

# Towards Continuous Knowledge Engineering

---

Ph.D. Thesis

Manuscript Concept, [14.10.2001], Version 1

Klaas Schilstra  
kschilstra@yahoo.co.uk  
+44 (0) 1383 861066



# Towards Continuous Knowledge Engineering

PROEFSCHRIFT

ter verkrijging van de graad van doctor  
aan de Technische Universiteit Delft,  
op gezag van de Rector Magnificus prof. ir. K.F. Wakker,  
voorzitter van het College voor Promoties,  
in het openbaar te verdedigen op dinsdag 26 Februari 2001 om 13:30 uur

door

KLAAS OTTO SCHILSTRA

ingenieur informatica,  
Technische Universiteit Delft,  
geboren te Oudega, Hemelumer Oldeferd.

Dit proefschrift is goedgekeurd door de promotor:

**Prof. dr. H. Koppelaar**

Samenstelling promotiecommissie:

<b>Rector Magnificus,</b>	voorzitter
<b>Prof. dr. H. Koppelaar,</b>	Technische Universiteit Delft, promotor
<b>Prof. dr. ir. J.L.G. Dietz,</b>	Technische Universiteit Delft
<b>Prof. dr. H. J. van den Herik,</b>	Universiteit van Maastricht
<b>Prof. dr. ir. E.J.H. Kerckhoffs,</b>	Technische Universiteit Delft
<b>Prof. dr. ir. M. Looijen,</b>	Technische Universiteit Delft
<b>Dr. ir. O. C. G. Adan,</b>	TNO Bouw, Rijswijk
<b>Dr. J. Krugers,</b>	TMC, Ventouse, Frankrijk
<b>Prof. dr. ing. habil. W. Gerhardt-Hackl,</b>	Technische Universiteit Delft

This research was supported by TNO Netherlands Organisation for Applied Scientific Research and the Ministry of Economic Affairs.

All trademarks contained within this document are the property of their respective owners.

Coverdesign by M. Koorneef

*Published and distributed by:* DUP Science

DUP Science is an imprint of  
Delft University Press  
P.O.Box 98  
2600 MG Delft  
The Netherlands  
Telephone: +31 15 27 85 678  
Telefax: +31 15 27 85 706  
E-mail: [dup@library.tudelft.nl](mailto:dup@library.tudelft.nl)

ISBN 90-407-XXX-X

Keywords: knowledge systems, knowledge engineering, metaphor

Copyright © 2001 by K.O. Schilstra

All rights reserved. No part of this material protected by this copyright notice may be reproduced or utilised in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage and retrieval system, without written permission from the publisher: Delft University Press

Printed in the Netherlands

*To Mandy*



# Contents

---

<b>CONTENTS.....</b>	<b>VII</b>
<b>PREFACE .....</b>	<b>IX</b>
<b>CHAPTER 1 INTRODUCTION .....</b>	<b>1</b>
1.1    KNOWLEDGE ENGINEERING .....	1
1.2    PROBLEM STATEMENT .....	2
1.3    RESEARCH GOAL.....	5
1.4    RESEARCH PROGRAM.....	7
1.5    THESIS OVERVIEW .....	10
<b>CHAPTER 2 PROBLEMS IN KNOWLEDGE ENGINEERING .....</b>	<b>12</b>
2.1    SYMBOLIC VS. SITUATED COGNITION .....	12
2.2    WHAT IS A KNOWLEDGE SYSTEM?.....	19
2.3    KNOWLEDGE REPRESENTATION .....	27
2.4    ENGINEERING KNOWLEDGE SYSTEMS.....	43
2.5    PROBLEMS AND CHALLENGES .....	51
2.6    SUMMARY AND ANALYSIS .....	57
2.7    CONCLUSIONS .....	61
<b>CHAPTER 3 TOWARDS CONTINUOUS KNOWLEDGE ENGINEERING ..</b>	<b>62</b>
3.1    METAPHORS .....	62
3.2    WHAT IS CONTINUOUS KNOWLEDGE ENGINEERING?.....	67
3.3    FROM PRINCIPLES TO REQUIREMENTS.....	73
3.4    FUNDAMENTAL METHOD, ESSENTIAL TOOLS .....	79
3.5    ARCHITECTURAL AND TECHNICAL CONCERNS.....	86
3.6    EVALUATION METHOD.....	94
3.7    CONCLUSIONS .....	102
<b>CHAPTER 4 A SIMPLE TOOL .....</b>	<b>105</b>
4.1    SIMPLICITY VERSUS COMPLETENESS .....	105
4.2    MODELLING PRIMITIVES .....	111
4.3    KNOWLEDGE MODELLING.....	122
4.4    KNOWLEDGE SYSTEM DEVELOPMENT .....	127
4.5    CONCLUSIONS .....	134
<b>CHAPTER 5 AN ADVANCED TOOL .....</b>	<b>137</b>
5.1    COMPLETENESS VERSUS SIMPLICITY .....	137
5.2    MODELLING PRIMITIVES .....	144
5.3    KNOWLEDGE MODELLING.....	161
5.4    KNOWLEDGE SYSTEM DEVELOPMENT .....	169
5.5    CONCLUSIONS .....	170
<b>CHAPTER 6 CASE STUDIES .....</b>	<b>173</b>
6.1    OVERVIEW .....	173
6.2    BUILDING MATERIALS REGULATION KNOWLEDGE SYSTEM .....	175
6.3    MASONRY DAMAGE DIAGNOSTIC SYSTEM .....	183

