Graphic Representation and Visualisation as Modelling Support for the Knowledge Acquisition Process

By

ANNE HÅKANSSON

Division of Computer Science
Department of Information Science
Uppsala University
SE-751 20 Uppsala

Email: anne.hakansson@dis.uu.se

Uppsala 2003
The national Graduate School for Human-Machine Interaction, supported by the Swedish Foundation for Strategic Research aims to supply Swedish industry with interdisciplinary front edge knowledge in order to develop efficient, safe and user-friendly IT-products and real time systems.
Dissertation in Computer Science presented at Uppsala University to be publicly examined in Hörsal 2, Ekonomikum, Uppsala, Friday, January 16, 2004 at 13:15 for the degree of Doctor of Philosophy. The examination will be conducted in English.

ABSTRACT


The thesis describes steps taken towards using graphic representation and visual modelling support for the knowledge acquisition process in knowledge-based systems — a process commonly regarded as difficult. The performance of the systems depends on the quality of the embedded knowledge, which makes the knowledge acquisition phase particularly significant. During the acquisition phase, a main obstacle to proper extraction of information is the absence of effective modelling techniques.

The contributions of the thesis are: introducing a methodology for user-centred knowledge modelling, enhancing transparency to support the modelling of content and of the reasoning strategy, incorporating conceptualisation to simplify the grasp of the contents and to support assimilation of the domain knowledge, and supplying a visual compositional logic programming language for adding and modifying functionality.

The user-centred knowledge acquisition model, proposed in this thesis, applies a combination of different approaches to knowledge modelling. The aim is to bridge the gap between the users (i.e., knowledge engineers, domain experts and end users) and the system in transferring knowledge, by supporting the users through graphics and visualisation. Visualisation supports the users by providing several different views of the contents of the system.

The Unified Modelling Language (UML) is employed as a modelling language. A benefit of utilising UML is that the knowledge base can be modified, and the reasoning strategy and the functionality can be changed directly in the model. To make the knowledge base more comprehensible and expressive, we incorporated visual conceptualisation into UML’s diagrams to describe the contents. Visual conceptualisation of the knowledge can also facilitate assimilation in a hypermedia system through visual libraries.

Visualisation of functionality is applied to a programming paradigm, namely relational programming, often employed in artificial intelligence systems. This approach employs Venn-Euler diagrams as a graphic interface to a compositional operator based relational programming language.

The concrete result of the research is the development of a graphic representation and visual modelling approach to support the knowledge acquisition process. This approach has been evaluated for two different knowledge bases, one built for hydropower development and river regulation and the other for diagnosing childhood diseases.

Keywords: Artificial Intelligence, Knowledge-Based Systems, Knowledge Acquisition, Graphic Representation, Visualisation, Unified Modelling Language, Modelling Rule-Based Systems, Visual Programming, Declarative Programming, Information Networks, User-Centred Design, Knowledge Interface

Anne Håkansson, Division of Computer Science, Department of Information Science, Uppsala University, Kyrkogårdsg. 10, Box 513, SE-751 20 Uppsala, Sweden

© Anne Håkansson 2003

ISBN 91-506-1727-3

urn:nbn:se:uu:diva-3812 (http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-3812)

Printed in Sweden by University Printers, Uppsala 2003
This thesis is based on the work conducted for the following papers:


Contents

Thesis overview ........................................................................................................... 1
1.1 Introduction ........................................................................................................... 1
1.2 Methodology ..........................................................................................................3
1.3 Scientific contribution ...........................................................................................7
  1.3.1 Creating a Methodology for User-Centred Knowledge Modelling .................. 8
  1.3.2 Enhancing the Transparency of the Contents of the Knowledge Base .............. 9
  1.3.3 Conceptualising to Simplify Grasping the Contents of the Knowledge Base .......... 10
  1.3.4 Enhancing Transparency and Supporting Modification of the Reasoning Strategy .............................................................. 11
  1.3.5 Adding and Modifying Functionality through a Visual Compositional Logic Programming Language ........................................... 11
  1.3.6 Supporting Assimilation of Domain Knowledge ........................................... 12
1.4 Structure of the thesis ..........................................................................................13
1.5 Articles ................................................................................................................16
2 Fundamental topics in Knowledge-Based Systems .................................................19
  2.1 Systems containing Knowledge .........................................................................20
    2.1.1 Knowledge and Reasoning Strategies ...................................................... 20
    2.1.2 Systems ..................................................................................................... 21
  2.2 Knowledge-Based Systems ................................................................................ 21
  2.3 Knowledge Engineering .................................................................................... 22
  2.4 Knowledge Acquisition ..................................................................................... 23
    2.4.1 Elicitation .................................................................................................. 24
    2.4.2 Modelling .................................................................................................. 26
    2.4.3 Formalising Information into a Knowledge Representation ....................... 27
    2.4.4 Recognised Problems ................................................................................. 28
3 Graphic Representation and Visualisation as Modelling Support for the Knowledge Acquisition Process ................................................................................. 33
  3.1 An Approach to a User-Centred Design Methodology ...................................... 36
    3.1.1 Related Work ............................................................................................. 37
    3.1.2 User-Centred Knowledge Modelling ....................................................... 39
  3.2 Graphic Modelling .............................................................................................. 41
    3.2.1 Related Work ............................................................................................. 41
To my son Andreas
and my parents, Eva and Erik
ACKNOWLEDGEMENTS

Many people have been supportive during my work with this dissertation. First and foremost, I would like to thank my supervisors Professor Andreas Hamfelt and Professor Åke Hansson for their contributions and support. Professor Åke Hansson dedicated his time and effort at the beginning of the research work and Professor Andreas Hamfelt gave his valuable time and effort at the end providing guidance and constructive suggestions.

I like to express my thanks to people at the Department of Numerical Analysis and Computing Science (NADA), Kungliga Tekniska Högskolan, KTH, Stockholm University for helpful advice and for their working time. In particular, Professor Kerstin Severinson-Eklundh and Dr. Ann Lantz for supporting me while I was associated as a Ph.D. student and for giving me the opportunity to attend the Ph.D. courses at the Graduate School for Human-Machine Interaction. Moreover, I am very grateful to Dr. Robert Ramberg at the Department of Computer Sciences, DSV, Kista Stockholm, for an interesting day spent with him there discussing mental models.

I am indebted to my colleague Fil. Lic. Lars Oestreicher for numerous pieces of advice, encouragement and different kinds of support. I am most grateful for his role as an informal supervisor, for his patience in reading drafts and for useful discussions, especially concerning the Human-Computer Interaction. His role as a supervisor when writing Article “Dynamic libraries as a means for the management of information browsing in networks” will not be forgotten. Fil. Lic. Lars Oestreicher has engaged himself in my research and his continuous and very competent advice has really improved my work. Thank you!

I am highly grateful to my colleague Dr. Anneli Edman. I must not forget the valuable assistance with reading drafts and her role as a supervisor when writing Bachelor Thesis, Master Thesis and when writing Article “The KANAL-system, an intelligent rule editor supporting the knowledge acquisition process”. She has also been given numerous sound recommendations for the dissertation. Furthermore, I would like to express my thanks to Dr. Mats Cedvall for supporting my work with the Unified Modelling Language, UML, for reading drafts and for providing important perspectives. I am highly grateful to Cynthia Jimes for reading drafts and making comments during the work with the articles and the thesis. I am also grateful to Zeynep Kiziltan and Magnus Larsson for making comments during the work with the thesis. Your help will not be forgotten!

Professor Larry Lucardie has engaged himself in my research and his competent recommendations have improved the dissertation in several respects. Moreover, I am grateful for fruitful discussions with him about Knowledge Based Systems. I would like to thank the domain expert Professor Lennart Strömquist who worked hard during the development of
the Environmental Impact Assessment (EIA)-system and supplied all the
domain knowledge used by it. I must also thank him for his devotion to
support the end user to provide a useable educational expert system.

I am grateful to Charlotta Öijer and Annika Widmark for our fruitful and
inspiring collaborations during the work with the EIA-system and the
KANAL-system, respectively. I am also grateful to Eva Helenius for
discussions on incorporating UML with expert systems, Torsten Jonsson for
 collaboration in the CombiLog project, and Tomas Andersson for
implementing the ViCoLL interface. Moreover, I would like to thank some
other present and former Ph D students, Docent Brahim Hnich, Maria
Hägglund, Narin Mayiwar, Inger Persson, Isabella Scandurra, Tomas
Snäckerström, and Monica Tavanti. It has been a pleasure working with
these people. Moreover, I would like to express my thanks to Leif
Håkansson and Inger Holmberg for their support and for reading drafts
during their holidays. In addition, I would like to thank those people who
participated in the investigation with the KANAL-system versus Wisdom.

My thanks are also due to all the former directors of studies Torsten Palm,
Ingrid Klint and Narin Mayiwar, for providing me with a teaching position
and all the other teachers, Dr. Agneta Granskog-Eriksson, Dr. Anneli
Edman, Fil. Lic. Lars Oestreicher, Helena Dominique, Dr. Mats Cedwall,
Dr. Anna-Lena Johansson, and Dr. Mats Nordström, who have helped me to
improve my teaching and been supportive. I would also thank the former
head of the department Prof. Anders Christoffersson and the director of post-
graduate studies Prof. Anders Ågren and Docent Vladislav Valkovskyy for
believing not only in me, but also in my work. Furthermore, I want to thank
the administrative and technical staff, Ann Skogh-Gunnarsson, Anna-Lena
Torstensson-Kåberg, Lars-Göran Svensk, Gunilla Klaar, Eva Enefjord,
Ingrid Lukell and Pierre Hjälm for continuous support and for taking care of
all that extra work. Furthermore, I would like to say a big thank you to other
colleagues at the Department of ADB and Datalogi, as well as colleagues at
the ADB and Datalogi and the Division of Computer Science (DSV) and
staff at the Department of Information Science for their engagement and
encouragement. And besides these people, I would like to thank all the
people at DIS for forming an enjoyable working environment despite
difficulties. It has been a pleasure working in such a department. I wish to
thank all of those, too numerous to name, who have contributed to my work.

Finally, my heartfelt thanks go out to my son, Andreas, and my parents,
Eva and Erik Håkansson, for their tremendous patience, continuous support
and other resources. Without the support of my son and my parents, this
thesis would have never been realised. I would like to express my thanks to
each and every family member, my brother Leif and his family, Marielle,
Michelle and Marcus, my relatives, and our dear family friends, for their
support. Moreover, my thanks go to all my friends for their understanding
and patience over the last couple of years.
Thesis overview

1.1 Introduction

Developing a software system in the field of artificial intelligence demands the development of a system that performs feats that normally require human intelligence. The human knowledge involved is usually transferred through a language, via a model, providing either an informal or a formal representation of the system (Buchanan et al., 1983; Hayes-Roth et al., 1983; Waterman 1986; Lenat & Guha, 1991; Durkin, 1994; Jackson, 1999).

The stages of the development of a system containing knowledge is a tightly connected process where all the stages are iterated until the system is acceptable for use (Hayes-Roth et al., 1983; Buchanan et al., 1983; Mcgraw & Harbisson-Briggs, 1989; Peterson et al., 1990; Jackson, 1999; Hoppe & Meseguer, 1993; Beauvieux, 1990; Peterson et al., 1990; Polat & Guvenir, 1993). During the development of a knowledge-based system, the knowledge acquisition is the most crucial step to obtain usable systems. Knowledge acquisition comprises the eliciting, modelling and encoding of domain knowledge. Eliciting knowledge means acquiring it from a domain expert. Modelling knowledge means structuring it into some form of knowledge representation. And, finally, encoding refers to the transfer of the modelled knowledge into and its implementation in a working computer system. To ensure the usefulness of the system, the system is tested for usability through verification and validation of the knowledge base. However, problems arise in each activity. For example, in conducting the elicitation, there may be problems with communication between the knowledge engineer1 and the domain expert2, and in modelling there may be a mismatch between the experts’ expressions and the symbolic representation in the system.

The high costs and the problems associated with knowledge acquisition (Beauvieux, 1990; Peterson et al., 1990; Polat & Guvenir, 1993; Mcgraw &

---

1 The knowledge engineers are the persons who usually develop knowledge-based systems.
2 The domain expert is the person who has the expertise within the domain currently under development.
Harbisson-Briggs, 1989; Sandahl, 1992; Alonso et al., 2000) clearly motivate the development of tools to support the major parts (if not the whole process) of knowledge acquisition. Such a tool should give users the opportunity to focus on extracting and modelling the domain knowledge instead of concentrating on the encoding of the knowledge into the available knowledge representation. One such approach is to display the domain knowledge by using graphical representation and visualisation.

This thesis proposes a graphic representation and visual modelling approach to support the knowledge acquisition process. More specifically, it puts forward proposals for:

- how to incorporate the two types of users, i.e., domain experts and end users, in the modelling by supporting a user-centred means of transferring knowledge between the human and the computer model; this is achieved by using graphic representations of the knowledge.
- the use of an established graphic representation scheme in new areas of knowledge representation and presentation.
- the use of graphic representations of declarative knowledge for program development and as a support for the navigation of the hyperlink structure in networked information structures, such as the World Wide Web.

A common denominator for all three of these proposals is the strong emphasis on visual representation and presentation of knowledge, aiming at the construction of useful and expressive conceptual models of knowledge. The proposal is to enhance system transparency and to support the alteration of the contents of the knowledge base by describing the contents of the knowledge base graphically, e.g., using Unified Modeling Language (UML). Furthermore, the aims of the of the proposal was to reduce the problem with complexity of the knowledge base and support the alteration or amendment of reasoning strategies in a knowledge-based system by means of conceptualisation. This conceptualisation is intended to improve the semantic understanding of the formalisation of domain knowledge, exemplified through the use of UML diagrams. Moreover, with the means of the proposal it is possible to enhance the transparency of the reasoning strategy and support the alteration of this strategy by using graphical description. Additionally, the proposal aims to facilitate changing the functionality of the system and facilitates the program development in a compositional declarative relational programming language by visualising program composition operations and components reflecting the set theoretical semantics of the language.
1.2 Methodology

The following topics have proven useful for the work with graphic representation and visualisation modelling support in the knowledge acquisition process. The main subject area in this thesis covers the different knowledge engineering methods, with the focus being on techniques for evolutionary or incremental development (Hayes-Roth et al., 1983; Patterson, 1990; Sommerville, 1996; Sommerville, 2001; Durkin, 1994; Peterson et al., 1990; Gonzales & Dankel, 1993), and model-based knowledge engineering (Neubert & Studer, 1992; Wielinga et al., 1992; Gruber, 1993). Within these areas, in particular, the knowledge acquisition process which encompassing elicitation, modelling and formalisation, is known to be difficult (Durkin, 1994; Tallis et al., 1999; Sandahl, 1992), which was also found in our work (Håkansson & Öijer, 1994).

The design of intelligent user interfaces includes a variety of topics ranging from the application of artificial intelligence and knowledge-based techniques to issues of human-computer interaction. The development of intelligent user interfaces aims to help users to access complicated information structures or to solve complex tasks (Shneiderman, 1997; Kim & Gil, 2000; Waern, 1997; Mayer, 1983; MITECS, 2002). A range of users may be utilising a knowledge-based system to perform a variety of tasks and, therefore, a combination of dialog styles is often appropriate for the interaction (Mayhew, 1992; Shneiderman, 1998; Wærn, 1997; Preece & Keller, 1990; Macaulay, 1995; Lantz, 1996; Kahn, 1996; Sebillo et al., 1998). User interfaces are important in knowledge-based systems and it seems to be a general agreement on the successfulness of systems that are relying on these (Darlington, 2000). Hence, these will play a significant part in this thesis.

The research presented here is founded on the concept of declarative programming language as an interpreter of a knowledge base, with a hypermedia system providing a supportive user interface (Woodhead, 1991; Håkansson & Öijer, 1993). One immediate benefit from our use of declarative programming languages is that they support the users in implementing knowledge as antecedents-consequents, which is a representation close to our own reasoning mechanisms (Anderson, 1990). The users can state that some relation is the case and that certain may constraints apply to these relations. These statements, i.e., the ones known to be true, are declared in the programming language as expressions by the users and not in term of how the application derives its consequences from the facts. Moreover, the statements do not have to be composed in any particular order, and if only small adjustments are needed, the statements can be changed and extended very easily (Genesereth & Nilsson, 1988).
Additionally, logic programming languages are particularly well suited for rapidly prototyping and allows iterative developments, which is a common method for developing knowledge-based systems (Peterson et al., 1990.). These advantages are the main reasons why we have used a declarative programming language approach in our work.

We developed an expert system, called EIA-system, which uses texts and pictures in the menu interface design (Mayhew, 1992) and for direct manipulation of the interface (Shneiderman, 1998) in order to support the end users (Håkansson & Öijer, 1993). To reduce the amount of errors, the end users are, e.g., guided through consultation and the number of answer alternatives is limited. The research is also rooted in incremental development through the use of intelligent editors with natural language processing (KANAL-system) developed by us (Håkansson & Widmark, 1996). This system was developed as a support for the domain experts to develop rule-based systems. Furthermore, it draws on previous empirical work on knowledge-based systems (Stylianou et al., 1995) in the form of a usability test (Nielsen, 1993; Rubin et al., 1999).

To find out if shells, with a special focus on the KANAL-system, induce problems with the interface, domain knowledge and reasoning strategies and to find weaknesses and conceivable solutions, a usability study was conducted to compare two different shells, the KANAL-system (Håkansson & Widmark, 1996; Håkansson et al., 2000) and Wisdom (Lucardie, 1998; Dignum, 1999) (Håkansson, 2002). The intended usages of the compared shells are rather similar because they are both used to build diagnosis systems through a declarative programming language, but the user interfaces differ in terms of the knowledge presentation. One of the shells implements a dialogue-controlled interaction (KANAL-system), and the other one uses decision-tables and frames (Wisdom).

