

Finin, T., Weber, J., Wiederhold, G., Genesereth, M., Fritzson, R., McKay, D., McGuire, J., Pelavin, P., Shapiro, S., Beck, C. 1992. *Specification of the KQML Agent Communication Language*. Technical Report EIT TR 92-04, Enterprise Integration Technologies, Palo Alto, CA.

Fowler, M., Scott, K. 1997. *UML Distilled: Applying the Standard Object Modeling Language*, Addison-Wesley Object Technology Series.

Gaines, B. R. 1991, *An Interactive Visual Language for term Subsumption Languages*. IJCAI-91, Sydney, Australia.

Genesereth, M., Nilsson, N. J. 1987. *Logical foundations of artificial intelligence*. Morgan Kaufmann, Los Altos, CA . 1987. 405 p.

Genesereth, M., Fikes, R. 1992. *Knowledge Interchange Format, Version 3.0 Reference Manual*. Technical Report Logic-92-1. Computer Science Department, Stanford University. Available at: http://www-ksl.stanford.edu/knowledge-sharing/kif/#manual.

Gruber, T. R. 1993, *A Translation Approach to Portable Ontology Specifications*, Knowledge Acquisition, vol 5(2): 199-220.

Guarino, N. 1995. *Formal Ontology, Conceptual Analysis and Knowledge Representation*. International Journal of Human and Computer Studies, 43(5/6): 625-640.

Guarino, N. 1998a. *Formal Ontology and Information Systems*. Formal Ontology in Information Systems, Guarino, N. (ed.). Proceedings of FOIS'98, Trento, Italy, 6-8 June 1998. Amsterdam, IOS Press.

Guarino, N., Masolo C., Vetere G. 1998b. *OntoSeek: Using Large Linguistic Ontologies for Gathering Information Resources from the Web*. LADSEB-GNR Int. Rep. 02/98, March 1998.

Heflin, J., Hendler, J., Luke, S. 1998. *Reading Between the Lines: Using SHOE to Discover Implicit Knowledge from the Web*. Proceedings of Fifteenth National Conference on Artificial Intelligence (AAAI-98). Workshop on AI and Information Integration. July 26-30, 1998, Madison, Wisconsin.

Jonker, C., Kremer, R., van Leeuwen, P., Pan, D., Treur, J. 1998. *Mapping Visual to Textual Representation of Knowledge in DESIRE*. Proceedings of KAW'98: Eleventh Workshop on Knowledge Acquisition, Modeling and Management, Alberta, Canada, April 18-23, 1998.

Kent, R. 1998. *Ontology Markup Language* (unpublished). Available at: http://wave.eecs.wsu.edu/CKRMI/OML.html.

Kremer, R. 1998. *Visual Languages for Knowledge Representation*. Proceedings of KAW'98: Eleventh Workshop on Knowledge Acquisition, Modeling and Management, Alberta, Canada, April 18-23, 1998.

Lehtola, A., Tenni, J., Bounsaythip, C. 1998. *Definition of a Controlled Language Based on Augmented Lexical Entries*. Proceedings of the Controlled Language Applications Workshop 98, Carnagie Mellon, Pittsburg, USA, 21-22 May, 1998, pp. 16-29.

Lindsay, P., Norman, D. 1977. *Human Information Processing, An Introduction to Psychology*. Second Edition. Orlando, Florida. Academic Press Inc. 777 p.

Luger, G., Stubblefield, W. 1998. *Artificial Intelligence, Structuries and Strategies for Complex Problem Solving*. Third Edition. Addison-Wesley Longman Inc. 824 p.

Mahalingam, K., Huhns, M. 1998. *Ontology Tools for Semantic Reconciliation in Distributed Heterogeneous Information Environments*. Intelligent Automation and Soft Computing: An International Journal on Distributed Intelligent Systems, Kamel, M., Jamshidi, M., (ed.). 1998.

Nosek, J. T., Roth, I. 1990. *A Comparison of Formal Knowledge Representation Schemes as Communication Tools: Predicate Logic vs. Semantic Network.* International Journal of Man-Machine Studies 33: 227-239.