6.4	MOISTURE DAMAGE DIAGNOSTIC SYSTEM .....	193
6.5	FIRE REGULATIONS ADVISORY SYSTEM .....	199
6.6	MEBIS .....	214
6.7	CONCLUSIONS .....	224
<b>CHAPTER 7 EVALUATION .....</b>		<b>227</b>
7.1	RESULTS OF CRITERIA EVALUATION .....	227
7.2	EVALUATION OF SOLUTIONS.....	237
7.3	CONCLUSIONS .....	244
<b>CHAPTER 8 CONCLUSIONS AND RECOMMENDATIONS .....</b>		<b>247</b>
8.1	CONCLUSIONS .....	247
8.2	RECOMMENDATIONS .....	252
8.3	CONTRIBUTIONS.....	254
<b>SUMMARY.....</b>		<b>255</b>
<b>SAMENVATTING.....</b>		<b>257</b>
<b>BIBLIOGRAPHY .....</b>		<b>259</b>
<b>CURRICULUM VITAE.....</b>		<b>269</b>



# Preface

---

The research presented in this thesis was conducted within the Knowledge Based Systems Department of TNO Building and Construction Research. Working on the development of knowledge and information systems, the research of this department is directed to facilitating this process by the development of dedicated tools and techniques. This provides the practical background of the research described in this thesis.

Within this context the problem with knowledge system development felt most directly was the difficulty with which a knowledge model could be brought from an initial prototype to a professional system that would conform to the expectations of the end-users. Furthermore, the problem of maintaining and updating the knowledge contained in such a system was seen as just as great a problem as it was to acquire it in the first place.

Being in a practical context meant sometimes pragmatism and theory conflicted. Priority was given to the practical approach. This will be mentioned in the text. The applied nature of the research environment was felt as an essential factor in the development of the ideas in this thesis.

My thanks ...

The word expert will be used to denote the ‘owner’ of the knowledge. Nowadays, this person is not necessarily an expert, but the term succinctly states what kind of person or role is intended. To show exactly how inconsistent present nomenclature is, the term ‘expert system’ has evolved into ‘knowledge system’, to stress the fact that the intent is that the system contains knowledge, rather than attempt to be an expert in some field. This thesis therefore discusses knowledge systems based on the knowledge of experts that are not necessarily experts in their field.

Furthermore, and more generally, it may occur within this thesis that ‘man’, ‘he’, ‘his’ or ‘him’ term is used. I second the motion made by the now late Herbert Simon, that these pronouns should be seen as androgynous nouns encompassing both sexes, just as man denotes mankind, both male and female (Simon 1982, pg. 4). A side-effect of this approach is that by changing the meaning of these words, the current literature of the world can also be reinterpreted to have been politically correct all-along. That this also fits with the situated view on meaning and knowledge is of course beside the point.

On a typographical subject, quotations will be surrounded “with double quotes”, while single quotes will be used to accentuate or ‘stress’ certain terms.



# Chapter 1

## Introduction

---

*The beginning is the most important part of the work.*

– Plato

Life on earth depended solely on DNA to carry information from one generation to the next for a very long time. Then, in the animal kingdom parents started passing experience to their offspring by teaching the skills needed for survival and through culture. Human kind found ways to communicate knowledge through speech and writing. This has created our ability to retain knowledge and pass it on through time without direct contact. Starting with stories told by the fireplace, later written down in books, and eventually placed as hypertext on the Internet, our knowledge made us human. The essence of this, the scientific enterprise is based on this free dissemination of ideas, to enable us to make sense of the world around us. Our outlook on the world and methods of problem solving are based on the discoveries made by our ancestors, sometimes many hundreds of years ago. In that same time, a great deal of knowledge has been lost. Knowledge systems provide a new medium to retain and spread knowledge, and to pass on what has been discovered by others.

In this introductory chapter, a short background on knowledge systems and knowledge engineering is given as well as some insight into the research described in this thesis. To this end, a problem statement is formulated and the research goal is defined. This is discussed as to its practical and scientific content. In addition, the structure of the research program is provided. Finally, an overview of the remainder of the thesis is given.

### 1.1 Knowledge Engineering

Knowledge systems differ from all earlier media to retain and disseminate knowledge devised before. Knowledge contained in stories, books as well as scientific theories has to be acquired through training, study and reading. A *knowledge system* is a software system that makes a body of knowledge operational to solve problems using a separate representation of knowledge that can be employed dynamically. This aspect provides a great deal of flexibility and allows a knowledge system to approach a broad category of problems, without each of the problems being foreseen or enumerated in advance. This allows them to be used to solve problems that would require humans experience or training. In many cases, the origin of the knowledge lies with human experience and learning. To some researchers this is a defining aspect for knowledge systems (e.g. van den Herik, 1988).

*Knowledge engineering* is the discipline of developing knowledge systems using sound and principled methods. Perhaps because it is a relatively new field, the term ‘engineering’ is used to stress its relationship with other engineering disciplines. It evokes an image of trust and professionalism, and is meant to denote that it employs centuries of knowledge passed on and refined over time, on the development of artefacts. Using methods of design, implementation, deployment and maintenance, it is similar to the approaches found in construction and other engineering disciplines.

For their use within knowledge engineering, these methods have been adapted for the special needs of knowledge.

Knowledge engineering focuses on the relationship to engineering practices, in much the same way as software engineering has, which has also been a natural influence. Most current knowledge engineering approaches follow an engineering metaphor that approach a knowledge system as a finished product, for which the knowledge can be designed as part of the specification. This leads to a perspective on knowledge engineering that is constructivist in nature.

The problems found in knowledge engineering are often approached from the perspective afforded by that same engineering metaphor. Pure engineering solutions are used to overcome the problems caused in part by the nature and origin of knowledge. The view proposed in this thesis is that the root cause of these problems lies in the strict adoption of the engineering metaphor, because it goes against the dynamic nature of knowledge and its origin in learning.