To measure the usability, there are different methods for collecting information, for example, observations, interviews, thinking aloud, constructive interaction, and coaching (Nielsen, 1993). Several different methods, such as observations and interviews, can be used together in a usability study because they supplement each other and the advantages and disadvantages of the methods partly make up for each other (Nielsen, 1993). Through observation studies, quantitative data is collected and qualitative data is gathered in interviews sessions. However, neither of these methods is perfect for extracting data about mental models and therefore, a task has been assigned to the subjects in order to extract these models. In this study the task was to draw a picture that corresponds to the subjects’ own image of the contents of the system after implementing the domain knowledge.
The methods selected for collecting information were observations\(^3\) (six subjects), interviews\(^4\) (twelve subjects) and a drawing task to examine mental models\(^5\) (twelve subjects) (Håkansson, 2002). The number of subjects was fixed in advance since the groups had to be homogeneous, both in experiences and in skills, and these groups had to be representative for both systems to assure both reliability\(^6\) and validity\(^7\). Hence, three groups worked with each of the systems and all were interviewed separately, and eventually asked to draw a picture of their view of the contents of the system. To get a reliable result in this investigation, repeated tests with both systems were performed with different test users. Additionally, the same issues can be checked in the different tests. To get a valid result, neither the observation nor the interview could be too long. By including time constraints in the interview scheme, the problem with frustrated subjects who answer the questions with short and non-relevant answers just to get the interview over with, was avoided. Furthermore, to minimise problems with interruptions, neither the observation nor the interview took place just before other appointments of the test users.

To wrap up the results, we used an inductive analysis, i.e., started from a few vague ideas of interests and developed these findings inductively (Lincoln & Guba, 1985; Ekbom, 2002). The derived conclusions are mainly qualitative, since the main parts of the results are based on the interviews. The results of the usability test showed the necessity of increasing the illustrative level of the contents of the knowledge base by visualisation. The results further demonstrated the necessity of providing a transparent system and a graphic user interface describing the knowledge so that the user can readily grasp the contents of the system (Håkansson, 2002). All these topics, as mentioned above, are main points of the departure for this thesis.

---

\(^3\) Observation is an extremely important usability method [Nielsen, 1993] and is a simple method to use for non-skilled experimenter. By using this method, it is easy to check whether the test user utilises the software, unexpectedly, i.e. different from the software developers intentions. Furthermore, this method supports the checking of learnability and errors.

\(^4\) Interviews can be used to extract the test users’ subjective satisfaction by studying their opinions. An advantage with interviews is flexibility. The interviewer may follow up ideas, explore answers and enter upon motives and feelings (Bell, 1987). Within an interview it is possible to ask questions in advance and further develop the answers from earlier asked questions.

\(^5\) Drawing images of the contents of a system is for ascertaining the subjects’ mental model of the system. These models are the ways in which people model processes (Allen, 1997) by combining or transforming inputs to producing outputs. Mental models can be developed through experience of interacting with the system (Rook and Donnell, 1993) where the type of interaction can support or modify an existing mental model or create a new mental model.

\(^6\) Reliability refers to the extent of getting the same result from repeated measurements (Nielsen, 1993).

\(^7\) Validity measures if a question really tests what it was suppose to test (Bell, 1987).
The application of visualisation in software engineering has become more and more popular. One of the latest and most promising ways of achieving visualisation is through the Unified Modeling Language (UML), which more or less has become a standard notation for developing object-oriented systems (Jacobson et al., 1998; Booch et al., 1999; Rumbaugh et al., 1999; Larman, 1998; Larman, 2002). UML has also been used in knowledge engineering, or more specific in CommonKADS methodology, for developing knowledge-intensive information systems (Schreiber et al., 2001). Even though this standard is normally largely inappropriate for rule-based knowledge-based systems (Rumbaugh et al., 1999), we argue in this thesis that it is appropriate to use some of the diagram notations from UML, provided that some modifications are made to these notations (Håkansson, 2001).

The tendency to use visualisation as an integral component of programming languages has grown rapidly over the past ten years and a large number of tools, methods, and strategies have been developed for this purpose (Burnett et al., 1998; Burnett, 1999; Agusti et al., 1998; Puigsegur et al., 1996; Puigsegur et al., 1997; Gurr, 1999; Erwig, 1989; Robbins et al., 1996; Chang et al., 1986; Weber & Kosslyn, 1986). The idea behind visual programming languages is to make the programming more accessible, to improve the correctness when tasks are performed and to improve the speed of the programming (Burnett, 1999). Some common strategies in this context are, e.g., concreteness, directness, explicitness, immediate visual feedback with four levels of liveness (Tanimoto in Burnett et al., 1998) and conceptual simplicity (Robbins et al., 1996). Studies have shown that visual programming techniques can be effectively utilised also for declarative programming languages since the focus of the language is on structural relationships between the data and these tend to be inherently multi-dimensional (Burnett & Ambler, 1992). Since the visualisation of compositional declarative programming language normally uses set-based semantics, we investigated both Venn diagrams (Venn, 1880 in Baker, 1990), and Euler diagrams (Euler, 1768 in Baker, 1990) as well as a combination of these, known as Venn-Euler diagrams (Gil et al., 2000; Puigsegur et al., 1996; Puigsegur et al., 1997). Unlike the previous approaches to visual programming, which supply, e.g., a visual support to pure first-order logic programming, we have visualised a compositional operator based relational programming language (Håkansson et al., 2001; Håkansson & Hamfelt, 2003).

There is a central core of modelling techniques for this research. Knowledge transfer (Sandahl, 1992) deals with how to transfer knowledge from domain experts to end users, conceptual modelling (Luger et al., 1993) describes how to create a semantic model of the domain, and conceptual
design (Norman, 1986) aims at modelling the system from the users’ intentions. From these models, a new methodology for user-centred knowledge modelling was developed (Håkansson, 2003:a). This methodology operates largely on a graphic representation of domain knowledge, reasoning strategies and functionality. Since each of the models mentioned above is accepted as common methods for their intended purposes and, used in accordance with the intention in our research, the resulting user-centred knowledge model ought to be valid for its intended purpose.

However, a limitation in this research is that the tool that will be utilised in the development has not been fully implemented and, hence, the suggestion of using UML in this research has not yet been thoroughly tested. All parts of UML may not be equally useful for these purposes, but this has to be further investigated. Nevertheless, in this work it is shown that this methodology can be used beneficially as a conceptual model also for rule-based systems. A feasibility test shows that UML’s diagrams are valid for rule-based systems if properly adjusted. Moreover, several reviewers at research conferences have pointed out that they welcome this research and are looking forward to seeing more of it in the future. It is important to note that a successful implementation of the user-centred knowledge modelling methodology developed in this thesis needs a considerable amount of testing before it can become a fully-fledged methodology for supporting users in developing knowledge-based systems.

1.3 Scientific contribution

The scientific contribution of the research conducted for this thesis is a graphic representation and visualisation modelling methodology to support the knowledge acquisition process, and especially modelling domain knowledge. This methodology:

- Incorporates the different users in modelling and consulting domain knowledge, reasoning strategy and functionality through a conceptual model.

- Enhances the transparency and supports the modelling of content by utilising graphical description of the content of the knowledge base.

- Makes it easier for the users to grasp the contents of the knowledge base, reduce the problem with complexity of that knowledge base, and facilitates the alteration of reasoning strategies in knowledge-based systems by means of conceptualisation.
• Enhances the transparency of the reasoning strategy, both statically and dynamically, and supports the modification of that strategy.

• Supplies a visual compositional logic programming language for adding and modifying functionality in a compositional declarative relational programming language by visualising operations and components of program composition reflecting the set theoretical semantics of the language.

• Supports an assimilation of domain knowledge by utilising conceptualisation on top of links in hypermedia systems such as the World Wide Web.

1.3.1 Creating a Methodology for User-Centred Knowledge Modelling

To centre the knowledge acquisition process round the various users, i.e., both the design users and the end users, we introduce a methodology for constructing a user-centred knowledge model. User-centred knowledge modelling sets out to model knowledge directly through the interface and, therefore, affects the original knowledge engineering process. More specifically, it changes the ordinary knowledge acquisition process, which usually uses the distinct stages, conceptualisation and formalisation, to model knowledge with the assistance of a conceptual model. Our user-centred knowledge model fuses the conceptualisation and formalisation into one homogenous stage, which we refer to as knowledge acquisition, with several iterations taking place within that stage. Thus, the activity incorporates explicit conceptual modelling and formal modelling directly into the interface, where the latter is automatically performed by the system.

The proposed user-centred knowledge modelling integrates different models taken from different disciplines. It incorporates knowledge transfer, conceptual models construction and conceptual design to support the different users. This combination facilitates the adoption of the users’ perspectives and is an attempt to mold the system in the manner best-suited to the different users of the system, that is, the design user and the end user. The design user extends and modifies the tool, the end user utilises the developed system.

Knowledge modelling in the user-centred manner proposed involves the usage of mental models and a conceptual model, and the adoption of a modelling view and a consulting view. The users utilise their mental models when working with the tool and the system as well as when learning to work with them. A conceptual model is the actual modelling tool utilised for modelling the domain knowledge, reasoning strategy and functionality. This model is presented through the modelling view, which has been specially
designed for the design users, whilst the consultation view is the view intended for and designed to meet the requirements of the end users.

The visual conceptual model constitutes a link intermediate to the different users and the system, which states the contents of the system to the users. To make the model comprehensible, there are some requirements that the system ought to fulfil, in particular, in terms of consistency, visual processes, feedback and documentation. These requirements need to be supported in the user interface of the system and, therefore, a visual modelling tool or language such as UML is utilised. This language employs standard notations, which are visually presented in a number of different diagrams. It also supports development of software systems. Standard notations assist consistency and the diagrams display the processes visually. Feedback is constantly provided during the development in the form of positive and negative reactions, for example, the users will be informed of completed actions and errors found. Documentation of the system is explicitly presented as a visual conceptual model in the user interface through the use of a variety of different diagrams for presenting a different aspect of the content of the system and providing a picture of the way in which the system operates.

In theory it is possible to utilise a graphical interface to model knowledge in knowledge-based systems by allowing several different diagrams to be used for the modelling. This implies utilising visualisation in the knowledge acquisition process by acquiring domain knowledge through the use of a visual modelling language. Finally, visualisation can also be a facility for developing and changing applications with compositional operators and, thus, this facility can be utilised to add or modify the functionality of the system.

1.3.2 Enhancing the Transparency of the Contents of the Knowledge Base

To enhance transparency and support the modelling of content by utilising graphical description of the content of the knowledge base, we illustrated how the design user can be supported while inserting the knowledge and while keeping track of the knowledge inserted. Furthermore, it is important to know how the system processes this content. The method we use to support the developer in software design is UML, developed for object-oriented system, but here modified to suit rule-based systems.

We use several diagrams to illustrate how the content of a system can be displayed by showing how various diagrams can support the design user. By providing these presentations, we believe the design of knowledge bases will be made easier, and logical and physical errors will be avoided. But a particular advantage is that the design is conducted from the perspective of
the domain, and is therefore ideal for the users, instead of being governed by the system’s characteristics. The end user can also be supported by some of these presentations. However, one important issue is to let both the design user and the end user share at least one diagram to simplify the communication between these users and support the end users’ comprehension of the domain. This sharing of diagrams and the explicitness of the knowledge in the diagrams makes it possible for the design user to have expectations about the knowledge inserted by the end user.

1.3.3 Conceptualising to Simplify Grasping the Contents of the Knowledge Base

To simplify for the users grasp the contents of the knowledge base, reduce the problem of complexity of that knowledge base, and facilitate the alteration of reasoning strategies in knowledge-based systems, we introduce the use of conceptualisation in diagrams. The diagrams in the graphical interface need extensions to simplify the view of the knowledge base and to search for knowledge already implemented in the system. To achieve such simplification and searching, a kind of conceptualisation is used as a means of enhancing transparency and reducing the problem of the complexity of the knowledge bases. Conceptualisation is also used for illustrating and changing the reasoning process at different levels of abstraction and to obtain a semantic understanding of the formalisation of the domain knowledge.

The visual diagrams use concepts that correspond to the rules. Thus, instead of presenting a rule’s physical structure, a concept with semantic meaning is applied to a rule. In knowledge-based systems, the notion of a concept is expected to grasp the semantics of a rule and apply a meaning to it. Such semantics are utilised to change the way in which the users view the rules, i.e., to enable them to understand the result of a rule from a semantic point of view. These rules are perceived as semantic concepts, which, together with relationships, create a conceptual knowledge management structure where these concepts may reduce the problem of the complexity.

Utilising concepts in a time sequence diagram may not support the users in recalling major parts of the contents of the knowledge base. An additional and more expressive diagram is introduced where the concept can be illustrated hierarchically. This allows complex structures to be represented in a simplified form and permits the modelling and visualisation of rules and their structures at different levels of abstraction. Moreover, in a cluster-based visualisation, concepts supply an overview of the contents of the knowledge base as well as underlying meaning of the clustered rules. The assignment of concepts provides complimentary information to the user by presenting and explaining the concepts already utilised. This supports the
user’s comprehension and, moreover, the developer’s recall, and thereby supports the recycling acquisition processes.

1.3.4 Enhancing Transparency and Supporting Modification of the Reasoning Strategy
To enhance transparency of the reasoning strategy, statically and dynamically, and support the modification of the strategy, we introduce visualisation in the form of graphic diagrams. As mentioned above, conceptualisation can improve the semantic understanding of the formalisation of domain knowledge by applying concepts to rules. If conceptualisation is applied to rule-bases, this can make the reasoning strategy easier to control and to change, and can also facilitate modelling of the reasoning strategy. In addition to dealing with the contents of the knowledge base, the modelling must be improved to support a change in the reasoning. This makes it necessary to include visualisation of the structure and presentation of the contents to promote comprehension of the reasoning strategy.

Diagrams are also used to explain the order the interpreter traverses the rules during their execution. Utilising concepts and visualising them in a time sequence diagram and collaboration diagram illustrates the rules and their relationships in a static and dynamic manner, respectively. Static presentation involves visualising the actual contents of the rule in the knowledge base to support changing the reasoning strategy. Dynamic presentation depends on the inputs the user inserts into the diagram, i.e., it is dynamic in the sense that it changes with the inputs, and visualises the rules and relationships corresponding to a specific conclusion. Actively using concepts supports the development of systems. If the contents of the diagrams are constantly presented to the engineer developing the system, then he can interact with the concepts to avoid quality assurance problems, e.g., by trying to perform verification with redundant, conflicting, subsumed or circular rules.

1.3.5 Adding and Modifying Functionality through a Visual Compositional Logic Programming Language
To supply adding and modifying functionality in a compositional declarative relational programming language, we introduce a visual compositional logic programming language, ViCoLL. Visualisation of program composition operations and components reflects the set theoretical semantics of the language and supports novices in changing the functionality in an already existing system.

ViCoLL’s graphical interface uses a combination of two different visual logic diagrams, Venn and Euler diagrams, to form a Venn/Euler diagram.
These Venn and Euler need modification and therefore differ from the original diagrams. In addition, the visual interface of the system utilises graphical objects, and it is well known that graphical objects use a considerable amount of screen space.

To support the use of more sets and minimise clutter, three-dimensions are utilised in the representation of the sets. The extra dimension simplifies the presentation of additional sets. The use of 3D can make the recognition of what different structures have been designed to accomplish easier. This can be useful when a user wishes to understand the purpose of a particular part of a system. The latter can, of course, be invaluable for supporting the reuse of structures.

The declarative notion is preserved in the user interface. No meaning is assigned or notions are applied because an object is placed in a certain position with respect to another, whether above or below or in front or behind another object. Furthermore, the users are allowed to place the objects wherever they like. When the order of interpretation becomes significant, a kick-off question, which forms the starting-point for the execution, is utilised to follow the interpretation of the program.

1.3.6 Supporting Assimilation of Domain Knowledge

There is general agreement upon that a knowledge-based system can never be exclusively complete and, moreover, a knowledge base never can contain enough knowledge to handle all kinds of, e.g., consultations, conclusions, on-line help, and explanations. Therefore, the knowledge base needs to incorporate (or assimilate) external knowledge. To support assimilation of domain knowledge, we will incorporate external knowledge from the World Wide Web (WWW) – an almost inexhaustible storage of knowledge. However, large part of the knowledge is uninteresting and to extract the knowledge useful for the knowledge-based system, we have to discern this knowledge before providing it to the system. Thereby, we have to browse, explore the most interesting links and provide these to the system, which we do by applying conceptualisation to links at the WWW. To assimilate knowledge, the concepts applied to rules can be applied to links as well. To test an application using concepts to links and different ways of browsing networks, we introduced a user-centred perspective using personalised dynamic libraries, which are discussed below.

Dynamic libraries are used to simplify the navigation within the Internet by illustrating a kind of “historical network”. This network shows the user all the nodes that have been visited by using “the library concept”. Adopting a graphical view enables to show in what area these links were found. Since the dynamic adding of links may generate large networks, the filtering out of irrelevant links has been improved to simplify the users’
conceptualisation. The content in the dynamic library is not compared to a page until after the user has fetched the page. Checking the page contents for significant words before using a link reduces the number of visits to irrelevant links. It also makes it easier to maintain the conceptual network since the user does not have to go back and forth to different pages, which may disturb his conceptualisation.

The way these different issues, developing a methodology, enhancing transparency of the content of the knowledge base, simplification of grasping the contents, enhancing the transparency of the reasoning strategy, supplying visual programming language, and enabling assimilation have been realised is described in the following structure of the thesis.

1.4 Structure of the thesis

In the literature, a number of fundamental topics in knowledge-based systems, have been identified as accounted for in this thesis, with the focus being on the knowledge acquisition process. These form the theoretical platform for the work conducted for this doctoral thesis. The core of this thesis is using graphic representation and visualisation to provide modelling support for the knowledge acquisition process. It is an approach that enables a user-centred design methodology to be adopted. In so doing it facilitates graphical modelling, conceptualisation, visualisation of reasoning strategies in rule-based systems, visual programming and assimilation for supporting different users to model domain knowledge, reasoning strategies and functionality as well as enabling the users to consult a knowledge-based system. This thesis comprises seven papers, which are summarised in subsequent chapters, after which, issues for further research are presented.