Poulain, G., Athanassiou, E., Hoang Van, A., Kondakji, A., Bouladoux, J-M., Väisänen, M., Heikkonen, I., Ovaska, P. 1999. *WP 6 Final Evaluation Report*. ABS project documentation. CEC deliverable number A206BERB5DRP062b0.

Preece, J. 1994, *Human-computer interaction*, Addison-Wesley publishing company. 775 p.

Shneiderman, B. 1997. *Direct Manipulation for Comprehensible, Predictable, and Controllable User Interfaces*. Proceedings of IUI97, 1997 International Conference on Intelligent User Interfaces, Orlando, FL, January 6-9, 1997, 33-39.

Skuce, D. 1995. CODE4: *A Unified System for Managing Conceptual Knowledge*. International Journal of Human-Computer Studies (1995), 42: 413-451.

Sowa, J. F. 1984. *Conceptual Structures, Information Processing in Mind and Machine*, Addison-Wesley publishing company. 481 p.

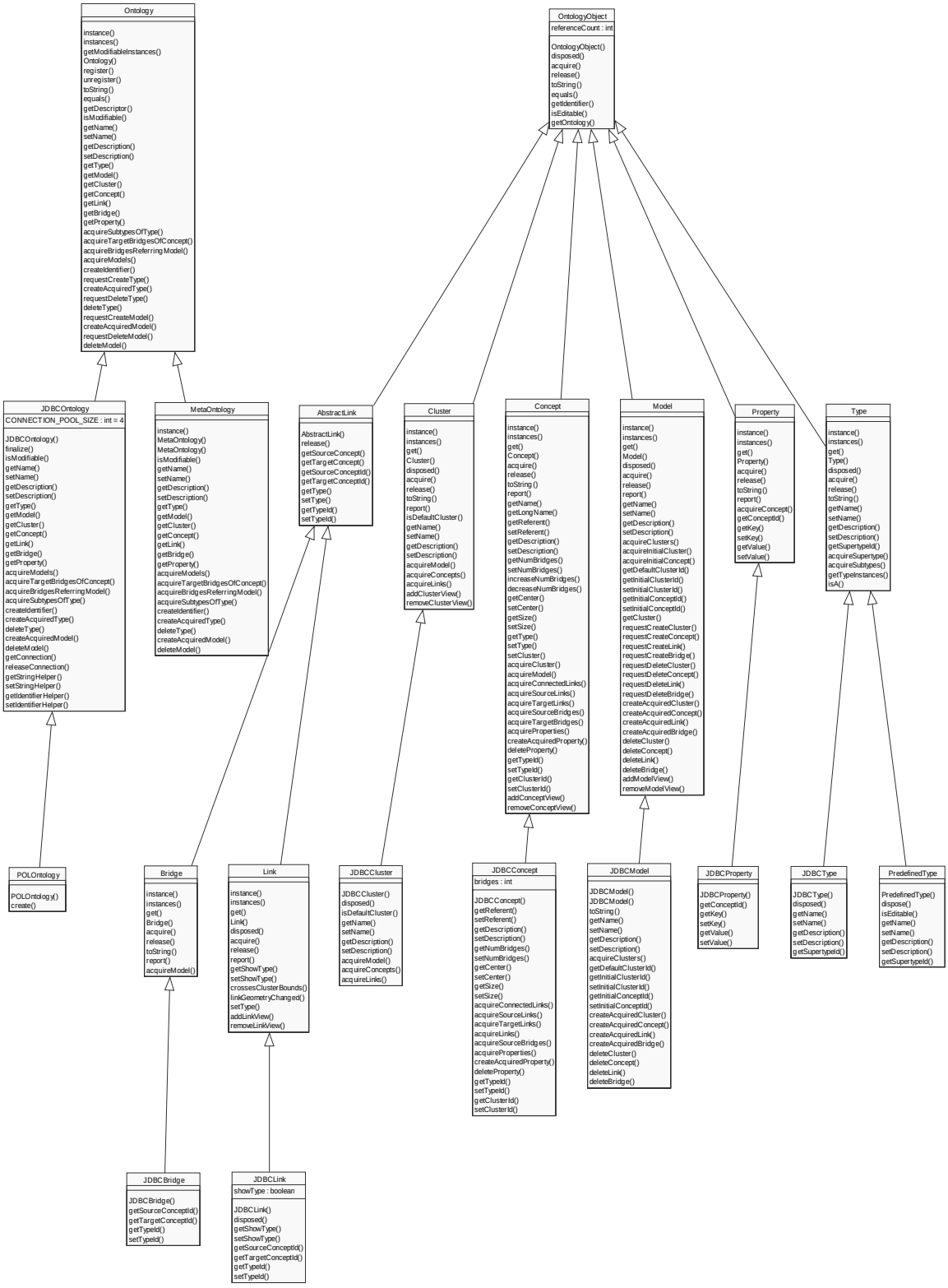
Swartout, B., Patil R., Knight K. Russ T. 1996. *Towards Distributed Use of Large-Scale Ontologies.* Proceedings of the 10th Knowledge Workshop (KAW'96). Alberta, Canada, Nov. 9-14, 1996.