All human knowledge has arisen slowly over time. Nature employs a different type of engineering. In line with this, a solution to some of the practical problems is therefore searched for at the level of a metaphor or more correctly, a shift in metaphor. Using a scientific metaphor, the shift is towards a continuous knowledge engineering process based on increasing insight, mirroring the learning in scientific development. In this perspective, the knowledge model is comparable to a scientific model to transparently retain a body of knowledge. This model can be invalidated by failed predictions or be replaced by a more elegant version. The knowledge system is more aptly described as a medium to make the knowledge model available to others. It is then still a product but the ongoing development is towards the improvement of the knowledge model over time.

## 1.2 Problem Statement

Knowledge has a substantial economic significance as a key resource for many existing and new companies. Increased awareness to the role and the importance of knowledge has led companies turning to knowledge management to improve the efficiency and effectiveness of their use of knowledge. In turn, this strengthened the attention for knowledge engineering and boosted the need for professional, industrial strength knowledge systems as a medium to retain knowledge and make it more readily available.

Developing knowledge systems is by no means an exceptional deed. The first knowledge systems were developed in the early 60s. Knowledge systems have since then been seen as AI's most successful commercial spin-off (e.g. Giarratano & Riley, 1994)). Other scientists have been quoted to say that there are "1000s of successful knowledge systems" (Lenat & Feigenbaum 1989), and some knowledge systems have shown to deliver enormous benefits, for example XCON (Bachant & McDermott 1984) which has saved DEC many millions of dollars. Most war-stories talk of prototypes knowledge systems that never got the chance to prove they could also provide such benefits (Crofts et al. 1989). And, as of yet, there are no known knowledge systems to date that rate in the category of killer-apps.

Knowledge systems suffer from a *bottom-line problem*.. This problem refers to the fact that the benefits of knowledge systems are often unclear, and coupled to a high cost and risk. The main benefit of a knowledge system is the explicit form of

knowledge and the availability of the knowledge. Either can only be measured through the use of the knowledge. The costs derive from the complexity of the systems and involvement of experts and knowledge engineers. Development of a knowledge system is in the majority of cases exploratory in nature, which has an associated risk.

There are different problems that can be targeted as root causes of the bottom-line problem. Following the terminology of Brooks (1987), these problems can be divided into intrinsic and accidental difficulties associated with the enterprise of knowledge system development. The intrinsic problems are fundamental and relate to the nature and origin of knowledge, as well as the complexities of software development. There are no quick fixes or even guaranteed solutions for these problems. The question is even whether such problems will ever see a solution. Accidental problems are those that have to do with inadequacies in current methods and tools, that may be solved in due time.

A knowledge system naturally depends heavily on the knowledge of the problem domain, often deriving from human experts. One of the root problems of the cost and risk associated with knowledge engineering is the intrinsic difficulty of formulating and formalizing knowledge. This is also known as the *knowledge acquisition bottleneck*. As this problem extends to the maintenance of knowledge systems as well, it is referred to within this thesis as the *knowledge acquisition and maintenance problem*. Most knowledge systems are developed by specialised knowledge engineers. They are well versed in interviewing techniques, analysing the experts, and additional material containing the pertinent information. By their training are also well prepared to enter such knowledge into a knowledge representation. Both knowledge engineer and expert need to work very hard at developing a good model of the necessary knowledge. Such a process is both time-consuming and expensive, and success cannot be guaranteed.

Even then, many knowledge systems never reach the point at which they are actually used by the people the system was intended for. Of a successful knowledge model or knowledge system prototype cannot be said that it will translate to a successful end product. There remains to be a great gap between a prototype and a commercial knowledge system, especially in the visualisation of knowledge systems, as compared to what users have become accustomed to in standard software (Crofts et al. 1989). For want of a better term, this is dubbed the *gap problem*. The acceptance of knowledge systems is not easy, as they do not restrict themselves to providing a service, but effect changes in the organisation and balance of knowledge. Knowledge is power, as the slogan goes.

The *brittleness problem* is a further challenge with knowledge systems. A knowledge system performs badly when confronted with problems just outside of its domain. This is compounded by not knowing when they might be wrong, or speaking outside of their scope. This effect is so strong that it is also called the knowledge precipice. The knowledge system performs well enough on the plateau, but very badly just outside of that. Some think this is caused by lacking depth in the knowledge that these systems have, others consider the missing part to be a large store of common knowledge.

Software systems are complex artefacts that are inherently difficult to develop. Knowledge systems represent a software system that in many cases is required to perform tasks that require more than a straight algorithmic approach or fixed

solutions. This makes knowledge system development the construction of solutions for difficult problems. This *complexity problem* places the development of a knowledge system firmly in same league as complex software.

This makes a demanding enterprise all the more difficult, as the introduction of such systems is already under pressure from fears of people that they are to be driven out of their jobs, neo-Luddite techno-phobia and other misconceptions. Combined with the uncertainty of the financial benefits to be gained, knowledge systems often have only limited commitment from the organisation. This means that management can easily kill knowledge systems when they do not produce results quickly enough. Different researchers cite company-wide commitment as a critical success-factor (Schreiber et al. 1999, de Hoog 1998, Stefik 1995, Steels 1992). Even XCON was cancelled by DEC three times during its life-cycle (McDermott 1982, 1980).

These problems combine to produce a bad bottom-line situation for knowledge systems, and give them even now a reputation of being experimental technology. The total effect of all of this is that knowledge systems have not made a great impact on the world as yet. Certainly not as much as was thought some 20 years ago, when knowledge systems were heralded as the expert-in-a-box. Knowledge systems were the perfect receptacle for mining the jewels in the minds of expensive experts, their valuable knowledge safeguarded by the organisation, and copied infinitely.