The thesis is organised as follows:

Chapter 2 portrays the fundamental topics essential to the successful development of systems operating on knowledge. Various types of knowledge and a number of reasoning strategies have been proposed in the literature and accounted for. This chapter also considers the notions of different systems containing knowledge where the focus is on the knowledge-based systems.

The knowledge engineering techniques used to build these kinds of systems are briefly summarised in this chapter. This knowledge engineering is divided into distinct and characteristic stages for the development, with a vital part being the knowledge acquisition. This knowledge acquisition is described with the various parts entailed, i.e., elicitation, modelling and
formalising knowledge. Knowledge elicitation is the interaction between the domain expert and the knowledge engineer and includes the collection, interpretation and analysis of domain knowledge. Knowledge modelling implies that domain knowledge is represented graphically by utilising modelling techniques. This results in a conceptual model, which lies somewhere between the knowledge engineer and the system. This model is used to formalise from which the knowledge is implemented in the system. There are, however, some problems associated with acquiring and eliciting knowledge that are also described in this chapter.

Chapter 3 starts by describing the work of supporting different users, i.e., the domain experts and/or knowledge engineer in the knowledge acquisition process and the end users in the consulting process. An expert system, the Environmental Impact Assessment (EIA) system (Håkansson & Öijer, 1993), which assists the end users, is briefly presented in this chapter. It uses texts, pictures and on-line help to advise and be educational within a specific domain, i.e., for hydropower development and river regulation projects. In addition, an intelligent rule editor, the Knowledge Acquisition with Natural Language (KANAL) system (Håkansson & Widmark, 1996; Håkansson et al., 2000) is presented, which assists the domain expert, in the role of a knowledge engineer. This system used a dialogue-controlled interaction for inserting, modifying and verifying the knowledge in the knowledge base. Moreover, a usability test of the KANAL system, which displayed the necessity of providing a transparent system and improving the user interface through graphics and visualisation is described briefly (Håkansson, 2002).

Chapter 3 also presents an approach for using graphic representation and visualisation as modelling support for the knowledge acquisition process in the form of user-centred knowledge modelling. Moreover, this chapter presents some graphic modelling of rule-based systems by utilising the visual Unified Modeling Language (UML). These diagrams are used for inserting and modifying knowledge and the reasoning strategy. Furthermore, the chapter presents a means by which to handle these diagrams through the use of conceptualisation, which is used to enhance modelling the domain knowledge and coping with the contents of the knowledge bases by applying concepts to rules.

Visualisation of reasoning strategies, including both static and dynamic presentation of rules in a knowledge base, is described in Chapter 3. Again concepts are applied to rules to support the users. Moreover, a visual compositional logic programming language is described. This is used for developing declarative programs and for adding or changing functionality in knowledge-based systems. Finally, assimilation is discussed as an extension of the knowledge bases with knowledge from hypermedia systems. To find interesting information in complex hypermedia systems, conceptualisation is
applied to the links. Concepts applied to links may have a correspondence to the same concepts applied to rules.

Chapter 4 provides a summary of the articles included in this thesis. The first paper is presented as a summary of the use of graphic representation and visualisation as modelling support for the knowledge acquisition process. This is an approach to user-centred knowledge modelling in knowledge-based systems. In this approach, the combination of different manners of acquiring knowledge, i.e., knowledge transfer, conceptual modelling and conceptual design are used to obtain a means of user-centred knowledge modelling. The most significant parts are the different users – the design user and the end user and their mental models. These mental models are to be used to learn and to handle modelling and consulting with the system through a conceptual model. Furthermore, the different views of the system are essential since they provide the design user’s modelling view and the end user’s consulting view in the form of graphic diagrams. The different types of user, the mental models, the conceptual model and the different views are presented briefly.

The second paper is presented as a summary of the proposed method for graphic modelling of rule-based systems utilising the UML. It concerns an approach to modelling knowledge in knowledge bases. To provide visual modelling of the knowledge in rule-based systems and enhance transparency, several diagrams from the visual modelling language UML are utilised. Some of these are directly applied onto the content of the system, but some are adjusted to the knowledge representation of the KANAL-system, i.e., to the production rules. The purpose of using UML is to provide many different views of the same content and to support the users when modelling and altering domain knowledge.

The third paper is described as how to manage these diagrams through the use of conceptualisation. It is a visual conceptualisation for knowledge acquisition in knowledge-based systems. This conceptualisation, which concerns semantic concepts and relationships, provides an approach to viewing and reusing the contents in a knowledge base and changing the reasoning strategy in the system. The conceptualisation is visualised with the help of the graphical sequence diagram from UML, which describes the contents of the knowledge base. However, it is difficult to track down specific pieces of knowledge in large diagrams and, therefore, we introduce a hierarchy with classifications. A kind of conceptualisation with semantic concepts is also applied in hypermedia systems to manage information browsing in large systems and to cope with complex systems.

The fourth paper presents visualisation of reasoning strategies. It supports illustration and modification of the reasoning strategy by using visualisation and suggests that visualisation is used to follow the reasoning strategy more
easily and to change it in the knowledge-based system or, more particularly, in rule-based systems. Since it concerns strategies, the presentation needs to be illustrated with a stepwise execution. The visualisation includes both the static and dynamic presentation of rules and facts in the knowledge base. The static presentation visualises the actual contents of the knowledge base without being affected by any external values. The dynamic presentation, in contrast, visualises the rules used and the facts relevant to a specific consultation, i.e., the presentation depends on the inputs inserted by the users.

The fifth and sixth papers introduce a visual compositional logic programming language, ViCoLL, which is both a visual compositional logic language and a visualisation tool for declarative relational programming. The functionality of a knowledge-based system needs to be changed to satisfy specific domain tasks. Changing written programming code can be difficult, especially if the user of a tool is a novice to the programming language. To facilitate the development of functionality in a system, this visual compositional logic declarative programming language, ViCoLL is provided. This language uses visual program composition operations, which reflect the set theoretical semantics of the language. Set theoretical semantics is visualised in Venn diagrams and Euler diagrams and, therefore, a combination of these constitutes the interface.

The seventh paper discusses applying conceptualisation to links in hypermedia systems. A hierarchy of concepts is exemplified through the use of a dynamic library as a means of managing information browsing in networks.

In the final chapter, Chapter 5, a concluding discussion and remarks are discussed along with suggestions for further work.

1.5 Articles

The thesis is based on the work conducted for the following articles:


The paper “Visual Conceptualisation for Knowledge Acquisition in Knowledge Based Systems” is presented as a poster at the conference of The Twenty-second SGAI International Conference on Knowledge Based Systems and Applied Artificial Intelligence (ES2002), Cambridge, UK.

The ideas and solutions in the paper “ViCoLL — a Visual Compositional Logic Language” are mine inventions. This paper has been presented as a poster at the conference of Symposia on Human-Centric Computing (HCC01), Stresa, Italy. Lars Oestreicher and Andreas Hamfelt were the supervisors and Torsten Jonsson was working on the same project of visualising the text-based compositional programming language, CombiLog (Hamfelt et al. 1998). The paper “Visualisation of a Declarative Relational Programming” is an extension of the work with ViCoLL, developed by me. What is more is that I am the main author.

In the paper on the dynamic library, the solution of making the links more intuitive, by using concepts to links and gather the interesting concepts in dynamic libraries, was invented by me. Lars Oestreicher supervised this work.
2 Fundamental topics in Knowledge-Based Systems

One way of viewing the development of software systems in the field of artificial intelligence (AI) is to consider that the development represents the encoding of human intelligence into computer-based systems (Negnevitsky, 2002), or that it is an attempt to model aspects of human thought on computers (Howe, 2003). Enabling the transfer of a person’s intelligence, implies that the person concerned must communicate his or her model of the world first to the software engineer and then into the system itself. However, usually, there is no direct relationship between a natural language and the real world, which means that the sentences of a language must be interpreted in terms of a conceptual model held by human beings (i.e., must interpret what the linguistic expressions mean) and this model must, in turn, be related to a specific situation (Sowa, 1984, see also Hagert 1986). Moreover, it is most certainly true that errors may arise from misinterpretation or differing background knowledge when linguistic expression is mapped to a model or when a model is mapped to a domain.

In this context, a conceptualisation is an abstract, simplified view of a world\(^8\) (of knowledge) to be formally represented (Howe, 2003). Formal representation requires that conceptualisation is specified and represented as a concrete data structure. It has to be presented as a collection of objects, concepts and other entities that are assumed to exist in the domain, and must then express relationships that hold between them.

Encoding knowledge in any formal system is a challenge (Hobbs & Moore, 1985). To make it possible to process knowledge in a computer system, the forms of knowledge must be described in a very precise manner. Modelling in software development means the creation of abstract models of, e.g., objects and processes (Beekman, 1994). These models are built to communicate the desired structure and behaviour of the system to the users (Booch et al. 1999). They let the users visualise and control the contents of the system and, in addition, they enhance the comprehensibility of the system under development by offering the opportunity of simplifying and reusing software. The models can also guide the construction of the system since they design sets of concepts and ideas that mimic some kind of system (Beekman, 1994).

---

\(^8\) The term “world” will be used not to mean the world we live in specifically, but a (mostly) contained model of some specified knowledge. For example, the knowledge needed to describe a number of diseases is one such “world of knowledge” that would be required in a knowledge based diagnosis system.
Although the modelling processes required for software development are intended for all kinds of software systems, the huge amount of variation that appears with the multitude of systems, application types and organisational cultures makes it impossible to be prescriptive about the process (Wasserman, 1996 in Pfleeger, 2001). Thus, different types of software need different types of processes and the models used are selected on the basis of, e.g., risks, platforms, the number of users and the scope of the project.

The modelling process includes a series of steps, which are repeated until the system is complete and acceptable for use (Pfleeger, 2001). Some characteristics of the process are, e.g., the prescription of all the major process activities and of the resource needs, the adjustments to constraints and the organisation of sequential activities.

Modelling a process involves understanding how to bridge the gap between what should be and what really is (Pfleeger, 2001). It also defines a common understanding of activities, resources and constraints. By introducing a model of the system, inconsistency, redundancy and omissions in the process can be identified. Hence, the process should be tailored for the specific situation in which it is to be used and the model should support the finding of faults in development, and meet the budget and constraints.

2.1 Systems containing Knowledge

Software systems have been developed to assist humans in, e.g., decision-making and the intelligent search for information (Durkin, 1994). A successful implementation of intelligent behaviour in a system is dependent on the knowledge the systems use to reason with and the methods of reasoning. Hence, the domain expert’s knowledge and reasoning is to be modelled in a system, resulting in a system containing knowledge\(^9\). The knowledge in the systems must capture the domain knowledge the domain expert uses effectively, i.e., it must be able to reproduce the knowledge he or she uses to solve a particular problem within the selected domain. The system must also incorporate the domain expert’s problem solving strategy.

2.1.1 Knowledge and Reasoning Strategies

When a user is operating a knowledge-based system, the knowledge that he extracts is the expertise of the domain expert or experts. This expertise consists of ground knowledge (i.e., facts), deep knowledge (experience gained) and shallow knowledge (i.e., rules-of-thumb) (Durkin, 1994). During the extraction of domain knowledge, this knowledge is often expressed in

---

\(^9\) A single system does not necessarily contain only one expert’s knowledge. Instead several experts’ knowledge can be implemented into a system, as long as their expertise is not contradictory.
terms of facts, rules, concepts, relationships, assumptions and tasks (Tansley & Hayball, 1993), which are used for rapid problem solving (Darlington, 2000).

The knowledge acquired from the domain expert is categorised in terms of, e.g., procedural, declarative, semantic and episodic knowledge (McGraw & Harbisson-Briggs, 1989; Durkin, 1994). Within these categories, there are also distinctions between types of knowledge, such as meta-knowledge, heuristics and structural knowledge (Durkin, 1994) that can be used by the system.

Besides the knowledge handled by the system, the system also needs to handle implementing reasoning strategies. Reasoning is the process of drawing conclusions by utilising human beings’ problem solving strategies. Commonly used reasoning strategies in these systems are deductive, abductive, inductive, analogical, common sense and non-monotonic reasoning (Durkin, 1994).

2.1.2 Systems

During the recent decades there have been some changes in and ramifications from the genre of systems containing knowledge. Along with these, definitions have been altered or invented to fit different organisations. These systems are generally so called knowledge-based systems and expert systems (Hayes-Roth et al., 1983; Waterman, 1986; Anderson, 1992; Durkin, 1994; Awad’s 1996; Jackson, 1999). Other examples are knowledge systems, knowledge-intensive information systems (Walker et al., 1987; Schreiber et al., 2001), decision support systems (Darlington, 2000; Skim et al., 2002), and the most recently coined term, knowledge management systems (Awad, 1996; Awad & Ghaziri, 2004; Liebowitz, 2001).

Systems are developed for different purposes. Some of these systems are characterised by having been designed to solve generic tasks and because they can be categorised according to different problem types. Problem types can be: control, design, diagnosis, planning, prediction and simulation (Hayes-Roth et al., 1983; Jackson, 1999). However, the terminology and the kind of system have very little influence on the knowledge acquisition process since many systems containing knowledge operate on the same principles, i.e. the knowledge and the techniques for acquiring the knowledge are similar.

2.2 Knowledge-Based Systems

Every computer system is bound to utilise some conceptualisation during the design, whether explicit or implicit. And this is indisputably the case with knowledge bases or knowledge-based systems. The term knowledge-based
system includes all types of systems built on some kind of domain knowledge, independent of the implementation (Waterman, 1986). These systems simulate human reasoning through making representations of human knowledge (Anderson, 1992; Jackson, 1999). To solve problems, the systems use the knowledge in a reasoning process and utilise heuristics or approximate methods to process the knowledge (Anderson, 1992; Jackson, 1999). Whence, a knowledge-based system needs to deal with a degree of complexity that requires a considerable amount of human expertise. The system must be reliable and must be capable of explaining and justifying solutions (Jackson, 1999).

One characteristic of knowledge-based systems lies in the architecture of the system, i.e., the architecture includes a knowledge base and an inference engine (Waterman, 1986). The knowledge will have been acquired from an external source, e.g., a domain expert, and will have been expressed in some special-purpose language and stored in the system. The code for the domain knowledge is kept in the knowledge base separately from the code implementing the reasoning. This architecture makes it easier to update and maintain the knowledge base and the inference mechanism as well as the user interface (Waterman, 1986; Håkansson & Öijer, 1993).

2.3 Knowledge Engineering

Knowledge based systems are developed through the process of knowledge engineering (Anderson, 1992). This type of development differs from traditional system development in the sense that the system specification is not known well enough for the traditional development process (Peterson et al., 1990). One reason for this is that the expert within the domain cannot instantly remember all details needed in the system, so the system development must be iterative and support the experts in their knowledge recall (Peterson et al., 1990).

In knowledge engineering, the key initial phase is acquiring knowledge from an expert possessing expertise in the domain relevant to the problem under consideration (Awad, 1996). This is a significant phase of the development process, and one on which the usefulness of the product is highly dependent.

In this context, knowledge acquisition can be analysed in terms of a model specifying the process for the construction of a knowledge-based system (Buchanan et al., 1983; Jackson, 1999). During the development of the system, the knowledge acquisition can be divided into several stages, including the identification stage, the conceptualisation stage and the formalisation stage (Hayes-Roth et al., 1983; Buchanan et al., 1983; McGraw & Harbrisson-Briggs, 1989; Peterson et al., 1990; Jackson, 1999). The
Knowledge acquisition phase involves the identification and structuring of knowledge relevant to the problem in hand. Other significant stages are implementation and testing, see Figure 1.

![Figure 1. Stages involved in knowledge acquisition (Source: Hayes-Roth et al., 1983).](image)

The development is usually iterative, with refinement (i.e., fine-tuning of rules), redesign (i.e., changing the structure of the knowledge) and reformulation (i.e., changing the understanding of the system, concepts and reasoning) (Buchanan et al., 1983).

Quality assurance is incorporated as a continuous process covering an entire development cycle of the system, from the initial stage involving the design to the eventual ongoing maintenance activities (Hoppe & Meseguer, 1993). The central concepts are verification, validation and evaluation. Verification and validation are used to check whether the system gives the correct solutions and satisfies the requirements it has been designed to fulfil. Evaluation checks the structure of the knowledge base and the features of the inference engine against the domain under development (Beauvieux, 1990; Peterson et al., 1990; Polat & Guvenir, 1993). Some of these activities can, to some extent, be performed automatically using software (Håkansson & Widmark, 1996).

### 2.4 Knowledge Acquisition

Knowledge acquisition is a central activity since the performance of systems is heavily dependent on having acquired the appropriate knowledge. Knowledge acquisition starts with gathering domain knowledge from different sources within a specific domain, e.g. from the literature, from experimental data and from human experts. (Morrison et al., 1991). The knowledge engineer will gain a basic familiarity with the domain from reference material, enabling him to understand basic terminology and concepts. From domain experts, the knowledge engineer deepens his knowledge of the domain.

Knowledge acquisition is the entire process of converting the extracted knowledge into a suitable form for use in the knowledge-based system. It
includes the *eliciting, modelling* and *formalising* of information to produce well-structured domain knowledge that is usable for solving a task, and inserting this knowledge into the system (Sandahl, 1992). Knowledge elicitation consists of collecting, interpreting and analysing the knowledge; knowledge modelling involves organising the knowledge; and formalising transforms the domain knowledge into the knowledge representation of the system where inserting is the task of putting the structured knowledge into the system.

However, knowledge acquisition is not only the process of adding knowledge and modifying the existing knowledge base. It also incorporates various means of validating and verifying the knowledge. Knowledge acquisition tasks that only require adding knowledge are very different in nature from tasks that involve altering existing knowledge bases (Tallis *et al.*, 1999). Simply adding means that one can start with a fresh knowledge base without considering the knowledge inserted earlier, whereas alteration requires that the system is able to present its contents to the user in a suitable manner. The user has to know about the existing knowledge to avoid problems with redundancy or conflicting knowledge but it is also vital to understand when some parts are affected by the changes.

Each of the knowledge acquisition activities incorporates difficulties. Several techniques have been developed to avoid these. These techniques can be used for eliciting and modelling knowledge, with or without being computerised. It is even possible to distinguish a common theme among the various techniques for eliciting knowledge since some of them including modelling techniques are better adapted to the knowledge representation methods than others.