Taveter, K. 1998. *Intelligent Information Retrieval based on Interconnected Concepts and Classes of Retrieval Domains*. Eigth DELOS Workshop on User Interfaces for Digital Libraries. Stockholm, Sweden, 21-23 October, 1998.

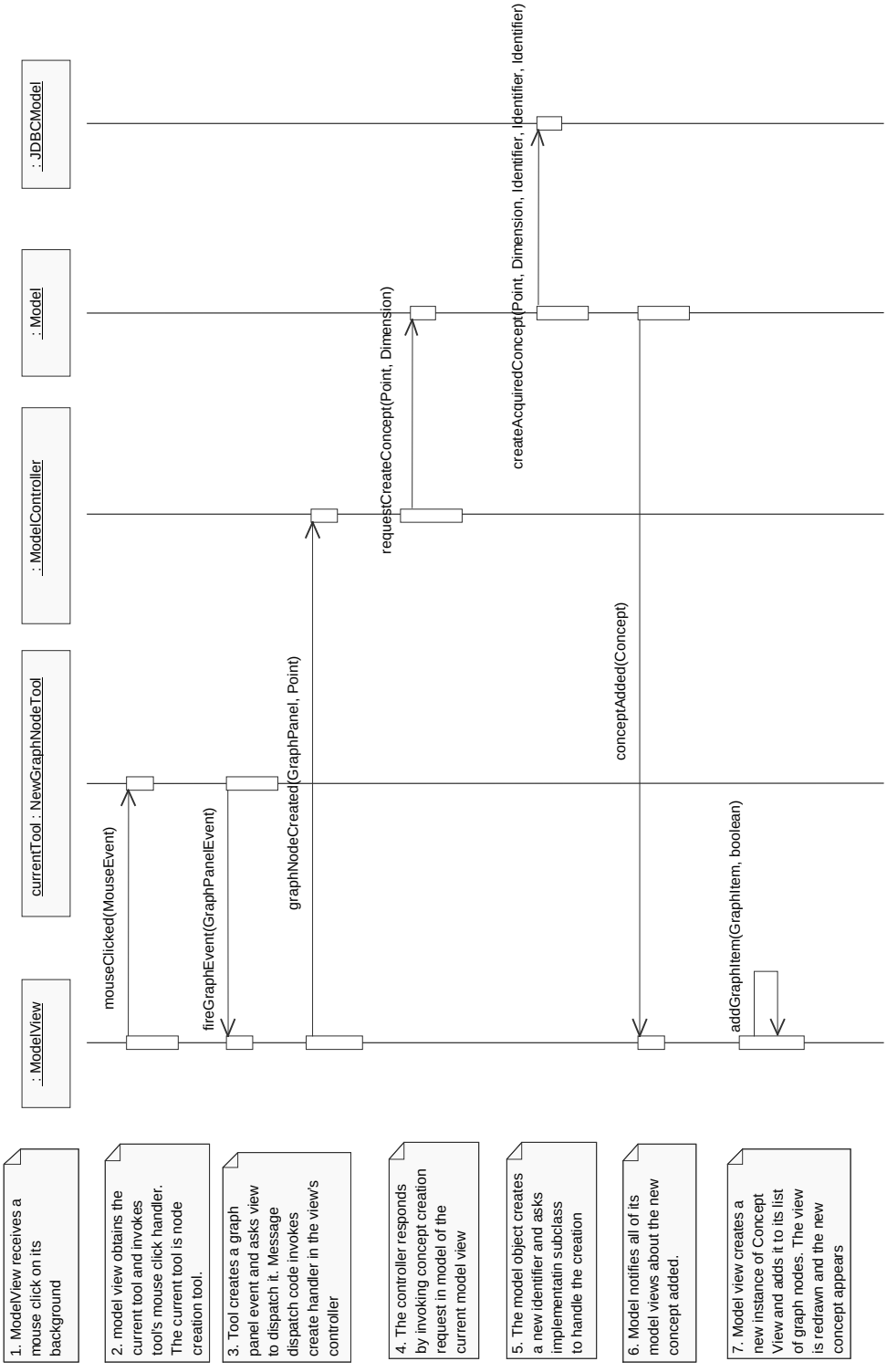
Uschold, M., Gruninger, M. 1996. *Ontologies: Principles, Methods and Applications.* The Knowledge Engineering Review, 11 (2): 93-136. 1996.