The bottom-line problem and its root causes are central issues to the future of knowledge systems application and research. There is a great need to 'sell' knowledge system technology to the greater public as a viable solution to practical knowledge problems. The research should focus on translating the possibilities of today to the applications of tomorrow. Increasing the profitability of knowledge systems, and making this clear improves acceptance of knowledge systems. Providing approaches and tools that can be used to realise the systems as cost-effective, high-quality solutions, strengthens both the science of knowledge engineering and the underlying science of AI.

By having more knowledge systems, there will be more knowledge on building knowledge systems. The cry for innovation will become greater as increasingly more complex and critical tasks are tackled, leading to new research initiatives. Research into the reasons behind the problems this poses and their possible solutions will remain important for improving the state-of-the-art in knowledge engineering.

A greater utilisation of knowledge technology will also lead to better understanding of what knowledge is and what role it can play in intelligent behaviour. From such a practical experience, general lessons may be gleaned and our knowledge of our technology and ourselves can be increased. Knowledge systems are partial models of human reasoning and apply this on artefacts that embody this theory. By examining the differences and similarities, new insights can be gained into what knowledge is, how it comes into being, and how it is used. In some way, these insights tell us of who we are and how we ourselves operate. In time this may allow us to create fully intelligent constructs but examining how our current, limited knowledge has already changed what we consider intelligence and what not, shows how important these questions really are.

Some researchers see this role of knowledge systems as models of human reasoning less as part of the field. They distance the field of knowledge engineering from the encompassing field of artificial intelligence, e.g. Stefik (1995). This seems to become

somewhat of a tradition for fields of study originating in AI to dissociate themselves from their roots. This also finds its resonance in one of the definitions of AI: “*Artificial Intelligence is the study of those things humans can, but computers cannot do (yet).*” In other words, if a computer can do it, it is not intelligent behaviour. For example, chess now no longer requires intelligence, just a powerful enough calculator. Therefore, if a difficult problem has a (partial) solution, it is not AI. Still AI can gain valuable knowledge from knowledge engineering research.

Knowledge engineering research can prove to be an excellent example of the needs of the practical engineering aspects and the forward oriented nature of scientific research meeting in the middle. To further knowledge engineering and AI both in terms of applications and science, there is a definite need to translate what is possible now into useful solutions for corporations and individuals today.

### 1.3 Research Goal

The practical objective for this thesis is to address the bottom-line problem. From that point of view, the question is simply: what kind of changes can be introduced to improve and clarify benefits, as well as reduce cost and risk of knowledge system development? The answer to this question can remain simple, limited to treating these symptoms of underlying problems. These solutions will however be limited in effect, if not based on understanding of these problems.

Therefore, to be able to deal with the bottom-line problem, its root causes must be addressed. The problems in knowledge engineering were described as:

- The knowledge acquisition and maintenance problem
- The gap problem
- The brittleness problem
- The complexity problem

These are seen as the main culprits in the respect of cost, risk and lack of benefits, as well as lack of clarity as to the benefits. The scientific objective is therefore to explain these problems and devise solutions to them leading the questions: what causes each of these problems, and what can be done to alleviate or remove these problems?

These problems are not new or unknown. In fact, they have been present in one form or another in knowledge engineering from the beginning. Current approaches for dealing with the problems are inspired by an engineering metaphor. They aim to address the symptoms of these core problems by methodological methods. With added control and decomposition the cost and risk are reigned in and made manageable. While these approaches have been able to reduce the effects of the problems to an extent, the problems remain to pose a threat to knowledge engineering.

The alternative approach to address each of these problems is to solve or alleviate each of them individually. To take but one example, reuse of knowledge is perceived as a possible solution to the knowledge acquisition and maintenance problem. The idea is that removing one of these problems is responsible for a certain part of the bottom-line situation. A successful solution should have a partial effect on the bottom-line. The practical success of these approaches is limited.

A possible explanation for the limited success of both the symptom-directed and problem-directed approaches is that there is a further problem beyond the four root problems. Without that in mind, and without having solutions to the individual

problems acting in concert, current scientific approaches are merely addressing second order symptoms.

This hypothesises a problem that is related to the manner in which knowledge, knowledge systems and knowledge engineering are perceived and acted upon under the engineering metaphor. Behind the roles of people, the approach to management of projects, the views of the products, the process that is employed to tie these together, and the tools that are used to support all of these exists a world of assumptions and preconceptions. This sets the stage for a singular perspective shared by the vast majority of professionals engaged in the research and development aspects of knowledge engineering. This sees a knowledge system as a final product of a grounded, principled process performed by professionals, using tools made for these professionals. The ingrained nature of this metaphor probably even means that any mention of the adverse effects of engineering as the backbone of knowledge system development will most likely be met with instant disbelief.

The direction that this research has examined these practical and scientific problems has consequently been more global than these individual problems or their symptoms. The causes are searched for in current assumptions and approaches stemming from preconceptions present in this engineering metaphor underlying knowledge engineering. The research in this thesis embraces the philosophy that knowledge systems are open-ended, dynamic artefacts that must evolve in response to their environment. The direction that the global solution is looked for is to organise knowledge engineering around *scientific metaphor*, as an insight-based learning process.

The scientific metaphor sees the development of a knowledge system as a continuous process of insight, change, and application leading to further insights. It will serve as the basis for an approach for the development of knowledge systems called *continuous knowledge engineering* (CKE). The goal for this thesis is to examine the ability to apply that approach as a continuous, learning approach to knowledge engineering and determine whether this approach constitutes a solution for the main scientific problems as well as the practical bottom-line problem. The research in this thesis therefore aims to address the problems both from a practical and scientific perspective.