2.4.1 Elicitation

Knowledge elicitation is an important part of knowledge acquisition. It denotes the communication between the domain expert and the knowledge engineer, where the role of the engineer is to support the expert in formulating the part of knowledge that is interesting for input to the system. Elicitation has to be systematic, and knowledge has to be stored in some intermediate representation (Jackson, 1999). The separate tasks identified in the process of knowledge elicitation are *collecting, interpreting* and *analysing* (Durkin, 1994).

During collecting of knowledge, the most significant domain knowledge is extracted from the expert’s mind. During this gathering a basic understanding of the domain knowledge is achieved, whereupon the expert proceeds to try to collect more specific knowledge. The required knowledge contains the domain objects, the basic relations, the common solutions and solution strategies (i.e., the structural knowledge) and, if available, the
expertise (known as heuristics). If possible, it should also contain the common sense knowledge used to solve difficult problems (called epistemological knowledge) and conceptual knowledge (required for modelling domain elements, including developing the reasoning strategy). All this knowledge facilitates the modelling of objects, relations, facts and other domain elements, including strategy planning, causal and formed-based reasoning diagrams (Alonso et al., 2000).

Following the collection sessions, an interpretation of the knowledge collected takes place. “Constructing an interpretation” involves gaining an understanding of the knowledge collected, which is largely unstructured, and reviewing and organising it (Alonso et al., 2000) with respect to a selected conceptual framework (Jackson, 1999). The gathered knowledge is transcribed and reviewed to identify the key chunks of knowledge (Durkin, 1994), a process akin to the chunking of information\textsuperscript{10}. Knowledge about strategies, rules, heuristics and concepts are good examples of this central knowledge. It contains both general aspects including the specification of overall problems and more specific aspects encompassing the definition of specific goals and constraints and the scope of the problems.

The final analysis is meant to reveal hidden or vague chunks of knowledge with the end goal being to provide an insight into how to form theories relating to the organisation of the knowledge and problem-solving strategies. During analysis, concepts, objects, facts and the relationships between them are described in some detail. Analysis also uncovers rules, strategies, agendas and procedures, including meta-knowledge and heuristics.

Since knowledge acquisition is an iterative process, there will be both newer and existing knowledge to consider during the analysing process. For this purpose, the knowledge gained is recorded in a knowledge dictionary. The knowledge in such a dictionary can be used for sorting and relating similar chunks of knowledge, hopefully in a manner that is compatible with that of domain expert.

Techniques for Eliciting Knowledge

Some techniques for elicitation have been developed around theories founded by cognitive psychologists concerning the types of knowledge a human being normally uses, how humans organise this knowledge mentally and how they use it efficiently to solve a problem (Durkin, 1994; Carlsson, 1993; Gaines & Shaw, 1995). Some of the most common techniques for the collection process are interviews, case studies, forward scenario simulation, goal decomposition, pure reclassification, induction, association table

\textsuperscript{10} A chunk is a piece of information that is of undefined absolute size, but which can represent a single concept in a person’s mental model.
traversal, protocol analysis, laddering, concept sorting and repertory grids (see, for example, Durkin, 1994; Darlington, 2000; Schreiber et al., 2001).

The result of sorting and relating knowledge needs to be reviewed (Durkin, 1994) and, from this review, the conceptual structures will evolve, strategies will become apparent and the networks of rules will grow. At some points during the development, the expert needs to confirm the understanding he has gained, i.e., the model of the knowledge collected that has been arrived at. It is the opinion expressed in this thesis that this understanding of the elicited knowledge can be supported by a graphic representation of the knowledge.

2.4.2 Modelling

Naturally, the domain knowledge and strategies of the systems have to correspond to the current domain. Therefore, the knowledge gathered must also be adequately modelled in the system.

One approach to the modelling of this knowledge is the use of a conceptual model (Luger et al., 1993). The conceptual model provides a sketch of the system, which can be analysed in greater detail (Anderson, 1992).

A conceptual model can be beneficial to both the domain expert and the knowledge engineer. The model can be regarded as a representation of the domain expert’s view of the domain and the relevant domain knowledge (Alonso et al., 2000). It can also be regarded as representing the knowledge engineer’s view of the domain. The model is the knowledge engineer’s evolving concept of the domain knowledge. Moreover, the use of a conceptual model can simplify the task of knowledge engineering greatly (Durkin, 1994) if it can be made to work as an intermediate link between the expertise and the system.

A conceptual model can underlie the construction of the formal knowledge base. However, such a model is neither formal nor directly executable on a computer. Instead, it is situated between the experience in the world, i.e., the human expertise, and the creation of code for the computer, i.e. the final implemented system (Durkin, 1994). It is used as an intermediate model for designing and testing the system. It can be regarded as a template with which to constrain and codify human skills (ibid.).

One difficulty with the use of a conceptual model is the actual construction of the model of an expert’s reasoning strategy and the knowledge needed to support this strategy. Correct modelling of this strategy is one of the key factors determining success (Alonso et al., 2000). One problem with using conceptual models in knowledge acquisition is the choice of knowledge representation (Jackson, 1999). Usually the model must be adjusted to the representation technique and each step of the refinement
can be troublesome and can lead to either errors in the logic (i.e., in the thinking) or physical errors, (i.e., errors in programming language code).

There can be no strict dividing line between elicitation techniques and modelling techniques. Some modelling techniques are based on certain elicitation techniques and, therefore, resemble each other in their design and use. Moreover, the modelling techniques can be used for eliciting knowledge, e.g., during the analysis phase, where experts can be asked about the correctness of a certain model.

Visual Representation for Modelling

Modelling knowledge in computers requires a large amount of interaction with the computer to get the knowledge properly represented since the domains usually are diffuse. In this context, there are several different graphical dialog styles that can be used in a user interface for modelling support, examples being, menus (Mayhew, 1992), fill-in forms (Burnett and Ambler, 1992; Burnett et al., 1998), icons (Sebillo et al., 1998; Preece & Keller, 1990), metaphors (Lantz, 1996 Kahn, 1996) and direct manipulation (Shneiderman, 1998; Mayhew, 1992). All of these have different properties when it comes to supporting the user’s overview and understanding the interaction with the system (Norman, 1986)

A conceptual model can also be constructed on the basis of data flow diagrams (DFD). DFD present a visual representation of the elemental structure of a system (Anderson, 1992), and the conceptual model is a high level DFD. There are many other techniques for representing knowledge visually, such as conceptual maps, inference networks, flowcharts, decision trees (Durkin, 1994) decision tables (Darlington, 2000) and repertory grids (Schreiber et al., 2000). No details will be given about any of these in this thesis, suffice it to say that they all have properties that enable a user to gain an understanding of the knowledge that they represent.

2.4.3 Formalising Information into a Knowledge Representation

The formalisation of the domain knowledge is a process intended to transfer the conceptual model into some kind of formal knowledge representation. This process is, therefore, mostly concerned with finding ways in which knowledge can be formally described for purposes of symbolic computation (Jackson, 1999). The notion of being “formally described” involves using a language or notation with a well-defined syntax and well-defined semantics to capture the knowledge. The syntax shapes and restricts the form of a statement in a language and the semantics defines the meaning of the expressions.

The most commonly used knowledge representation techniques are production rules (known as the antecedent–consequent rules), semantic
networks (Quillian, 1968) frames (Minsky, 1975), logic programming (Kowalski, 1974) and object-attribute-value triplets (Durkin, 1994; Darlington, 2000).

Logic Programming Language
All the representation techniques mentioned above can be implemented by using a logic programming language. Logic programming is a declarative programming paradigm based on predicate calculates and has a set-theoretical semantic, i.e., it operates on sets of statements of truth. The use of declarative programming languages has evolved from the early AI research and, traditionally, the requirement has been for a programming language that handles symbolic computation (Jackson, 1999). A symbol is an entity that has been chosen to represent a chunk of information with a greater or lesser degree of complexity in a given reference system. This notion of a symbol forms the crucial link between AI and formal systems of logic and mathematics.

In imperative programming languages there is a long tradition of advocating structured programming, essentially based on schemes (for sequences, conditionals, and iterations). This structured programming is to a large extent carried over to the now predominating object-oriented paradigm. Similarly, functional programming languages support similar schemes for compositional programming, either via built-in operators or via facilities for defining higher-order functions taking functions (programs) as arguments. Compositional logic programming methodologies have been developed by, e.g., Hamfelt et al., (2001).

Logic programming uses logical clauses rather than functions. In logic programming, knowledge about problems and assumptions that are sufficient to solve is stated explicitly (Sterling & Shapiro, 1986). The program is provided with a problem that is formalised into logical statements. The execution of this program is an attempt to solve the problem. In a logic program, the knowledge is stated as set of axioms, which can be read and executed as procedure of a recursive programming language, that is, if the axioms are stated in Horn clause logic. A logic program defines a set of consequences, which is its meaning and the computation of that program is a deduction of consequences (Sterling & Shapiro, 1986).

2.4.4 Recognised Problems
The knowledge acquisition process is often regarded as problematic and is often considered to be a bottleneck in the design process (Mcgraw & Harbisson-Briggs, 1989; Sandahl, 1992; Durkin, 1994; Darlington, 2000). Basic problems associated with constructing knowledge-based systems arise in every activity involved in knowledge acquisition, i.e., during eliciting,
acquiring, representing, structuring and refining the knowledge for the system (Sandahl, 1992).

According to Sandahl (1992), a general characteristic of knowledge acquisition is that it is a time-consuming part of system development. Errors, gaps and inconsistencies may be difficult to discover, which may require that there will be a great deal of interaction between experts and knowledge engineers to debug the knowledge. This makes the acquisition expensive, e.g., because the standard approach to knowledge acquisition involves at least two expensive people, namely, the domain expert and the knowledge engineer (Morrison et al., 1991). In addition, the time devoted to knowledge acquisition is often too short. This may be because the domain expert has other assignments with higher priorities or because the amount of time needed has been underestimated.

Communication problems also impede the knowledge acquisition process. This includes problems with imparting expertise (through the retrieval of tacit knowledge) and correctly mapping the domain expert’s knowledge into the knowledge base. There is often a representational mismatch between the way experts express themselves and the symbolic representation used in a computer system. The knowledge must be converted and formalised into a suitable representation in order to be stored in the system’s knowledge base. The conversion and formalisation processes are usually very complex.

The task for the knowledge engineer is to find a suitable representation and to bridge the remaining gaps. Since the expert can find it hard to articulate what he knows, it is the knowledge engineer’s problem to develop sufficient understanding to accomplish the task in hand. There may also be linguistic and cultural differences between the expert and the knowledge engineer, and the knowledge engineer’s deficiency of knowledge can lead to the wrong questions being posed.

Two additional difficulties that arise during knowledge acquisition concern the eliciting and modelling of the expert’s conceptual model used to describe a view of the domain and the relevant domain knowledge. The difficulties with eliciting the domain knowledge from the domain expert are caused by several factors (Durkin, 1994). The domain expert may:

- be unaware of the knowledge to be used in a system. The expert acquires expertise from experience with similar problems, which becomes deep knowledge. When the expert describes this expertise, he or she can make mental leaps that may be intuitive to the domain expert, but might not be to others. Moreover, a large part of the expert’s knowledge is tacit (Schreiber et al., 2001) and therefore not conscious and difficult to explain.
be unable to verbalise the domain knowledge. Many tasks, such as motor skills, are difficult to verbalise since they are learned largely by doing or are obtained indirectly from problem-solving sessions.

provide irrelevant knowledge. This occurs when an expert is interviewed and too many facts are produced. A considerable number of these facts may be irrelevant for the system design, although they may have appeared to be relevant to the expert at the time.

provide incomplete knowledge. Introspection may lead to the expert providing a description of his or her mental processes, but skipping matters that he or she considers to be obvious.

provide incorrect knowledge if he or she lacks knowledge on an issue or makes a mistake during the introspection. This implies that the knowledge must be thoroughly tested.

The expert may provide inconsistent knowledge when describing the problem-solving strategy. This may be because the expert adopts different approaches, for example, in different interview sessions. This does not necessarily mean that the knowledge is wrong, but that it is out of context; the two conflicting views have different origins and apply under different conditions.

One problem with modelling is that the domain expert may not be able to interpret the model to create a view of what knowledge is captured from the current domain. Moreover, the domain expert may not know which knowledge will probably be used in the system (Håkansson and Öijer, 1993; Håkansson et al., 1996).

Supportive acquisition techniques have been developed, which range from partly supporting the process to automating the entire acquisition process where the tools are: development software tools, used for system development or development supporting tools, used as a complement to the readily available development software tools. Roughly, development software tools can be categorised into, e.g., shells (Jackson, 1999; Darlington, 2000; Lucardie, 1998; Alonso et al., 2000), high-level programming languages (Jackson, 1999), multiple paradigm programming environments (Jackson, 1999; Darlington, 2000) and additional modules (Jackson, 1999). Some development software supports are: intelligent editors (Davis et al., 1985; Lenat et al., 1983); (McGraw & Harbison-Briggs, 1989), explanation facilities (Patterson, 1990; Chandrasekaran & Swartout, 1991; Edman, 2001) and quality assurance facilities (Mengshoel, 1993; Håkansson & Widmark, 1996).

Still, the domain expert is not efficiently assisted in the extraction or modelling of the domain knowledge. It is difficult to transfer the knowledge into system-specific terms in the dialogues. It is also difficult to incorporate
non-perceptible knowledge, such as tacit knowledge, in the system, which is troublesome when managing the knowledge. The transparency problem, i.e., is difficult to find out the contents of the knowledge base as well as the reasoning strategy, is troublesome when managing the knowledge, e.g., in order to assemble knowledge pieces for a knowledge base. Moreover, the problem of complexity makes the knowledge base cumbersome, which can cause problems with, e.g. finding and remembering the knowledge that users have inserted into the system (Håkansson, 2002).

To support knowledge modelling with a user interface, there are some different techniques that can be utilised e.g., graphical interfaces, natural language processing, hypermedia and multimedia systems (Woodhead, 1991; Kemp, 1996; Bielawski & Lewand, 1991 in Darlington, 2000; IncWell, 2002; Håkansson & Öijer, 1993) and visual programming languages (Burnett and Ambler, 1992; Burnett et al., 1998; Burnet, 1999; Gil et al., 2000, Gurr, 1999; Harel, 1988 Nardi, & Zarmer, 1991.). However, despite using one, two or even several of these different paradigms, problems still remain. There are problems with, e.g., how to simplify the modelling process, change reasoning strategy, and how to alter of the functionality through the use of visual techniques. This is the research topic of this thesis.
3 Graphic Representation and Visualisation as Modelling Support for the Knowledge Acquisition Process

The contribution of this thesis has been an attempt to incorporate all users, i.e., domain experts and end users, in the process of knowledge transfer during the development of knowledge-based systems. The domain experts are involved in the modelling process and the end users are involved in the consultation process, when they are consulting the system for, e.g., advice.

In an attempt to support the end users during consulting the system, we developed the Environmental Impact Assessment (EIA) system (Håkansson and Öijer, 1993). This is a diagnosis system which has an advisory and an educational role and works according to a stepwise assessment method, called the EIA method, which is applicable in hydropower development and river regulation projects (Strömquist & Tatham, 1992; Haag and Haglund, 1998; see also AEIA). The EIA system has a modular architecture with a self-contained user interface, which makes it possible to combine a declarative programming language with multimedia technology for the presentation of knowledge. In the user interface, the system supports the end users through the application of text, pictures and explanations to deal with queries, to present conclusions, and for the on-line help. The declarative programming language comprises the rule-based expert system, i.e., the inference mechanism, the knowledge base, question base and the conclusion base. The EIA system uses deductive reasoning as the inference technique.

The EIA system was developed in cooperation with Prof. Strömquist, who developed the EIA method, as mentioned above (Strömquist & Tatham, 1992). The last version of the system was tested on Prof. Strömquist and Mrs. Tatham. Unfortunately, the EIA system could not be tested on the real users since the system works in tropical environment areas. However, according to Prof. Strömquist the system has been used for several years. Moreover, the system can support the end users during their work with the EIA system without support from domain expert, i.e., Strömquist, and knowledge engineers, i.e., us (Håkansson and Öijer, 1993).

By using declarative programming together with hypermedia technology, the system supports the end users in their work with the existing system. However, developing a software system is often an ongoing process and the knowledge base usually needs to be extended gradually. Changing an existing system tends to be difficult and time-consuming, besides which consequential errors often occur during modifications. Therefore, the system
must actively (or proactively) support the users, the knowledge engineer and/or the domain expert, in the modelling process if problems are to be avoided.

In an attempt to assist the domain expert — who may be a novice at computing — or the knowledge engineer — who may not be familiar with the current domain — to extend or develop a general-purpose knowledge-based system, we developed the intelligent rule editor, the Knowledge Acquisition with Natural Language (KANAL) system (Håkansson & Widmark, 1996). This editor supports the expert with the insertion, modification and verification of the domain knowledge through a dialogue-controlled interaction. The knowledge is inserted via fill-in dialogues using a very restricted natural language, i.e., using keywords in an otherwise well-defined language matching facilities (Weizenbaum, 1966; Gazdar and Mellish, 1989; Wetter & Nüse, 1992). Essentially, the KANAL-system has been designed as a general support for diagnosis and classification systems, but it should be able to deal with all kinds of systems, and should, thus, essentially be domain independent (Håkansson & Widmark, 1996; Håkansson et al., 2000).

The KANAL-system also handles some degree of quality assurance, in the form of verification and validation of the knowledge base (Håkansson and Widmark, 1996). The verification is supported by automatic inspection of the knowledge base for consistency and completeness, whilst the validation is achieved through examination of the reasoning strategies and, hence, the explanation facilities provided. These facilities are also used to support transparency and to discover any lack of domain knowledge. It is easy to support the part of the knowledge acquisition where the expert inserts additional knowledge and extends the knowledge base. However, it is more difficult to support the domain expert during the elicitation.