Uschold, M., King, M., Moralee, S., Zorgios, Y. 1998, *The Enterprise Ontology*. The Knowledge Engineering Review. vol.13: 1. March 1998.

Appendix A: Model classes



Appendix B: Concept creation sequence



Appendix C: Interview questions

Background and CONE Usage

1. Do you have any prior experience of ontology tools? If yes, which tool and how long have you used it?

2. For what purpose have you been using CONE? What is the domain you have been modelling?

3. What is the content of your models and why did you create them?

4. During the test period, approximately how many hours have you been working with CONE?

5. During that time, how many models did you create?

6. How would you estimate the size of the models you created? How many concepts does your typical model have? How about the largest and the smallest model you created?

7. Did you use CONE to try out various models, or did you design the model first and then input it to CONE?

8. In your opinion, what is the best and the worst feature of CONE? Why?

Graphical representation

1. How would you describe CONE's way of representing ontologies? Is it clear or disorganized? Is it easy to understand?

2. Do you recall having any problems understanding the diagrams presented by CONE?

3. When you started using CONE, was the meaning of all objects on the screen obvious to you? Did you understand their purposes and see the connection between them and the theory of ontologies?

4. Thinking only the graphic design of the model view, how would you comment on that? Did you have any problems reading text, seeing concepts, relations or bridges? Are the colors appropriate? How would you improve it?

5. Is the amount of detail adequate for your purposes? Would you like to have something added or removed? What?

6. What kind of benefits do you see in using different zoom levels when viewing ontologies?

7. Have you used multiple windows to display multiple views to the same model? Why?

Editing tools

1. Considering editing tools of CONE, do you think it is easy to move and resize nodes?

2. Do you use selection marquee to select multiple nodes?

3. Do you think that the toolset provided by CONE is adequate? Do you miss something? Have you used them all?

4. Last time you used CONE, did you notice that you tried to apply a wrong tool and you had to switch to a correct tool before proceeding?

5. Last time you used CONE to create a model, how long did it approximately take? How large was the model?

6. Are you satisfied with CONE's tools for editing properties and attributes of a concept? Is the dialog-based interface adequate, or should editing be available directly in the model view?

7. How often do you reorganize the graph layout? Do you often try to make crossing relation edges disappear? Would you like to have this action automatized?

Clusters

1. Do you use clusters to group concepts? What is the criteria you use to classify concepts?

2. In the models you have created recently, how many concepts do you typically have in one cluster? How many clusters?

3. Please estimate, how many concepts would you put in a model if clusters were not available?

4. Is the loading time for clusters adequate? When you click at control to expand the cluster, is the response fast enough?

5. In your point of view, are there any problems with using clusters? Does the use of clusters bring any disadvantages or restrictions?

Viewpoints and bridges

1. Do you use parallel models to represent different viewpoints of some ontology? What is the basis of your classification method?

2. In the models you have created during the test period, how many viewpoints and bridges did they approximately have?

3. Is the graphical representation of bridges adequate?

4. Do you use bridges primarily as a means of organization, classification or navigation? Why?