The alternative scientific metaphor that is used will change the view taken on these problems and in directing the search for a solution based on these explanations. The areas for which adaptations are sought for are in:

- People: the roles that people play,
- Project: the view of projects that exists,
- Product: the product that is the result of the activity,
- Process: the kinds of processes used,
- Tools: the tools employed in the development itself.

It is not the intent to excommunicate all approaches and techniques used within the engineering metaphor. A synthesis of these elements with new approaches is used to compose an alternative approach to knowledge engineering in its classic sense. Both aspects of the development method and the evaluation of different features possessed by knowledge system development tooling will be prominent aspects of the research shown in this thesis.

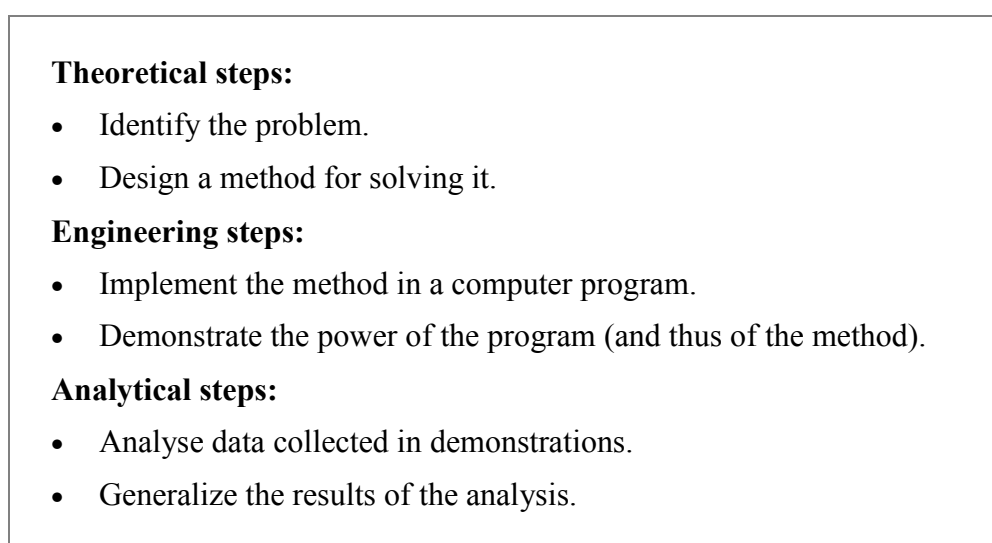


## 1.4 Research Program

This section describes the organisation of the research activities. As an abstraction of the activities performed, it aims to add clarity and structure to the portrayal of the research.

### 1.4.1 An Experimental Scientific Program

Human knowledge grows through the problems experienced by mankind and the attempts to solve them. The explanations for these problems allow understanding of the world surrounding us and guide the search for novel solutions. Realising these solutions and verifying the results gives better tools. Furthermore, it corroborates the explanations that were used. With increased structure and transparency, the knowledge to be extracted can be more easily found and verified.



**Figure 1-1 An Experimental Scientific Program (Buchanon 1989)**

For that purpose, an experimental research program was explicitly used to improve the quality of the research in a practical setting. Beyond merely structuring the task at hand, the adherence to such a program is considered to make the results and the manner in which they were created more transparent and easier to understand. Such an experimental scientific program was proposed by Buchanon (1989), shown in the above Figure 1-1. Combining theory, engineering, and analysis this research attempt to make a solid contribution to the knowledge of knowledge engineering. This program is not unlike that of earlier proposals from Popper, Kuhn and Lakatos, but is a derivative more suited to problem solving.

However, it must also be stressed that the program is a framework for an iterative approach. This is a first step towards an alternative way of developing knowledge systems. No purely theoretical advances are envisioned, but rather an attempt to develop a viable approach to knowledge engineering with a different perspective in mind.

### 1.4.2 Theoretical Steps

The first part of the research will concern itself with the theory and practice of knowledge engineering, to identify the problems more clearly and come to an explanation of those problems. The basic problems that will be examined are the four

root problems: the knowledge acquisition and maintenance problem, the gap problem, the brittleness problem and the complexity problem.

The clarification of the problems and the determination of their causes is performed based on the examination of the assumptions underlying the field of knowledge engineering. In this examination, different aspects of knowledge engineering are discussed. The first subject is the cognitive model that is used to link knowledge systems to general intelligent behaviour. This is followed by a concise account of their history as well as their basic architecture. Another subject that is examined is the methods of knowledge representation that are used. Finally, the way that knowledge systems are developed is considered. For each of these areas, some attention is given to alternative approaches that exist in each of the examined areas.

This identifies an underlying metaphor at work that sees a knowledge system as an artefact that like any other can be designed, implemented, deployed and maintained. It is the assumptions that follow from this metaphor to describe, perform and improve knowledge engineering that are seen as the actual cause of the problems.

Based on the explanation a solution direction is proposed in the form of an alternative metaphor for knowledge engineering that seeks to adapt the current approaches to a more continuous form that is directed as a process of constant insight development and knowledge acquisition. This makes use of many of the features of the alternative views that exist in each of the described areas.

To enable assessment of the quality of the realisation of this solution an evaluation program is devised to allow the analysis of the proposed solution. These criteria seek to reveal whether the proposed effects do indeed take place. Furthermore, they aim to differentiate the current metaphor from the alternate metaphor proposed in this thesis.

### 1.4.3 Engineering Steps

The engineering steps aim to implement and demonstrate the proposed solution. Such an implementation aims to show at the very least an existence proof that a realisation is possible. In addition, it makes it possible to carry out demonstrations of the implementation in experiments, applications and case-studies. Demonstrations can take many forms and can provide different levels of support for claims that are made about the method. The most elementary question to be answered is whether the proposed solution works at all. Another question is the scope of problems for which it presents a workable solution.