The KANAL-system has been tested on the knowledge base of the EIA-system. Both the EIA-system and the KANAL-system are based on a logic programming language that uses the Horn clause subset of logic (Kowalski, 1974; Sterling & Shapiro, 1986; Nilsson & Maluszyński, 1995). The knowledge in the knowledge base of the EIA-system can be extended and modified via the KANAL-system. Moreover, the KANAL-system can also insert knowledge into empty knowledge bases.

To find out if the KANAL-system incorporated problems with the interface, and to find weaknesses and conceivable solutions, a usability test was carried out (Håkansson, 2002). Since the collaboration depends heavily on systems’ interfaces, these were in focus during the investigation. An expectation was that the test would indicate if any of the representation forms suits a majority of the users and, if so, the reasons. To test the
different representation forms, the KANAL-system was compared to Wisdom (Dignum, 1999; Lucardie, 1998).

The KANAL-system has been compared to another broadly equivalent shell, Wisdom\(^{11}\), to illuminate problems with the systems, i.e., the comparison is intended to reveal differences and show if and where the same errors may be reproduced. These systems are similar in many respects (e.g, they both use rule-based, advice-giving, bottom-up reasoning) but the means of representation in the interface is very different: The KANAL-system uses a dialogue-driven rule-acquiring technique, and Wisdom uses decision-tables, frames and Wislogic (Dignum, 1999; Lucardie, 1998) to acquire knowledge.

The task for the observation study in the comparative study was to construct a diagnosis system for different childhood diseases. The subjects were studied from, e.g., frequent use of menu bar and manuals, amount of dead time, number of user errors, time spent recovering from errors, saying something negative about the system, ask the observer for the help with the system, successful insertion of information, successful finding of properties and total time for the observation. Each observation was analysed from these enumerated issues and some more and compared to the result of the other observations. The interviews were formal and examples of the questions are: what they think of the system and the interface; can anything be improved concerning the system and the interface; is the system adapted to different levels of expertise; and did something go wrong and, if so, why. Each question was analysed and compared to the other subjects’ answers. The task for drawing was to draw a picture that corresponded to the subjects’ own image of the contents of the system, i.e., after implementing the domain knowledge. These different pictures were analysed and compared to the other subjects’ pictures.

The usability test illustrated some problems with the systems: Wisdom was unfriendly and none of the subjects were able to accomplish the task within the time available. More specifically, the problems encountered, for example, the insertion procedure through the decision tables was too difficult, the menu option for draw up questions was difficult to find and to couple to the rules. During interview sessions and the sessions where the subjects were constructing drawings of what they though were the contents of the knowledge base, also mentioned in the methodology, some problem with Wisdom were mentioned (Håkansson, 2002). For example, the decision-tables felt like being up-side-down and it was difficult to get a grip of the correlations between main tables and sub-tables. Another problem was to

\(^{11}\) Wisdom is a commercially available knowledge acquisition shell developed by Arthur Anderson Co (Dignum, 1999; Lucardie, 1998).
decide which insertion method the subjects were supposed to use. In fact, some of these problems were more serious than the problems encountered with the KANAL-system.

Some of the problems in the KANAL-system involved the interaction with the system, and the transparency and complexity of the knowledge base. The problem with the interaction was finding the options in the menus, and some difficulties arose when inserting the rules. The non-transparency included problems with coupling of different chunks of knowledge, which is essential for a system that draws conclusions using these different chunks. Furthermore, as a result of the combination of a lack of transparency and the complexity, the users had problems finding and remembering the knowledge already inserted. During the interviews and the tasks of constructing drawings of the contents of the knowledge base some improvements for the KANAL-system were suggested (Håkansson, 2002). For instance, it would be helpful if it were made possible to support the users in creating sub-tasks until those sub-tasks could finally constitute a main task. Moreover, it would be advantageous to improve the representation of the domain knowledge by using a kind of map to show the contents of the knowledge base. Furthermore, using a drag-and-drop facility to insert knowledge and avoid using “syntax resembling code” in the interfaces would enhance the process. In conclusion, the KANAL-system needs to be improved by means of the following: [i] use a better visual knowledge modelling technique, [ii] apply conception onto rules, and [iii] have the ability to change functionality. The following sections will describe how it may be possible to meet these objectives.

[i] A proposal for user-centred knowledge modelling to support the users in modelling and consultation and a means of graphic modelling is suggested to improve the knowledge acquisition process (See Section 3.1, and Section 3.2).

[ii] Conceptualisation is applied to illustrate large knowledge bases and visualisation is used for viewing and changing the reasoning strategies (See Section 3.3, and Section 3.4).

[iii] A visual programming language is provided to recognise and change the functionality of the system and extending the knowledge base is supported through assimilation (See Section 3.5, and Section 3.6).

3.1 An Approach to a User-Centred Design Methodology

The term user-centred knowledge modelling is intended to suggest an approach to a knowledge-based system design methodology that is moving the modelling and consultation closer to both the domain experts and end users. To support different users, the domain knowledge, reasoning strategy
and functionality are modelled directly into the system using a modelling language. A candidate for this modelling language is Unified Modelling Language, specifically because it uses several views to reflect the contents of the system in terms of, e.g., identification of goals, description and viewing of the objects and relationships, description and modelling behaviour, and because it supports grouping and denotation. Furthermore, UML is a standard that merges several different existing modelling techniques available for system development, e.g., state-chart diagrams. A user-centred design methodology that uses UML’s different modelling techniques and views can support users when developing knowledge-based systems based on declarative programming language and when consulting existing systems, which is discussed in Håkansson (2003:a).

3.1.1 Related Work

Knowledge Acquisition Design System (KADS) is a methodology for building expert systems (Wielinga et al. 1992, Darlington, 2000). KADS has rapidly become the European de facto standard methodology for building knowledge-based systems (Tansley & Hayball, 1993), however, it can be expensive and is, therefore, mainly suitable for middle or large sized expert systems (Darlington, 2000).

Managing the complexity of the knowledge engineering process is the primary motive behind KADS. To cope with complexity, KADS introduces multiple models (Jackson, 1999), e.g., an organisation model, an application model and a task model. The organisation model represents the socio-economic environment. The application model is a representation of the problem to be solved and the functions to be fulfilled. The manner in which a function is fulfilled is modelled in the task model. Different models are used at different stages in the knowledge engineering process. For example, the organisation and application models correspond to the identification stage in Buchanan et al. (1983) and in (Jackson, 1999).

In KADS, the conceptualisation stage is split into two models: One model of the co-operation (i.e., the communication) and one model of expertise. In modelling the co-operation, the problem-solving behaviours decomposed into primitive tasks. The model of the expertise, in contrast, models the knowledge elicitation, that is it analyses the different kinds of knowledge the expert brings to the problem solving process. These models are combined to produce a conceptual model (Terpstra et al., 1993) that is real-world oriented and describes the competence in expert problem solving.

To construct an artefact that corresponds to the conceptual model, a design model is used (Terpstra et al., 1993). The design model specifies the appropriate computational techniques and representational mechanisms. These techniques and mechanisms will be used later to realise the
specification. According to Jackson (1999), the design model takes us part way to the implementation, but it still leaves us with something, which is less than a runnable program. In our approach, the purpose of constructing or extending the conceptual model is to design an executable model. During the modelling, the contents of the model will be translated into the programming language code for execution at any time. As Jackson (1999) says, the kind of pencil-and-paper analysis that KADS uses can be performed successfully, but the question is whether it is possible to be certain that a model is correct before implementing it into the system and testing its validity in the relevant domain.

CommonKADS is a successor of KADS and is a methodology for Knowledge Engineering, Knowledge Management, Knowledge Acquisition, Knowledge Analysis and Knowledge System Development. CommonKADS uses some of the notations of UML, namely: Activity Diagrams, State Diagrams, Class Diagrams and Use Cases (Schreiber et al., 2001).

In an “activity diagram”, an activity state is represented by the activity or task to be carried out in that state. This diagram is used for modelling the organisation process and the control structure. The “organisation process” is a model of the business process at different levels of abstraction and the “control structure” provides an alternative graphical notion for describing the structure of a task method, e.g., a diagnostic method. Moreover, CommonKADS uses activity diagrams to model the information input and/or output and to show the dependencies between the data from related processes. It can also be utilised to interact with the external world.

The “state diagram” is used to model the state of the system over a period of time and to model the dynamic behaviour of the system. This diagram describes the temporal order of sub-tasks and provides an image of the sequence of events and decision-making (Schreiber et al., 2001).

In CommonKADS, the graphical notation for a domain schema of the knowledge is based on the UML’s “class diagram” (Schreiber et al., 2001). The class diagram is used to describe the context of the information for the task analysis. It also describes the structure of objects handled in a task. The classes are called entities or concepts and their association in UML is called their relationships. Besides the concepts and relationships, CommonKADS has introduced an additional notation for modelling knowledge structures, that of rule-types.

Usually the purpose of use-case descriptions is to describe the system’s functionality from an external viewer’s perspective. In CommonKADS, the “use-case diagrams” are incorporated in an agent model as a summary of the agents’ interactions, where the agents are the people involved in the system. These use-case diagrams present the actors and the services (or use-cases) and, the relationships between them (Schreiber et al., 2001).
In addition to these diagrams, annotation diagrams are used within CommonKADS. These diagrams include additional chunks of information that are difficult to model, e.g., large or complex systems.

Use-case diagrams, class diagrams and state-chart diagrams of CommonKADS are all applied in a similar manner as the diagrams used in our contribution (Håkansson, 2001), however, the activity diagram is used differently. In CommonKADS this diagram displays a task or activity carried out in a state, whereas in our approach it shows the contents of the knowledge base. Moreover, it displays concepts that correspond to domain knowledge in that base.

As seen UML diagrams can be used in CommonKADS to build knowledge-based systems in object-oriented fashion. In our approach, however, we will apply UML diagrams to knowledge-based systems that have been developed in a declarative fashion. This affects the UML’s diagrams, since they cannot be used in their original form as they are used in CommonKADS.

3.1.2 User-Centred Knowledge Modelling

Visualising the knowledge acquisition process implies modelling the domain knowledge and consulting the system in a user-centred manner, which requires combining different disciplines. Visualising, modelling domain knowledge and transferring this knowledge to an end-user are not usually covered by one single model in software engineering. Our approach to user-centred knowledge modelling is novel in that it integrates models found in several different disciplines, software engineering, human-computer interaction, and cognitive psychology (see, e.g., Anderson, 1990), thereby incorporating knowledge transfer, conceptual modelling and conceptual design in a user-centred knowledge model intended to support the different users (Håkansson, 2003:a, pp. 4-5).

Within our knowledge modelling activity the most important notions are: the design user, the end users, the mental models (Hongliang & Nanyuan, 1996; Potosnak, 1989; Rook & Donnell, 1993; Sheeran, 2000), the conceptual model, and the modelling and consulting views adopted by the users. The design user is the user who extends and modifies the framework whereas the end user is the one for whom the developed system is being designed. These different users utilise their mental models whilst learning about and handling the modelling or in a consultation through the use of a conceptual model. The conceptual model is utilised for modelling the domain knowledge, the reasoning strategy and the functionality, as well as for browsing the knowledge and explaining the reasoning strategy. This model is presented through two different views: the modelling view, which is specially designed for the design users, and the consultation view, which is
designed for the end users (Håkansson, 2003:a, pp. 7-15). Our approach differs from other knowledge modelling approaches. In our approach the users models directly in the system whereas the other approaches usually models knowledge through a non-executable conceptual model that underlies the construction of the formal knowledge base (Luger et al., 1993; Durkin, 1994; Schrieber et al., 2001).

The user-centred knowledge model encompasses the needs of both the design user and the end user to allow the direct involvement of the design user to display the knowledge at different levels of abstraction and to display the entire contents of the knowledge base. The design user can be the domain expert or the knowledge engineer modelling the knowledge directly into the system. The end users are those who consult the system to obtain conclusions or find out about the reasoning strategy of the system (Håkansson, 2003:a, pp. 6-8).

The user-centred knowledge model tries to draw on the users’ mental models to facilitate the process of gaining an understanding about and modifying the system. The conceptual model can support the users in forming some kind of mental models that are guiding their experience with the system and which is to discover how the system operates, and predict future interactions with the system (Håkansson, 2003:a, pp. 8-11).

In this user-centred knowledge modelling, the conceptual model is designed into the system directly to support the extension and modification of the domain knowledge, the inference mechanism and the functionality of the system. The conceptual model is utilised by the design users to develop the system by modelling the domain knowledge. From the model, the knowledge is transformed into a knowledge representation of the system. The end users utilise the conceptual model to consult with the system and also understand the conclusions inferred by the system (Håkansson, 2003:a, pp. 11-15).

Through the conceptual model, the different types of knowledge — that is procedural and heuristic knowledge, semantics and meta-knowledge — can be presented along with the reasoning strategy and the functionality of the system. Moreover, the conceptual model can simplify checking in-built explanation facilities, which can be used to extend or modify the knowledge base and for consulting the system.

To make the conceptual model of the system comprehensible to the users, it is illustrated from two perspectives in parallel: By adopting a modelling view and a consultation view. These views utilise graphic representation and visualisation to provide modelling support as well as consultation support. For this purpose, several different diagrams from UML are used to display the contents of the system. UML employs standard notations, which are presented visually in several different diagrams and can support the
development of knowledge-based systems. UML supports visualisation of the contents of the system, as well as, visualisation of the processes (Håkansson, 2003:a, pp. 17-26).

3.2 Graphic Modelling

From the perspective of knowledge representation, graphic modelling can represent a great improvement on the existing knowledge acquisition processes under certain circumstances. It facilitates the design for the insertion and modification of knowledge and also supports tracking of the knowledge inserted. During a consultation it provides support for the data entry and it may also make it easier to comprehend the conclusions of a session. This can be accomplished by supplying a well-designed graphic user interface to a rule-based system tool, which is suggested in Håkansson (2001). Graphic modelling of the knowledge can be accomplished by utilising a graphic modelling language, such as UML (Unified Modeling Language) (Jacobson et al., 1998; Booch et al., 1999).

3.2.1 Related Work

UML is usually used for modelling object-oriented systems, but it can also be used for modelling of other types of systems, such as frame and constraint based systems (Helenius, 2001; Cranefield et al., 2001; Renker et al., 2002).

UML/OCL (Unified Modeling Language/Object Constraint Language) is a knowledge representation language for knowledge based configurators (Felfernig et al., 2002). These configurators support the configuration of complex products and services and must be enhanced with the capability of knowledge sharing and distributed configuration problem solving. Configuration is a special kind of design activity where the product is built of predefined component types and attributes (Felfernig et al., 2002; McGuinness & Wright, 1998). These are composed to conform to a set of constraints. Configuration knowledge base development is usually only feasible for those who can handle the formal representation language thereby explaining why UML/OCL is employed. This knowledge base development uses UML class diagrams and OCL expressions.

Another approach of using UML in other types of systems is conceptual modelling of constraint satisfaction problems (CSP) Renker et al. (2002). This is a form of modelling support for CSP and uses the UML object diagrams for the development of CSP models.

12 Configuration can be seen as a kind of design activity where the configured product or system is built of a predefined set of components (Felfernig et al., 2002).
UML is also used for software agents\textsuperscript{13} with UML-based ontology modelling (Cranefield \textit{et al.}, 2001). The knowledge is displayed in the form of object diagrams combining existing knowledge and information about ontology. An ontology is an explicit specification of a conceptualisation (Gruber, 1993). Each ontology can define a set of classes, relations, functions and object constants. The resulting language is a domain-specific specification of a conceptualisation.

To build ontologies, a language and a method have been developed to describe the ontologies, the principles for designing ontologies, and to address issues relating to reaching agreements for sharing and integration (Ushold & King, 1995; Puerta \textit{et al.}, 1994). The terminology adopted includes concepts, conceptualisation, ontological theory and ontology.

In our approach, the knowledge is not presented in an object-oriented fashion. Instead, UML is used to display rules in the knowledge base (Håkansson, 2001). Moreover, for displaying constraints, we suggest the use of the UML’s state chart diagram, but we have not really focused on the constraints inherent in the rules.

### 3.2.2 UML Diagrams in Rule-Based Systems

In its current form (Version 1.1, 1997), UML is not directly applicable for modelling knowledge in knowledge bases. However, UML can easily be adapted to systems containing knowledge. UML provides several types of graphic diagrams, which can be utilised for inserting and modifying the domain knowledge as well as for reasoning strategies required in the knowledge acquisition process. These diagrams can also be used to generate knowledge about static and dynamic domain knowledge and to inform the users about the system’s processing. This use of these diagrams has been demonstrated (Håkansson, 2001)

By providing such diagrams it can be easier to design knowledge bases from the perspective of the domain instead of the system’s characteristics. Moreover, both the design user and the end user can share some of the diagrammatic representations, which can simplify the communication between these users, and make the domain more comprehensible.

### 3.3 Conceptualisation

Even when UML diagrams are used to illustrate the domain knowledge in the knowledge base, it is still difficult to comprehend knowledge bases with rich sets of rules. To reduce the information, we introduce the notion of

\textsuperscript{13} An agent is a program that performs some information gathering or processing task in the background. Typically, an agent is given a very small and well-defined task.
conceptualisation in some of the diagrams. Apart from the knowledge base, a hierarchic diagram is established by visualising concepts ranging from abstract to concrete level.

### 3.3.1 Related Work

The notion of conceptual models is widespread and used in many areas of research. Because of this, the term is not understood in the same way in all disciplines. The two perspectives most closely related to the notion as used in this work are the knowledge engineering and the human-computer interaction ones.

The idea of utilising conceptual models to describe the contents of a knowledge base has its origins in object-oriented methods (Larman, 1998) and knowledge acquisition methods (Luger et al., 1993) where conceptual models were used to capture real-world concepts (objects), their attributes, and the associations between these concepts. For capturing the conceptual model, a UML class diagram notation is introduced. By incorporating conceptual models in the developmental phases, an effort is made to create a draft conceptual model, the goal being to obtain a basic understanding of the vocabulary and the concepts used in the requirement specification. In the object-oriented genre, conceptual models are provided as a complement to the Unified Modelling Language (UML) (Larman, 1998). According to Larman (1998), the model is a representation of concepts in the problem domain. It also presents generalisations and taxonomic classifications. Generalisations are supported by identifying commonality among concepts and defining general or specialised concepts. Taxonomic classifications among concepts are constructed and graphically illustrated in hierarchies.