The implementation in this case means developing method *and* tools to realise the requirements set out in the theoretical phase. There are many different ways to implement these requirements and create an implementation. In the course of this thesis, the implementation of two knowledge modelling tools will be shown. Each is based on a different philosophy, and thereby implements a different subset of the total requirements.

The first tool, the *Knowledge Base Editor* (KBE), is simplistic in nature, focussing on visual knowledge representations. The second one named *Intelligent Objects* (IO) expands the accomplishments of the first tool and introduces some more advanced concepts, such as object orientation. This allows us to answer questions on which features support which part of the continuous knowledge engineering approach.

Many researchers do not report on early implementations that failed, or previous versions of developed tool and what is left unrecorded is doomed to be repeated.

Comparing such pairs of tools is made extra difficult because of “our present inability to sort out essential from non-essential attributes” (Buchanon 1989). It is tedious to collect data and it is not obvious what is worth measuring.

To be able to compare the two tools the demonstrations also needs to show which features of these tools beget the benefits in combination with different realisations of the method get what results and come to some general conclusions about this. Pairs of implementations of the same program of requirements can provide interesting data for comparisons.

Data collection can be carried out by active experimentation or by passive observation, or by variations on either. The experimental nature of knowledge engineering is made more difficult by it being near impossible to perform controlled experiments (cf. (Lethbridge 1998, 1994)). It is hard to create similar experiments as the field of application and the skill levels of the participants is an important factor in the end result. The results of such controlled experiments therefore must be viewed with a critical eye. To eliminate these influences one would need great numbers of participants and a great number of projects to get the confidence levels needed. This is in most cases not a realistic option.

Because of this, but in majority due to the practical setting of the research it was not possible to organise case studies that were specifically geared towards the evaluation of the notions in this thesis. As this was anticipated, an approach was chosen where a number of candidate applications could be used to demonstrate different aspects of the tools and the approach. The number of projects so performed and analysed is therefore greater than would have been the case when the selection process was more directed. Five distinct knowledge systems that were developed at TNO realised with the tools are described in this thesis. These are examined as demonstrations of both tools and method. Each shows a different configuration of methods used incorporating different aspects of the continuous knowledge engineering approach, and different ways of employing the tools in the realisation of the knowledge systems. These case studies were examined through passive observation, which was the only option in this environment.

#### 1.4.4 Analytical Steps

The analytical steps need to evaluate the cases and attempt to generalize the results of the evaluation. The result of the analytical steps should be some answers to the questions that were posed in this thesis, but could also be new problems for the subsequence theoretical or engineering steps. This is the goal for this research: to complete at least one such cycle and come back to the origin. The aim is to take the first steps toward continuous knowledge engineering.

Therefore, one of the questions to which an answer is sought is whether the notions proposed in this thesis can work at all, in other words, can its base principles be realized. By examining the two tools, we can gain more insight in the factors that determine success and get more data as to the ability to realise each of these principles. The second question is whether it is a viable approach, that can be used to develop knowledge systems practically. The applications made using the two tools give us more insight into the practical ability to create knowledge systems. Finally, what is the scope of problems on which it can be used and what aspects of a problem determine what the benefits of employing continuous knowledge engineering are. Are

there limitations to the benefits and what factors determine these limitations, in other words, what benefits are realised in what circumstances.

### 1.4.5 On Science, Practice and Pragmatism

Considering the steps that were discussed just now, commercially speaking a successful implementation of the demonstrations in terms of benefits, cost and risk would suffice. Scientifically speaking one would like to complete all six steps to make advances in the underlying understanding. Given the practical context of the research then the latter steps are the hardest to execute. Partially this is due to the difficulty with which knowledge systems can be developed under controlled conditions, mentioned earlier. However, much is also due to the lack of control that can be exerted when projects' main goal is a practical one.

The projects used as case studies were selected on the basis of pragmatism. This means that they were used when it was both economically and practically possible, more than whether the project posed important scientific questions. The cases are as inclusive as possible and are quite representative of the work performed at the Knowledge Based Systems Department.

While the practical aspects can be seen as a weakness, it is also perceived as a strength. In keeping with the program described by Rodney Brooks (1991, pg. 139): "We must incrementally build up the capabilities of intelligent systems, having complete systems at each step of the way and thus automatically ensure that the pieces and their interfaces are valid. At each step we should build complete intelligent systems that we let loose in the real world with real sensing and real action. Anything less provides a candidate with which we can delude ourselves."

It is the contention of this thesis that the core of this program for researching and developing intelligent systems extends to developing knowledge systems and applies just as well to research into knowledge system development. If it is important to position the results of research into practice, why not do so continually. And, if the beginning is so important, then why not begin all the time.

## 1.5 Thesis Overview

This thesis describes research into the knowledge engineering and knowledge systems development. The chapters follow the description of the research program presented earlier:

**Chapter 2** reviews knowledge system background, current development methods and knowledge representation techniques. Furthermore, it discusses the main problems in knowledge engineering.

**Chapter 3** describes an alternative approach to knowledge engineering, called continuous knowledge engineering, and formulates a series of requirements for tools to support the approach. In addition, an evaluation program is defined.

**Chapter 4** presents the KBE as a knowledge system development tool, aimed to support the CKE approach by providing an easy, visual knowledge representation format and support for knowledge system development

**Chapter 5** describes IO as an object oriented modelling knowledge system development tool. It incorporates features of the KBE, but is more advanced and aims to provide a more complete, vivid knowledge modelling language.

**Chapter 6** reports on a number of real life case studies. This provides a demonstration of the tools and different elements of the continuous knowledge engineering approach.

**Chapter 7** collates and analyses the results of the case studies, by following an evaluation program. Consequently, a number of conclusions are reached concerning the effects of the method and the tools.