In the knowledge acquisition process of the knowledge-based systems, the conceptual model is, usually, the knowledge engineer’s evolving conception of the domain knowledge (Durkin, 1994). By using conceptual models it is possible to model objects, relations, facts and other domain elements including, e.g., strategy and causal reasoning diagrams (Alonso et al., 2000). The conceptual model resides somewhere between the human expertise and the implemented program and underlies the construction of the formal knowledge base, but it is not normally executable.

In software development, conceptual models are regarded as the outcome of a modelling process, which addresses the manner of inferring, structuring and controlling knowledge. This modelling is one of two activities for knowledge structuring, also called conceptualisation (Alonso et al., 2000). The other activity is the creation of a formal model of the system to address the domain knowledge in the system and implement it. In human-computer interaction, a conceptual model is regarded as “a general conceptual framework through which the functionality is presented” (Mayhew, 1992). A
good conceptual model should be explicitly designed in the interface and effectively presented through the user interface (Mayhew, 1992).

These approaches differ from ours since the original design conceptual models in UML describes concepts in object-oriented systems and are applied to objects (Larman, 1998). Our conceptual models in UML are used in Knowledge-based systems and the concepts are applied onto rules (Håkansson, 2003:b). Moreover, conceptualisation usually involves structuring knowledge into two different models, conceptual model and formal model. In our approach, conceptualisation involves structuring knowledge into conceptual models, which are directly implemented in the system to support the different users in implementing the domain knowledge or utilising it. Thus, the conceptual model is used as a medium for knowledge structuring, which is explicitly designed and presented through the interface to the design users and the end users, i.e., it is a kind of visual conceptualisation.

3.3.2 Conceptualisation Applied to Rules

Conceptualisation can enhance the modelling of domain knowledge, support grasping the contents of the knowledge base and support alteration of reasoning strategies and, as we have shown (Håkansson, 2003:b, pp. 3-6). In knowledge acquisition, conceptualisation can be used to find significant concepts and relationships by which the domain can be characterised. This conceptualisation can be utilised in the development of knowledge-based systems as a means of simplifying the handling of complex knowledge bases.

In a knowledge-based system, conceptualisation allows a change to be made in the way of viewing the knowledge chunks in the rules (Håkansson, 2003:b, pp. 6-8). A concept can be attached to a specific rule or to sets of rules to include semantic meanings. It then defines an action, a conclusion, a task or an interpretation of the rules corresponding to the consequent part of the antecedent-consequent rule; this implies that concepts can be applied to rules as sub-concepts, and that putting them together will determine a main task or conclusion.

To visualise the conceptualisation, the concepts applied to rules are presented graphically through diagrams. In the time sequence diagram of UML, concepts and relationships can support the users when they need to modify the current strategy of the system. In the hierarchical diagram, the semantic interpretations can be applied at different levels of abstraction, which in turn means that users may handle many clusters of rules. It can also facilitate the management of searches for rules and thereby simplify reusing rules (Håkansson, 2003:b, pp. 8-11).
3.4 Visualisation of Reasoning Strategies in Rule-Based Systems

Comprehending and changing the reasoning strategy of a knowledge-based system is supported by visualisation as shown in Håkansson (2003:c). This visualisation includes both a static and a dynamic presentation of the rules and facts that are involved in drawing a conclusion. The presentation needs to be illustrated with a stepwise execution, which is a feature found in UML diagrams. The visualisation is presented statically in sequence diagrams and dynamically in collaboration diagrams. Moreover, concepts are utilised to capture semantic notions at different levels of abstraction and are applied to the rules and facts in the various diagrams.

To control and change the reasoning strategy, relationships between different rules and facts need to be clearly presented. By applying concepts in diagrams and visualising the contents, the knowledge modelling is improved to support a partial change in the reasoning strategy (Håkansson, 2003:c, pp. 8-9). A defined concept is expected to grasp the semantics of a rule and attach an interpretation to it. Such semantics is utilised to change how the users view the rules, e.g., to understand a rule from a semantic perspective. A diagram is also used to demonstrate the order in which the interpreter traverses the rules during their execution.

Visualising concepts in a time sequence diagram and a collaboration diagram illustrates the relationships between the rules, statically and dynamically (Håkansson, 2003:c, pp. 6-12). The static presentation visualises the actual contents of the rule in the knowledge base to support changing the reasoning strategy while the dynamic presentations visualise an interpretation of a consultation. This means that dynamic presentation depends on the inputs the user inserts into the diagram, i.e., it is dynamic in the sense that it changes in accordance with the input, and visualises the rules and relationships corresponding to each conclusion. Thus, it visualises the system’s reasoning strategy, which changes according to the different inputs.

The assignment of concepts can provide complementary information to the user by presenting and explaining the concepts already utilised. This can support the user’s comprehension and, moreover, the developer’s recall, thereby supporting the iterative modelling processes.

3.5 Visual Programming

The design of a Visual Logic Programming Language, i.e., an interface built on top of a preferably compositional logic programming language can facilitate implementing and changing applications. The language presented
in (Håkansson *et al*., 2001; Håkansson & Hamfelt, 2003) preserves the notion of being declarative and does not apply any procedural meaning to the way in which different operators are actually executed, the way in which they are combined into larger operators or their position in the interface. Furthermore, it provides an interpretation in a set-based symbolic language that corresponds to sets displaying logic in graphical diagrams. These properties make it a good candidate to become an effective tool for defining and redefining the functionality of knowledge-based systems.

### 3.5.1 Related Work

Puigsegur *et al.* (1996) have presented a visual declarative programming language based on directed graphs, so called Higraphs (Harel in Puigsegur *et al.* 1996) and graphical set inclusion, such as Euler and Venn diagrams (Erwig, 1989; Puigsegur *et al.* 1996). Higraphs’ graphical symbols are used as nodes and the set inclusion is used to represent conclusions. The Higraphs is an alternative of visual syntax for pure Prolog (Bratko, 1990), but restricted to a subset of full first order logic to avoid some of the difficulties such as the explicit naming of variables (Puigsegur *et al.* 1996).

Cocoa is a rule-based visual programming language developed by Smith, Cypher and Spohrer (Burnett *et al*., 1998). In this language the users specify the production rules by describing the pre- and post-conditions of the rules. Cocoa uses programming by-demonstration with graphic simulation, which is the way the semantics is specified. The rules are displayed in a specific window and, during the execution, the execution order is shown.

3D-Visulan programs are an ordered set of pattern-replacing rules developed by Yamamoto (Burnett *et al*., 1998). Each rule consists of a before-world rule and an after-world rule, corresponding to the rules before and after an action. The program matches the before-world rule with the after-world situation and if a rule has the desired effect, the data-world is replaced with the after-world pattern. The execution can commence when all rules have been matched. There is no order between the patterns in the before-world, however when there are multiple rules, priority must be defined between the rules. If the rules are rear rules, which means that they trigger the execution of other rules, then they have the highest priority. Otherwise, the left rules have higher priority than the right ones and the top rules have higher priority than the lower ones.

Our approach is most similar to that of Puigsegur (Puigsegur *et al*., 1996), which also uses Euler diagrams and Venn diagrams to describe declarative programming languages. One difference is the actual diagrams themselves and another more important one is in the compositionality (Håkansson *et al*., 2001; Håkansson & Hamfelt, 2003). We use or-circles in a different manner — to spread the visual parts over the interface and thereby avoid getting
cluttered diagrams. Moreover, Venn diagrams are used with intersections in our work, which is not the case in Puigsegur’s et al. (1996) interface. Instead, they use smaller squares within other squares to mark the manner in which the objects belong to each other (Puigsegur’s et al., 1996). We believe that our version with compositionality is an advantage in the design of larger programs.

3.5.2 Visual Compositional Logic Programming

To minimise the need to learn the syntax and the program structure, visual components can be used to build and maintain the system. If visual programming components are to support changing of the written code, the notion of code has to be strengthened. This was the object of the KANAL-system.

The diagrams in UML are not directly applicable to declarative logic programming. These diagrams support procedural programming, but they do not support declarative programming since they were originally intended to handle object-oriented systems. Procedural and object-oriented programming methodologies are usually illustrated by using state-chart diagrams.

Changing functionality requires reprogramming a system already in existence, which is a process involving the re-implementation of or changing existing code. This is often a difficult task because of the complexity of the code. Visual programming is one potential approach intended to support program development to such an extent that a novice could perform programming tasks.

To facilitate program development, composition operations and components can be visualised as we have illustrated in Håkansson et al. (2001) and Håkansson and Hamfelt (2003:c). For CombiLog (Hamfelt et al., 1998) a graphical user interface was built on a compositional declarative programming language to support the user in developing new functions and in editing old functions and, specifically, to avoid them having to learn a great deal about the syntax. The interface to CombiLog consists of commonly used graphic representations of sets, Venn and Euler diagrams to enable the program contents to be visualised as sets. The result of the work is a graphical language tool called ViCoLL (Visual Compositional Logic Language) (Håkansson et al., 2001; Håkansson & Hamfelt, 2003:c).

In ViCoLL, a combination of Venn and Euler diagrams can support the users when they need to add or modify the current functionality of the system. In a modified combination of Venn/Euler diagrams in 3D, the declarative interpretation can be preserved and cluttered diagrams can be avoided by different shapes and by a fold/unfold facility. The diagrams can also facilitate searching for compositions and thereby simplify reusing them (Håkansson, 2003:b, pp. 8-11).
3.6 Assimilation

One problem with knowledge-based systems is that they may not contain enough knowledge to work properly within a domain. To provide a good platform for supportive tools, the systems may be combined with hypermedia technology, even extending the information search to the World Wide Web (WWW). However, knowing whether a certain chunk of information is available on the Web has become a problem due to search and extraction problems (Harmelen & Fensel, 1999). It is difficult to find and manage information on the Web, and it will become more serious as the Web continues to grow. Since Artificial Intelligence is often used for structuring knowledge and information, hopefully it will be able to be used to solve some of these problems. Applying AI techniques can, e.g., deal with searching and extracting information, and generating documents automatically.

3.6.1 Related Work

Keyword-based search often retrieves irrelevant information. The keywords can appear in some other context or the relevant information required to find the right pages can be missing even when several words are used to specify the contents. To extract the relevant information requires excessive browsing and reading by users. Automatic agents (i.e., searching programs) usually lack the common sense knowledge needed to extract the desired information.

3.6.2 Concepts Applied to Links

A successful manipulation of the contents of large and complex systems, such as large hypermedia systems and knowledge-based systems, can be supported by viewing the contents at a high abstraction level, conveying some form of context instead of presenting links or rules at a detailed level without supplementary information. To reach a higher level of abstraction, semantically richer concepts can be applied to the contents of the medium, i.e., to the links or the rules. To form an understandable and comprehensible knowledge space, these higher-level concepts must be related. This task resembles the conceptualisation used to uncover significant concepts and relationships in the current domain knowledge and includes, e.g., different kinds of data and the underlying structure of the domain in terms of causal, spatio-temporal and part-whole relationships (Jackson, 1999). In this work, conceptualisation is used to manage complex information systems by revealing significant concepts and relationships in the system. Conceptualisation can be used for complex hypermedia systems based on simple link representation, as well as complex knowledge bases with a rather rich network of relationships between the rules. Managing a complex information system requires the search for significant knowledge chunks in a
complex network of relations with a selective facility. To select the information, the users’ key concepts have to be defined in the system and applied to related information.

Thus, to assimilate knowledge, conceptualisation can be applied to a complex hypermedia system or even to the Internet itself. In hypermedia systems, concepts are connected to constitute a knowledge network. The concepts can convey semantics and be applied to the links to navigate in knowledge spaces and search for significant and relevant knowledge (Håkansson & Oestreicher, 2000). A dynamic library was used to set up conceptualisation hierarchically. The user can search information that is conceptually adjacent to the defined concepts without having to move around in a physical network searching for useless knowledge that has just been linked and, in this way, the user does not run the risk of losing track of the significant knowledge already found.
4 Survey of Papers

The main purpose of this thesis was to supply a framework that enables graphic representation and visualisation as a support to the knowledge acquisition process for modelling domain knowledge. To accomplish this, one result of this work is to enhance the transparency and to support the modelling of content by utilising graphic descriptions of the content of the knowledge base. This is achieved with graphic modelling languages — such as UML — to simplify and improve the modelling process. The language also presents knowledge in the system to the design user, as well as to the end user by offering several views of the system. The shift in perspective between these views may enhance the understanding of the consultation of the system as well as clarify the explanations produced by the system.

Additional results of this work are to simplify for the users to comprehend the contents of the knowledge base, to reduce the problems with complexity of the knowledge base and to facilitate the alteration of reasoning strategies in knowledge-based systems by means of conceptualisation.

A feature of the graphical representation is that it simplifies the process of comprehending the contents of the knowledge base, both in detail and when trying to obtain an overall perspective. The visual presentation of the conceptual model also reduces the perceived complexity of a knowledge base, and thereby facilitates the alteration of reasoning strategies in knowledge-based systems. The conceptualisation facilitates the semantic interpretation of the formalisation of the domain knowledge. The conceptualisation also supports the comprehension of rules in a knowledge base by re-describing these rules as semantic concepts with internal relationships and, in this way, it is used to create a conceptual knowledge management structure where the concepts and their structures reduce the complexity of the detail.

A feature of visual representation is the possibility of facilitating program development in a compositional declarative relational programming language by visualising program composition operations and components, while reflecting the set theoretical semantics of the language. This perspective results in a visual compositional logic programming language, ViCoLL, for adding or modifying functionality in knowledge-based systems.

All these results and features of graphic representation and visualisation used for system development are illuminated in the articles summarised below.
4.1 Article 1: An Approach to User-centred Knowledge Modelling in Knowledge-Based Systems

This paper introduces an approach to a methodology with which the users are put at the centre of knowledge modelling (Håkansson, 2003:a). Our proposed model is a combination of other models found in knowledge engineering, human-computer interaction and cognitive psychology, which are used for knowledge transfer (Sandahl, 1992), conceptual modelling (Luger et al., 1993) and conceptual design (Norman, 1986). The methods used for knowledge transfer are important for the success of knowledge elicitation and acquisition. The conceptual model contributes a graphical model of the domain knowledge and the conceptual design constitutes a bridge over the gap between the users’ mental models and the system.

The users who are most often involved in a system development are the design users, i.e., the domain expert and knowledge engineer, and the end users. Since the design users have had different experiences and have different skills, they should co-operate during the development. These design users develop for the end users, who are involved in consultation, for example, for advisory and educational matters.

The framework, used for development, and the developed knowledge-based system have to enable the users to create mental models and also fulfil the design users’ and end users’ expectations and requirements about the system as expressed in these models. The design user’s mental model can be used during the modelling of the domain knowledge, finding explanations for events that may occur and determining the appropriate actions to execute the desired changes. The end users’ mental models are mainly used during consultations with the system, when predicting further interaction, and when discovering how the system operates and what it has concluded in a consultation.

We introduced a conceptual model to support the users in building and utilising their mental models of the domain and of the contents of the system (Håkansson, 2003:a). The conceptual model supports a range of design users by clarifying the various technical terms in the system and assisting the users to apply these correctly by illustrating dependencies. The model also allows the users to insert different types of knowledge the system uses, e.g., procedural, declarative, heuristic, structural and meta-knowledge. By modelling the domain knowledge, reasoning strategy, and functionality, the conceptual model forms the framework for the design user. The model is also beneficial for quality assurance, i.e., verification and validation. The end users take advantage of the conceptual model during the consultation with the system not only to understand the conclusion, but also the strategy behind the specific conclusion.
The conceptual model is presented in the user interface through two different views, i.e., the modelling and the consultation view. The design user’s “modelling view” encompasses the interaction for modelling domain knowledge, changing reasoning strategies and functionality, as well as, utilising the explanation facilities. The end user’s “consultation view” concerns support consulting, the explanation facilities for questions, conclusions and reasoning strategies, and on-line help.

Knowledge modelling requires the user to insert, edit and test the domain knowledge through different types of knowledge presentation methods. Procedural knowledge is presented as objects in sequences to illustrate the inference mechanism and declarative knowledge is presented as templates for facts and rules with the concepts applied to the rules. Meta-knowledge is shown at different levels of abstraction in hierarchies as classifications of the knowledge in terms of concept clusters. Heuristic knowledge is expressed as sequences of strategies, but incorporates leaps when details are missing and, finally, structural knowledge is expressed in terms of sequences using sets and relationships with the effect of folding and unfolding the sets.

To support the users the system needs to give manageable domain knowledge through an interactive end user interface. In this user-centred knowledge model, the conceptual model becomes the graphical user interface of the knowledge-modelling tool and the system. This model is relevant to both the modelling view and the consulting view. In this way, the user interface is very closely related to the underlying design of the system.

Using different views may bring the user and the system closer together and bring the systems closer to the user’s activities. The views are exemplified through graphical knowledge modelling, which is based on diagrams in Unified Modelling Language (UML).

4.1.1 Results and Discussion

There are many problems associated with knowledge acquisition, e.g., simplify the modelling process, change reasoning strategy, and alter of the functionality through the use of visual techniques, see Section 2.4.4. We suggest using a user-centred knowledge model as a solution to some of the knowledge acquisition problems since it can support the entire knowledge acquisition process, including the elicitation process (Håkansson, 2003:a). A tool, built on this model, can give visualisation support by illustrating the domain knowledge graphically at the user interface. Thereby, this tool enables the domain expert to co-operate with the knowledge engineer in designing the system, to notice and rectify any deficiency of knowledge and, consequently, to provide additional knowledge where needed.

The model also supports when transferring some parts of the domain expert’s view of the domain to the end user including all the knowledge
needed to utilise the system for its purpose (Håkansson, 2003:a). For instance, several diagrams are used in both the development process and the consultation phase. Using the same diagrams for both the domain expert and the end user facilitates transferring domain knowledge.