**Chapter 8** summarises the main conclusions of this thesis and points out some directions for future research. In addition, the contributions of this thesis are summed up.

Parts of the research described in this thesis have previously been published in (Schilstra & Spronck 2000, Spronck & Schilstra 2000, Aarts et al. 1999).

# Chapter 2

## Problems in Knowledge Engineering

---

*The beginning of knowledge is the  
discovery of something we do not understand.*

– Frank Herbert

The term *engineering* in knowledge engineering was included to emphasize that the development of knowledge systems should follow established scientific method for the creation of complex artefacts. The treatment of different aspects of knowledge engineering shows an ingrained reliance on an engineering metaphor for the development of knowledge systems. This metaphor in its different aspects supplies the basic assumptions behind the current approaches. It gives rise to the current explanations of problems that are experienced and provides the inspiration for the attempts at solving these problems. This chapter examines the state-of-the-art in knowledge engineering and identifies its main problems from a practical and scientific point of view as being the result of this singular reliance on an engineering perspective of knowledge systems.

The first section discusses two cognitive models that each gives a foundation of knowledge systems as intelligent systems. In the second section, the knowledge system and its fundamental components are defined, going on to provide some examples of knowledge systems and an overview of their advantages and disadvantages. The third section discusses the roles of knowledge representation and assesses a number of knowledge representation techniques. In the fourth section the approaches to developing a knowledge system are discussed, the rapid prototyping approaches ‘of old’, and the current methodological approaches. The fifth section then collates the main problems that face knowledge engineering and discusses their background and causes. Finally, in the sixth section, some conclusions are reached and a global solution to these problems is proposed.

### 2.1 Symbolic vs. Situated Cognition

AI is concerned with the complex behaviour of agents in non-trivial environments. An *agent* is an autonomous entity able to sense and act, and capable of making changes in the environment that are in some way beneficial to the agent. This definition covers natural agents such as humans and animals, and unnatural agents such as machinery and computer viruses. The study of AI is concentrated on the different cognitive mechanisms are considered as responsible for mapping senses onto actions. Different models of cognition are explored to explain for human and other intelligence. These models are also used as the basis of the development of intelligent technologies.

The symbolic and situated models of cognition constitute two of the most influential ones. The symbolic model proposes some form of symbol processing as the basis of rational behaviour. The situated model perceives a more low-level mapping of inputs and outputs, determined by a tuning of the agent to the environment. The importance of the position taken on the mechanism underlying the operation of any intelligent system or agent is profound. It determines the basic assumptions of any approach to

create an intelligent system, and thereby structures the methods of realising such system, in a sound and practical way.

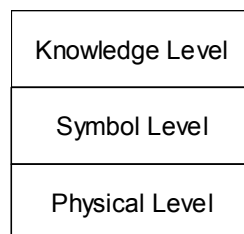
### 2.1.1 Symbolic Cognition

The symbolic model of cognition sees an explicit representation of knowledge as the basis of the mechanisms for choosing the best action. For this it uses the principle of rationality (Newell 1982): “If an agent has knowledge that one of its actions will lead to one of its goals, then the agent will select that action”. This is also known as the Maximum Rationality Hypothesis. Simply stated, the symbolic model of cognition is based on a view of the mechanism as a rational one, with rational behaviour as the ability to use sensory information and knowledge to produce actions beneficial to an agent in a goal-directed way.

In this view, knowledge is the medium by which agents can determine their next action. It feeds searching for the best action by telling the mechanism which action is best, rather than having to search for an action. This knowledge principle can be likened to compiled thought as it negates the need to think (search) about such a choice, when the best choice is ‘known’, also known as ‘knowledge is power’.

Stefik defines knowledge as “the codified experience of an agent” (Stefik 1995). This statement implicitly states that knowledge is acquired through learning. This emphasises the importance of the activity that involves the knowledge and leads to new knowledge. In addition, the term codified indicates that there is some medium holding the knowledge in some form or representation. On the other hand, it does not say that something will be using the knowledge. It is knowledge by virtue of its encoding and its basis in experience.

#### Knowledge Level



**Figure 2-1 Levels of Description of Intelligent Systems**

To clarify the distinction between rational behaviour and the representational issues Newell proposed a new and distinct computer systems level which he called the *knowledge level* (Newell 1982). An agent is said to be a knowledge level system when it rationally brings to bear all its knowledge onto every problem it attempts to solve. Thus, knowledge is the medium of transaction at the knowledge level and the behavioural law is the principle of maximum rationality.

Humans and other systems embedded in the real physical world can only attain bounded rationality. Bounded rationality means as rational as possible, taking into account constraints such as limitations in time and resources, but also in the inherent uncertainty and incompleteness of the available sensory information. There are therefore differences between a pure knowledge level system and a human that give telltale signs of the underlying human cognitive machine.

At the knowledge level, the behaviour of a rationally acting agent can be determined by examining the knowledge that an agent has, without regard to the particular representation of that knowledge or the implementation issues of a mechanism that employs this knowledge. In other words, an intentional agent's "behaviour may be predicted by the method of attributing beliefs, designs and rational acumen." (Dennett 1987).

### Symbol Level

The symbol level provides an implementation for a knowledge level system. At the knowledge level, there is no information or assertion about what kind of symbol system must support it. The implicit assumption is that a knowledge level system can be realised by many different kinds of symbol system.

A *symbol* is a representation of an entity or concept in the environment of the agent. A constellation of symbols, or symbol patterns can be combined to create a model of the world. A *model* is seen in this view as reconstruction of a reality from a specific perspective that contains the relevant issues for that purpose. It is akin to a scientific theory or mathematical model. By manipulating the symbols in the model in accordance with laid out rules, additional information can be derived of the model, and thereby of the environment, without the need to sense. This makes it possible for a symbol system to predict the future if a certain action would be undertaken or to look back into the past. As long as current information and the rules used are correct, this should hold true.