The result of using UML diagrams is that they constitute a conceptual model between the domain expert and the system (Håkansson, 2003:a). The process of transforming the models into a representation of the system is no longer an issue because it has been avoided altogether. This simplifies and improves the process of inserting domain knowledge because the transformations between different models are needless and the system handles the transformation from the model to the programming code. Using one model for knowledge modelling avoids unnecessary problems with incorrect transformations and bringing users into the knowledge modelling reduces errors, gaps and inconsistencies that is if the views clarify the domain knowledge, reasoning strategy and functionality. Furthermore, the conceptual model makes it easier to verify and validate the knowledge if it represents the domain expert’s mental model and the diagrams visualise the knowledge contained in the system.

To detect problems in covering the domain knowledge, i.e. a lack of information, the diagrams must be examined and checked to determine whether they support insertion of all kinds of knowledge that can be found within a domain. It is also necessary to check the usability of the system and to test if the domain expert’s mental models are supported to check the supportiveness of the system in the elicitation process.

It may turn out that more detailed specification is needed in the diagrams. Some of the diagrams may not be sufficiently expressive and some additional knowledge may have to be provided in these. Thus, the suggested use of the diagrams may have to be modified to serve specific purposes.

### 4.2 Article 2: UML as an Approach to Modelling Knowledge in Rule-Based Systems

To simplify and improve the knowledge modelling in rule-based systems, we have incorporate a graphical modelling language, based on the unified modelling language (UML), into the user interface (Håkansson, 2001). The aim of using a visual modelling technique is to support different users throughout the acquisition process and the consultation process. UML is used for visualising the content of the knowledge bases and the reasoning strategy because it uses diagrams to support the domain expert within the acquisition process.

In this paper, the graphical modelling language uses several graphic diagrams to represent the domain knowledge. The diagrams used are use-
cases, class and object diagrams, interaction diagrams with sequence and collaboration diagrams, and activity and state chart diagrams, and also packages (Håkansson, 2001).

Use-case diagrams are used for identifying the users’ goals and to describe the tasks to be performed, the modelling requirements and the functionality. In the use-case diagrams, the entire sequence of interactions that the design users and the end users have with the system is modelled.

The class notations are used as templates for the declarative knowledge to be inserted into the system, e.g., questions, rules and conclusions. The design users insert knowledge chunks through the templates and the end users use these during the consultation. These templates are presented in class diagrams with their associated relationships to clarify the relationships between them.

To display the interaction diagram, UML uses the sequence and collaboration diagrams, which we adopted in our work (Håkansson, 2001). However in our work, the sequence and collaboration diagrams are used to check the contents of the knowledge base and the reasoning strategy of the system since they display the interaction between the knowledge chunks and the relationships between them. The sequence diagram presents the contents and the interpretation of the knowledge base by using time sequences. The collaboration diagram explicitly shows the relationships between knowledge chunks and the interpretation for a specific consultation. Together with the related knowledge chunks it displays inputs from the users; the output is the result. The inputs will affect the presentation in the diagram as well as the output. The sequence and collaboration diagram are also used to visualise “Why” and “How” explanations.

The activity diagram and the state chart diagram are used to follow a consultation and the reasoning strategy since these diagrams illustrate the procedural behaviour of a declarative representation. The activity diagram displays the reasoning at a high level of abstraction and the state chart diagram gives an explicit definition of the behaviour of the knowledge chunks in the system over their lifetime with constraints and dependencies. An activity diagram can also visualise “Why” explanation and “How” explanation on different levels of abstraction. The state chart diagrams are a complement to the activity diagrams and display rules and also produce meta-rules. This diagram can also display the order in which questions are asked in advance of a consultation.

To comprehend large knowledge bases, packages encapsulate knowledge chunks and thereby decrease the complexity of illustrating the contents. Component and deployment diagrams can be used to view the architecture of the system and to incorporate notes in the diagrams, respectively.
4.2.1 Results and Discussion

The purpose of this work is to use graphic representation and visualisation when developing and utilising an existing knowledge-based system. This should include everything that concerns a knowledge-based system but our primary goal is to visualise modelling domain knowledge, modifying reasoning strategy and functionality and visualising explanations. These tasks are, in our point of view, the primary and most important for developing useable knowledge-based systems.

There are many reasons for using UML as a modelling means for rule-based knowledge-based systems. For example, UML makes it possible to describe the tasks the system performs and describe parts to be inserted into the system, as templates. Moreover, UML makes it also possible to investigate the system’s performance to decide if the system’s behaviour is acceptable, to show a procedural behaviour of a declarative representation and to illustrate the functionality, and to comprehend large systems. By providing these presentations, knowledge bases are designed from the perspective of the domain instead of the system’s characteristics.

There are also many reasons for using rule-based expert systems instead of object-oriented systems, such as frames. For example, rule-based expert systems have been popular with knowledge engineers for a long time, mainly because a large number of successful rule-based expert systems have been built since the 1960s (Durkin, 1994). Furthermore, rule-based systems are powerful and probably one of the best methods for representing human knowledge (Durkin, 1994), and as they support the building of almost all kinds of knowledge-based systems, visual support will have great implications for system design. Moreover, rule-based systems are more advantageous than object-oriented systems since people do not think in an explicit object-oriented fashion, but rather in an action-consequent manner, so production rules should be supported in the knowledge acquisition tool.

However, applying the UML directly in rule-based systems is not straightforward since object-oriented programming differs from declarative programming, and, therefore, the diagrams of UML have to be remodelled to fit the rule-based system. In rule-based system development, diagrams are applied for both the design user and the end user, i.e., they have to illustrate the relevant knowledge to be handled by the system when modelling and during consultation.

UML is used in the knowledge acquisition process since UML is a modelling language used when constructing software systems. Moreover, the UML’s diagrams can give a better understanding of the system as the reality is modelled in the diagrams. The diagrams constitute a simplification of that reality, i.e., a model of it. Models of UML are, among many things, used for capturing domain knowledge so that, e.g., engineers and end users, may
understand and agree on them and used for generating usable work products, e.g., user interfaces and databases. The models are also used for organising, finding, retrieving, examining and editing information in large systems and to master complex systems, i.e., dealing with complexity that is too difficult to deal with directly (Rumbaugh et al., 1998). Thus, the modelling facility of UML simplifies the problems associated with the human being’s ability to understand complex systems by focusing on one aspect at time. The modelling also supports dealing with and comprehending smaller and more easily solved problems. It uses a divide-and-conquer approach where a complex problem is attacked by splitting the problem into smaller ones.

Our remodelled UML diagrams have been tested on the knowledge base of the EIA system. These diagrams were applied on the rules of the knowledge bases as well as the programming code of the EIA system. This appliance illustrated that some diagrams enable different users to share them for different purposes. The test also shows that it is possible to use the diagrams to model knowledge-based systems and to cover the content, the functionality and manage the reasoning process. For the moment, we are implementing these diagrams in the KANAL-system. By implementing the diagrams in the KANAL-system, a powerful shell can be obtained that operates like a tool for the design user. It can also provide a powerful help facility for the end users during the execution of the system, especially in the consultation process.

It is often fairly efficient to let different users, design users and end users, share the same view of the system since this view can make it easier for the end users to comprehend the domain in a similar manner as the design users. Both the domain expert and the end user should utilise several different views in the development process and during the consultation phase for receiving additional knowledge about the contents and the questions to the users.

Visualising the contents of the knowledge-based system is only a partial step towards helping the design user to develop the system and helps the user to structure the domain knowledge. Supporting the entire knowledge acquisition process requires visualisation of the insertion process, reasoning process and functionality of the system.

One problem with visualising the functionality using UML diagrams is that it visualises programming code in a procedural manner by using a kind of state-chart diagram. If the system is developed in declarative programming language, these diagrams are not be appropriate to support the users when they are expanding or modifying the functionality, although, there is often a procedural reading of declarative programs. Therefore, we suggest a visual programming language as a complement to, extension or replacement of the state-chart diagram.
In an early attempt to use the modified UML’s sequence diagrams, these were applied to the EIA-system’s knowledge base to display the syntax form of the rules. However, using rule numbers and objects that corresponded to questions did not give the expected result. To understand and use a rule, the users had to search for the specific rule that the number referred to. This was not our intention with visualising the knowledge base. Therefore, conceptualisation is to be incorporated in knowledge acquisition.

4.3 Article 3: Visual Conceptualisation for Knowledge Acquisition in Knowledge-Based Systems

In this paper we suggest the use of visual conceptualisation as a means of improving the semantic understanding of the formalisation of the domain knowledge (Håkansson, 2003:b). This visual conceptualisation simplifies the users’ understanding by reducing the complexity of the knowledge base, and supports alteration of the reasoning strategies in a knowledge-based system. The conceptualisation implies perceiving rules as semantic concepts where the concepts apply different meanings to the rules. That the conceptualisation is “visual” implies that the semantic concepts are ordered spatially in diagrams. These concepts illustrate the structure of the base for knowledge acquisition as well as the reasoning strategy.

Conceptualisation supports knowledge modelling by clarifying the meaning of the rules in large and complex knowledge bases (Håkansson, 2003:b). To assist with keeping track of the knowledge and to check its correctness the contents of the rules are made transparent. To simplify the understanding of the rules in the knowledge base, a concept, which is explaining the purpose of a rule, is applied to that rule. This concept indicates the contribution of the rules in a particular context. Using an informative semantic concept moves the comprehension of the rules from a physical structure to a more conceptual knowledge management structure by, e.g., providing an understanding of a specific rule’s purpose or its applicability to other rules without the need to actually penetrate the inner contents of the rules.

By structuring the rules in the knowledge base, the concepts are utilised to grasp the content in the rule base and made to present the structure of the knowledge base through the interface. In a structured knowledge base the semantic concepts are of assistance in finding and reusing the existing and meaningful rules by clustering rules with similar concepts in the knowledge base and presenting these clusters to the users. In addition, it is reasonable to provide structural concepts for classifying the rules into individual groups of semantically related members, e.g., rules that are clustered on the basis of a
feature shared by these rules. For instance, one such feature could be the similarity of a set of tasks.

The concepts are presented at different levels of abstraction. The ability to use general concepts as collective terms for the groups of rules composed allows rules to be built at different levels of abstraction instead of building rules only at the lowest and most detailed level. Supplying levels of abstraction enables a different number of rules to be presented in the knowledge base at a given time and thereby minimise the complexity. Hiding currently irrelevant information gives a better understanding of complex information spaces or knowledge bases and the users focus on the specific knowledge chunks.

Knowledge acquisition using conceptualisation is illustrated visually in a sequence diagram of Unified Modelling Language (UML). The sequence diagram shows the declarative knowledge in a procedural manner by presenting the contents of the knowledge base in sequences. It illustrates the connections between the different rules and the associated facts by displaying them as concepts with their relationships. The diagrams are used to create and modify the rules and to check how these rules are related, that is, the manner in which the rules are connected to each other in the knowledge base. The sequence diagram can support recognition since it helps the users to remember the rules that have been defined.

The conceptualisation constitutes a knowledge management hierarchy. The concepts are illustrated graphically in a hierarchical manner with the concepts or topics being the nodes and the relations between them connecting the concepts. Hierarchically, the most general concept is placed on the top with the most distinctive concepts at the bottom. The form of presentation chosen illustrates the complete knowledge base with the whole range of rules from the general concepts to the more distinctive ones. However, the presentation can leave some of these concepts unexpressed, or illustrate rules with the focus on either a cluster of concepts or a specific concept. Moreover, illustrating concepts hierarchically allows a simplified representation to be made of complex structures and permits rules to be built and visualised and their structures shown at different levels of abstraction. If concepts are clustered, a cluster-based visualisation supplies an overview of the contents of the collections.

### 4.3.1 Results and Discussion

Conceptualisation has been applied to the sequence diagrams that are graphic representations of the EIA-system’s knowledge base (Håkansson, 2003:b). Conceptualisation of rules supports knowledge acquisition by keeping track of the knowledge inserted and by controlling the correctness of the knowledge through individual, semantic rules and sets of rules. In
knowledge-based systems, the notion of a concept portrays the semantics of a rule and applies a meaning to that rule. This understanding is utilised to change the users’ comprehension of rules and even their understanding of the order in which the interpreter traverses the rules. The semantic conceptualisation also simplifies the comprehension of the rules in a knowledge base by clustering similar rules. Such clustering shorten the search for similar rules in the knowledge base, facilitate generalisation of rules, reduce the number of similar rules and clarify redundancy.

Allowing the visualisation of the structures of the rules at different levels of abstraction instead of only visualising rules at the lowest level is a powerful support because it is a supplement to the sequence diagram and thereby assists the domain expert during the knowledge acquisition process (Håkansson, 2003:b). The concepts or topics should be used actively to provide support for the conceptual model of the content in the system. It is preferable to present the model to the design user, who may continuously interact with it to avoid, e.g., verification problems.

The immediate concepts are presented to each concept. Using this facility can simplify the process of remembering the purpose of a concept used since it can support recognising the rule’s role in a context instead of recalling the motive for specifying the rule\textsuperscript{14}. A rule (or rules) is easily recognised if it is presented with an explanation of its purpose and with similar rules. Otherwise, the rule using a particular concept has to be explored independently of other rules to identify the purpose.

Clustering of rules with the aid of conceptualisation supports the definition of concepts at different levels of abstraction. Recognising similar rules at an abstract level allows these rules to be generalised. Moreover, clustering with conceptualisation decreases the search for rules that are handling similar tasks or topics and eliminate redundancy.

The interface presents rules at various levels of abstraction by utilising clusters of rules. This requires using concepts to illustrate the correlation between the rules. Applying concepts to rules allows the building and visualising of structures of the rules at different abstract levels instead of only at a detailed level.

The use of concepts assists in finding relevant and meaningful rules for search facilities by searching for a specific rule from a specified concept. The searching mechanism can be simple or more sophisticated. It either search for a specific rule using a semantic concept explaining an accomplished sub-task, or search for related rules, i.e., rules with a similar internal structure. The latter type of search can be used for consistency

\textsuperscript{14} The process of recognition is relative automatic whereas recall in this sense involves having to actively look back through something to find where it was met before.
checking when rules are updated or changed to avoid accidentally altering a rule used by other rules.

4.4 Article 4: Supporting Illustration and Modification of the Reasoning Strategy by Visualisation

This paper describes the use of visualisation to illustrate and modify reasoning strategy by using conceptualisation (Håkansson, 2003:c). To visualise the reasoning strategy of the system, concepts are displayed in diagrams to describe the rules in the knowledge base. Moreover, sequence diagrams are used to illustrate and change the reasoning strategy. The visualisation includes both static and dynamic presentations of the rules and facts involved in drawing a conclusion. Since it concerns strategies, the presentation needs to be illustrated through a stepwise execution. The visualisation is presented statically in UML sequence diagrams and dynamically in UML collaboration diagrams.

Sequence diagrams and collaboration diagrams apply conceptualisation to obtain an overview of the contents of a system by illustrating the rules of the knowledge base and their relationships. The diagrams are used to create and modify the rules and to check how these rules are statically and dynamically related. Static information reveals the manner in which rules are connected to other rules in the knowledge base, while dynamic information depict the rules connection during the execution with user-supplied domain knowledge. Another strength of sequence diagrams and collaboration diagrams is that they presents the interpretation of the knowledge base by displaying time sequences or specifying the appropriate order of interpretation for the rules. The diagrams demonstrate how different parts interact with each other, e.g., how rules are related and interdependent. Sequence diagrams display the static relationship of the rules, from which the users determine whether the behaviour of a system is acceptable by investigating its performance as displayed in the diagram. The sequence diagrams are also used to ensure that the rules that have previously been defined are not overlooked. Collaboration diagrams display dynamic relationship of rules and give an overview of relations between the input, the rules and the output and allow extensive control over the relationships. It is also possible to elaborate this diagram to control the outcome of different inputs, and to specify the rules involved in arriving at a result.

Moreover, these diagrams are used to visualise explanations. For the why explanation, the specific question is displayed with the rules and conclusions that are relating to the question in a sequence diagram. For the how explanation, the specific conclusion is displayed with the relevant rules and
facts that have been used in arriving at the conclusion in a collaboration diagram.

4.4.1 Results and Discussion
Several different reasoning strategies, e.g., deductive reasoning, inductive reasoning, abductive reasoning, and case-based reasoning, can be used in a knowledge-based system. Only a few of these are interchangeable, e.g., deductive, inductive and case-based reasoning can be provided by the system as pre-defined strategies. To achieve a successful system, that is, one which is sufficiently powerful, it is necessary to provide the domain expert with the opportunity to develop a strategy. However, in our illustration and modification of the reasoning strategy, the technique represented in a system is limited by the strategies that are usable in a system solving diagnosis and classification problems, i.e., deductive and inductive reasoning.

The problem with making a change in a reasoning strategy is that the design users have to know about all the strategies to utilise one of these strategies. Otherwise, the system has to be designed to allow the developing user to implement a new strategy or change an existing one. The interface has to assist the user in this process, by providing various means of input. For instance, one such method is direct manipulation where the rules and facts can be visually restructured on the basis of a reasoning strategy, which we have utilised in our approach (Håkansson, 2003:c). However, this may not be suitable for all kinds of reasoning strategies.

As mentioned before, the sequence diagrams have been applied on the knowledge base of EIA system. In this work, however, the sequence diagrams and the collaboration diagrams have been successfully applied to a knowledge base that handles childhood diseases (Håkansson, 2003:c). The reason for utilising this knowledge base is that it was used as the task during the observation sessions in the usability test, as described earlier (Håkansson, 2002). The assumption is that provided that the diagrams are applied to the knowledge base, these diagrams can also be utilised in the user interface of the KANAL-system and a new usability test with the same task can be realised.

As a result of this test, we think that assignment of concepts provides complimentary information to the user by presenting and explaining the concepts already utilised (Håkansson, 2003:c). This can support the user’s comprehension and, moreover, the developer’s recall, and thereby supporting the recycling of modelling processes. Moreover, active usage of concepts can support developing systems. If the contents of the diagrams are constantly presented to the engineer who develops the system, this engineer can interact with the concepts to avoid quality assurance problems, e.g., verification with redundant, conflicting, subsumed or circular rules. This
requires a real-time system that continuously updates the concepts and their relationships in the diagrams simultaneously as making the changes to the knowledge base.

4.5 Article 5: ViCoLL – a Visual Compositional Logic Language
To add and modify functionality of a system, we have developed a user interface to a compositional declarative relational programming language (Håkansson et al., 2001). Facilitating program development in a compositional declarative relational programming language is achieved through visualisation of the operations and the components that together compose the program. These reflect the set theoretical semantics of the language. These visualisations are employed in a visual compositional logic programming language for adding or modifying functionality in knowledge-based systems.

This visual programming language uses a graphical interface by using a combination of two different visual logic diagrams, Venn and Euler diagrams (Håkansson et al., 2001). Traditionally, graphical diagrams have been used to illustrate relations between sets, and the most popular and intuitive ones are the Venn and the Euler diagrams. Venn diagrams are expressive as a visual notation for sets and for the relationships between sets. Euler circles are considered more intuitive and are easier to draw. There are of course also drawbacks to the use of these types of diagrams: In a Venn diagram, it is difficult to comprehend all intersections and the Euler diagram is not as expressive as the Venn diagram because it lacks the shading.

To reduce the impact of the drawbacks mentioned whilst taking advantage of the benefits they offer, a combination of these diagrams has been used, i.e., these diagrams are composed into a so-called Venn/Euler diagram. This particular combined diagram differs from the original ones, as well as from other combinations of Venn/Euler diagrams described by other authors (Gil et al., 2000; Puigsegur et al., 1997). Our diagram, for example, does not present sets through the use of circles within circles as in Puigsegur et al. (1997) combined Venn/Euler diagram.

Using two different representation forms solves the intersection problem. One representation form is an elliptic figure, which displays disjunction. The other figure is a circle, which in connection with another circle represent intersection. To maintain the feeling of a united program, the and-circles are connected by the elliptic figure. Furthermore, the shadowing facility has been retained in the diagram and is utilised when two different and-circles are used and put together or if an and-circle is used in conjunction with an or-circle.
Using Venn diagrams and Euler diagrams usually produces cluttered interfaces when many sets are involved. Using too many objects in an interface may also produce clutter. To support the using of more sets and avoid clutter, three-dimensional representations (Kakuya, 1996; Najork, 1994; Oshiba & Tanaka, 1999) are utilised in the representation of the sets. An extra spatial dimension simplifies the presentation of additional sets. Another advantage of three-dimensional representations is the fact that humans tend to remember structures presented in three dimensions better than structures presented in two (Tavanti & Lind, 2001). This is useful when identifying the purpose of a particular part in a system and also when supporting the reuse of these structures.

The idea of using visualisation in a programming language has been tested on the compositional logic programming language CombiLog (Hamfelt & Nilsson, 1998). In contrast to traditional logic programming languages, programs (predicates) in CombiLog are composed from simple programs (more primitive predicates) by means of predicates composition operators. CombiLog’s semantics coincide with the usual semantics of logic programming languages. Moreover, CombiLog uses intuitive forms of problem descriptions such as predicates and operators. With this language it is difficult to develop a program by writing code, but if graphical objects are used in the interface, the problems with complexity of the implementation can be reduced. Since the programming language CombiLog is compositional, the appropriate visual interface is a Visual Compositional Logic Language (ViCoLL) (Håkansson et al., 2001).

4.5.1 Results and Discussion
A graphic interface must provide an easier programming environment for a domain expert without computer skills who could conceivably not know any programming languages and who might not be familiar with the internal structure of the system. ViCoLL has been developed to support programming at a fairly abstract level and has been successfully applied to CombiLog’s programming code (Håkansson et al., 2001). It captures the important predicates and operators of CombiLog and supports combining programs. Thus, it is possible to use the visual programming interface to provide such a graphic interface with textual programming code.

The idea of using a combination of Venn and Euler diagrams is an appealing one and has proved to be applicable for a set-theoretical semantic programming language (Håkansson et al., 2001). The declarative interpretation of the code in the interface has to be preserved, but it is difficult to avoid a procedural interpretation of the screen. Humans tend to interpret the objects placed on a screen in the same manner as they read books: from left to right and up to down. This will be influential when
reading on the screen because, unfortunately, this interpretation is probably unavoidable, but to avoid increasing the procedural interpretation, no meaning or notions will be applied on the axis, i.e., on the diagonal from top left to bottom right/one the axis of the figure. Furthermore, the users are allowed to place the objects wherever they like. When the interpretation order becomes significant, a question to the user provides a starting point for the execution, to ensure that it follows the interpretation.

To study the usefulness of this visual front-end, ViCoLL has been tested with small number of users. The users penetrated both the text-based CombiLog and the visual ViCoLL. It seems that these users understood ViCoLL slightly better than CombiLog. An initial reaction revealed by a user evaluation was that there was a lack of definition of the terms that the operators and components work with as well as a difficulty identifying the explicit connection between the operations and the components. Therefore, the layer between the abstract level and the syntax must be made more transparent to better support the users while they program the system. The absence of an explicit connection probably also affects the users’ sense of having the programming under control. This is an especially important aspect when the users are quite skilled programmers. The correspondences between different elements in the different sets, which have already been dealt with to a certain extent, need to be made more explicit to the users. Moreover, the users had problems with the tuples, which are used in the sets.

4.6 Article 6: Visualisation of Declarative Relational Programming

This paper extends the study in the previously described in Article 5. It describes the use of a combination of visual logic diagrams, Venn diagrams and Euler diagrams, to represent the programming code (Håkansson & Hamfelt, 2003).

ViCoLL makes use of CombiLog’s programming structures to build more abstract structures. To preserve the declarative interpretation of the code, hide the procedural execution flow and, thereby, simplify the code development in ViCoLL, the visual programming resides at a largely abstract programming level. This implies programming without displaying the correspondence between the visual objects’ and to the programming code and without explaining the collaboration between elements of different sets. Moreover, this paper visually illustrates operators for the single case, recursive case and construction case, or case, texts and tuples. These are presented in a menu bar from which a user chooses the operators.
4.6.1 Results and Discussion
ViCoLL’s interface is designed to support the user when he or she is about to change the functionality or develop new systems. A program is constructed in a set-based manner by composing operators together from smaller components in a Venn/Euler diagram. In the longer perspective, a major contribution of ViCoLL will be to facilitate and simplify programming-in-large. However, using a modified Venn/Euler diagram with or-circles leading outside the Venn diagram and to couple another Venn diagram may lose the semantics of being a finite collection of contours. Whether this will have major effects on the work or not, must be investigated.

The graphical objects in the menu bar have been applied to the predicates and operators of CombiLog, which they also cover. Therefore, these objects are used when programs are composed recursively from primitive predicates and composition operators.

The visualisation is to be tested as an interface for a commonly used logic programming language often used in knowledge-based systems. To be assured of the possibility of using the visualisation for changing the code of the KANAL-system, the interface of ViCoLL has to be applied in the KANAL-system and in some of the diagrams to support the design user.

With a carefully developed visualisation of a programming language, the domain expert or the design user may be able to develop new functionality as well as change the existing functionality of even a fairly complex system with greater ease. Naturally, this is not accomplished instantaneously and there will be many obstacles to consider. For instance, such visualisation requires figuring out exactly how graphical objects can best be applied in an interface to support users. To a large extent, this depends on the shaping of the users’ mental picture of the system since the users’ rate of grasping the information depends on the level of correspondence between the internal image of the system and the representation of the system.

4.7 Article 7. Dynamic Libraries as a Means for the Management of Information Browsing in Networks
A solution to reduce the problems with complexity of and facilitate navigation in hypermedia systems is the support of the comprehension of the links by attaching a semantic interpretation to them (Håkansson & Oestreicher, 2000). A hypermedia system consists of a large number of clickable links where each link represents a large amount of information, both explicitly and implicitly. The links are regarded as a means of putting together different pieces of knowledge into a knowledge network. However, the real purpose of the link is not always transparent since this knowledge is
often hidden in the linkage. In these cases, when the end user uses the link, an unknown — at least in all its detail — action will be performed. Thus, the actual consequence of the action may be more or less unknown before clicking and the link has to be followed to disclose the action. Increasing the transparency and applying a semantic association explaining the link would announce the event more clearly to the user. This is valuable when the user tries to decide whether the link is a useful or interesting resource without having to explore its content.

Direct manipulation in a network should, in essence, be equivalent to the user moving around in a conceptual navigation network, but as the network is formed as a physical structure it is usually imposed unconditionally on the user and, therefore, more or less guides the user’s experience. For the user, finding relevant information implies moving between different pages using links without knowing their actual action and where they lead. Using semantic associations to grasp the meaning of the links instead of using plain links, i.e., using conceptual links, could give a better understanding of large information spaces (Håkansson & Oestreicher, 2000). An improved awareness of the meaning inherent in the links could give a better understanding of large information spaces. For instance, if the user asserts a concept and the system will attempt to provide the user with supplementary support by finding the relevant links that lead to appropriate pages concerning the concept.

Information concerning the user’s interest is accumulated over the sessions and stored as concepts, in combination with the semantically related group of links, to produce a user profile. The resulting profile is designed as a dynamic library containing topics. Topics can be described in terms of larger conceptual structures and sets of links (Sowa, 1984). In the library, each topic is associated with keywords (or concepts) with the keywords describing the meaning of the topic. Keywords are also used to search for relevant information in the system.

To facilitate the changing of characteristics dynamically and the addition of new topics, deletion of irrelevant topics and updating of altered ones we suggest a dynamic library (Håkansson & Oestreicher, 2000). This library should support searches for relevant information by informing the user about the data expected to be found at the end of a link, e.g., pure text, images or sound, as well as the number of times the concept is found within each page referred to. The concepts in the library are presented at different levels of detail or abstraction and folding and unfolding facilities are used for the management of the branches holding the concepts.
4.7.1 Results and Discussion

Navigating through large hypermedia systems to obtain valuable information using a keyword search engine is very complicated. Within the scope of this paper, it is not worthwhile attempting to discuss the problem of the large number of links provided by a search engine (Håkansson & Oestreicher, 2000). Instead, the focus here is on how it is possible to check whether it is worthwhile exploiting the links within a page. The name of the link may be adjusted for the intended purpose, from the Web designers’ perspective, but this may not be desired information association for the end user. Furthermore, an end user may get the wrong impression of the action performed by the link, since they connect with the wrong causal events. It is the user’s association that is assigned when he decides upon the Keywords and these can be misinterpreted or can give a misleading intention. Another problem is that the developers can use concepts that will impart a totally different meaning to the user because of the ambiguity of the chosen words. Completely avoiding misinterpretations is almost impossible. However, assigning semantics or meaning to the links will constitute a complementary development environment for the end user.

Introducing direct manipulation and formation of concepts in networked applications transfer the users’ activities from a basically “physical link network structure” to a kind of “conceptual knowledge management network structure”. A conceptually structured knowledge management network provides a specific user with the user’s own predefined structures and give the end user opportunities to create sets of personal knowledge links. This implies centring and orienting the network around the individual user’s “knowledge space” rather than imposing predefined, available structures on the information. This conceptual network is then used in future interaction to make the navigation more closely adapted to the individual. It also opens up the way to making the activities more akin to acting upon the contents than searching for physical documents and links. However, the extent this conceptual knowledge management network structure works, will be studied.

The contents of a dynamic library have been applied to links at a web page. For that page, the dynamic library presents and explains existing concepts and relationships to support the user’s comprehension of the links (Håkansson & Oestreicher, 2000). It also supplies knowledge about former interactions by moving around the concepts of the library. The library is dynamic in the sense of continuously needing expansion and maintenance during interaction. It is altered concurrently with the insertion, deletion or change of the concepts and their meaning. Graphically, the concepts are illustrated hierarchically by letting the concepts or clusters of concepts constitute the nodes with the relationships intertwining these concepts.
5 Concluding Discussion and Remarks

The subject of this thesis is the adoption of user-centred modelling — a graphic representation and visualisation modelling methodology to support the knowledge acquisition process, and especially the modelling of domain knowledge. First and foremost, this research is of interest for the modelling of domain knowledge, including the insertion and modification of the knowledge, as well as for modelling the reasoning strategies and the functionality. Furthermore, these measures support the elicitation of the knowledge from the domain expert, but the degree to which this can be achieved will be the subject of a future investigation.

5.1 Result of the Research

Using a methodology for incorporating the different users in modelling and consulting domain knowledge, reasoning strategy and functionality is highly recommended. If the domain experts are involved in the knowledge modelling from the beginning of the development, and if the domain experts actually see the contents of the knowledge base and reasoning strategies, the probability of maintaining the validity and confidence of the knowledge-based system can be increased. Moreover, if the contents of the system is adequate with respect to the domain and the system uses suitable reasoning strategies, the conclusions can be assured to be valid and the end users can be confident with the result of a consultation. In addition, if the end users see the contents and strategies, they can also check conclusions more reliably.

The involvement of the domain experts and end users requires enhancing the transparency, thus supporting the modelling of content, which we achieve by utilising graphical descriptions of the content. We also illustrate that static and dynamic graphical descriptions can enhance the transparency of the reasoning strategy and support the modification of that strategy, which is necessary during the development of and consultation with a system.

Moreover, this requires making it easier for the users to grasp the contents of the knowledge base, thereby reducing the problem with complexity of that knowledge base, and facilitating the alteration of reasoning strategies in knowledge-based systems. We suggest introducing conceptualisation to facilitate this.

For adding or modifying functionality in declarative programming language we offer using a visual compositional logic programming language, ViCoLL. It reflects the set theoretical semantics of the language
by visualising operations and components of program composition. We believe that once it is learned, it is easy to comprehend.

To assimilate domain knowledge into knowledge-based systems, we propose utilising conceptualisation on top of links in hypermedia systems, i.e., the Web, and then incorporate this knowledge into the system.

5.2 Further Work
The modelling tool illustrates some of the suggested support activities and is not a complete product. In future work, the diagrams will be further extended to include the handling of quality assurance by verification and validation, and uncertainty. Although the diagrams have been modelled for production rule systems, the knowledge representation of the system is not fixed and the diagrams can easily be used in a hybrid system using rules and frames. It is also important to investigate the extent to which the graphical representation and visualisation can support verification and validation of the system.

Modelling knowledge with a modelling language cannot support modelling the entire part of the elicitation in knowledge acquisition because the part in which the domain expert actually extracts the initial domain knowledge can never be omitted. However, the system may support the later parts of knowledge elicitation. Usually shells are quite sufficient for automating the acquisition techniques, but they should also be able to automate the eliciting technique.

In this thesis, building a conceptual model in the interface suggests building specific interfaces for one or a few domains for diagnosis and classification systems, i.e., for one or a few design users. However, it is interesting to build a general-purpose system for a range of users. The interfaces should not be domain specific and the objects corresponding to knowledge should be general enough to be utilised in many kinds of domains.

The diagrams used for conceptualisation have yet to be tested on design users and end users. Displaying the reasoning strategy of the system graphically may not be enough to enable the strategy to be understood and to support alterations. Moreover, graphical modelling with diagrams may not be adequate for all kinds of domain knowledge. In addition, all different knowledge types, such as tacit knowledge, have not been supported in the diagrams. This requires a considerable amount of in-depth testing within different domains to investigate the need for additional support for some kinds of knowledge acquisition. It also requires testing to verify that the different knowledge types are sufficient to express knowledge and to educate
users that are learning the domain and the system and, if not, incorporate other types.

ViCoLL is not yet integrated in the UML’s diagram. Since UML does not support declarative characteristics ViCoLL has to be incorporated to assist development of declarative knowledge-based systems. Either ViCoLL has to be incorporated into UML as a detached part or into UML’s state-chart diagram. Moreover, as also mentioned above, the diagrams used in the modelling are not used for verification and validation. This facility is supported on a small scale in the KANAL-system, but the existing diagrams should be extended to handle this quality assurance in real-time, i.e., during the modelling process.

Employment of natural language has not been considered in the interface when making proposals for modelling the domain knowledge. This will be investigated in the future.

Uncertainty calculations were considered in the EIA-system (Håkansson & Öijer, 1993), but not in the shell or in the modelling work. This facility should be incorporated into the tool because some systems cannot operate without it.

The educational aspect of the knowledge-based system, i.e., where the system teach the users how to use the rest of it, has not been addressed in this work. This is a significant aspect since a system may not be used without any tutoring being provided to the design users and the end users. Therefore, this is a matter that will have to be considered later on.

People seem to recognise knowledge more easily than they can recall it. It would be excellent if the system utilised the recognition to support the domain expert. BUT does it?

The knowledge transferred between users includes both experts and novices. There is a difference between experts and novices in the field of computing, both quantitatively and qualitatively. Experts perform better than novices on quantitative measures, but they also demonstrate qualitative differences from novices in their use of representations and strategies, i.e., the experts are more knowledgeable about their domain and experts can apply and use this knowledge more effectively than novices, (McGraw & Harbisson-Briggs, 1989). There is an evolution of memory of knowledge from novice to expert where, initially, knowledge is incrementally refined on the basis of specific episodes. Facts that might initially have appeared to be discrete and unrelated can be grouped together in accordance with concurrent use of the facts. A great deal of experience of using this kind of knowledge enables the reasoning process to be refined and opens up the possibility of determining the usefulness and adaptability of the existing rules.
A usability test is necessary to investigate the possibility of using graphical modelling techniques to develop knowledge-based systems. This test must also investigate the possibility of modelling domain knowledge and reasoning strategies.

The research in this thesis has been conducted to support different users while they work with software systems and, in so far it has used visualisation to realise this goal. It is not only knowledge-based systems that need to address the problem of not being sufficiently user-centred. It would be interesting to continuing working with similar problems and systems. For example, visualisation could be used for constraint programming or for knowledge representations such as frame-based systems. It could also address other systems operating on knowledge, such as knowledge management systems.

References


Euler, L., 1768. Lettres à une Princesse d'Allemagne, St. Petersburg.


http://ksi.cpsc.ucalgary.ca/articles/KBS/KSS0/


78


Vanthienen, J., Dries, E., 1994. Decision Tables: Refining the Concept and a Proposed Standard, accepted for publication in Communications of the ACM.