This resounds in what is undoubtedly the hallmark of good old-fashioned AI (GOF AI), the *Physical Symbol System Hypothesis* formulated by Newell and Simon (1976). This hypothesis states that: 'A physical symbol system has the sufficient and necessary requirements for rational behaviour.' It asserts that rational behaviour is achieved through:

1. symbol patterns to represent significant parts of the problem domain,
2. operations on these patterns to generate potential solutions, and
3. search to select a solution from the possibilities.

In other words, this hypothesis states that some form of physical symbol manipulation can be discerned in all rational systems, and if it cannot be discerned then the system is not a rational system. The physical symbol system hypothesis restates the earlier relation between knowledge and symbol level in a more strict way.

The physical symbol system hypothesis also makes clear that any system that can implement the right symbol patterns is capable of rational behaviour. Computers are capable of implementing any effectively described symbolic process. The transistors and electronics circuits act as the physical level, implementing the symbol level within a computer. The development of intelligent systems based on this hypothesis involves a knowledge level model translated into a symbol level model on a computer. Therefore, properly programmed computers are capable of achieving intelligence. This hypothesis is a theory of cognition that explains the basic mechanism of intelligent behaviour and which makes an intimate link between cognition and a Turing machine as a symbol pattern manipulation engine. No small wonder that this hypothesis has been the mould for many AI researcher in search for intelligence, and is likened to absolute truth by some.



In humans, neurons are apparently capable of acting as the physical level for a symbol level system, in a way as yet unknown to us. The knowledge level of the human is the ability to work in rational ways, however bounded. This view emphasises human logical and analytical capabilities. In fact, it the symbolic theory of cognition ‘models’ humans almost exclusively as rational beings, marking other attributes as implementation issues or irrelevant.

### Role in Knowledge Systems

This philosophy of intelligence and rational behaviour underlies the majority of the research into knowledge systems and many other types of intelligent systems. These systems embody the symbolic theory of human cognition. Many successful and practical systems have been created in this way, which is considered evidence of the veracity of the theory.

Knowledge in the systems originates in experience and is either “acquired from external sources of expertise or learnt through problem solving” (Motta 1999). The question here is whether the first form is not seen to imply learning. This perhaps is what is sometimes vexing about this symbolic theory of cognition: it says nothing whatsoever about the way in which the symbols get into the system, or how these symbol patterns may change over time. There seems to be no need for learning from experience, as alluded to by Stefik. The assumption is that the system performs intelligently in an *objective, time-independent* way. If it is intelligent today implies it will be intelligent tomorrow.

There is also a tendency to view these systems from an almost mathematical viewpoint, where they operate based on a form of logic and can be proved to be sound, complete and tractable (Nilsson 1998). This is considered an essential and useful aspect of knowledge systems. This is strengthened by the fact that the knowledge in the majority of knowledge systems concerns mental tasks, such as law, medicine, engineering, etc. The work in many of these fields can be seen as either synthesis or analysis (Clancey 1985). These are overall deductive, model-based tasks. In these tasks, the link to reality is relatively tentative, with abstract sensory information (e.g. light-on) and abstract actions (turn-light-off). This fits well with the chosen application fields, and allows us to forget these peripheral issues such as real sensors and actuators. The assumption is that these could be added later.

#### 2.1.2 Situated Cognition



Initial state :  $\text{On}(C, A) \wedge \text{Clear}(C) \wedge \text{Clear}(B) \wedge \text{On}(B, \text{Table}) \wedge \text{On}(A, \text{Table})$

Goal state :  $\text{On}(A, B) \wedge \text{On}(B, C)$

**Figure 2-2 Sussman's Anomaly**

Many AI approaches have been tested on so-called ‘toy domains’. A *toy domain* is a simple environment for which certain basic problems could be formulated to be solved by a system. An example of a toy domain is the Blocks World, stacking block on top of one another. Sussman's Anomaly characterises a specific problem of goal interaction (Sussman 1975). To reach the end-goal, the action that immediately

realises one of the goals  $On(B, C)$  must not be chosen (see Figure 2-2). The running assumption was that if one could create a system that could solve an abstract version of some problem in an artificial world, that solution would translate and scale to a real world solution. Of course, as can now be safely said with perfect hindsight, the story is more complicated than that (Minsky 1988). The term *toy domain* since has been used to deride a certain application area as too simple to contain a relation to the real world. For example, scale Sussman's Anomaly to a robot driving a forklift truck and see what the real problems then would be.

From the problems with translating solutions from toy domains to the real-world originate AI approaches employing the situated theory of cognition. Situated cognition emphasizes the situation or *context* of the agent as determining much of the appropriateness of behaviour and determining the beneficial nature of action, where the symbolic theory of cognition attempts to abstract away from. Context refers to the context and environment, the history, and all other physical aspects of the perception, mental state, and behaviour of an agent. This is analogous to the view of Kuhn (1962), who argues that scientists' experimental observations presuppose their tacit ability to use their experimental equipment and to apply their frameworks of hypotheses and theory. The philosophical foundations of situated cognition theory are said to be laid out by Heidegger (1927), the first to point out the role of tacit pre-understanding and to elaborate its implications.

Situated cognition redefines knowledge as the ability to behave interactively with the environment. Within this framework, knowledge is no longer seen as the sum of all the concepts and relationships within the head. The knowledge can be taken out of the context, but the context cannot be taken out of the knowledge. To counter some of the naïve interpretations, it is not about the environment enabling the agent to act intelligent nor is it about the world containing knowledge outside of the agent like words on a sign. It is also not about a large body of common-sense knowledge that must be present and shared.

Situated cognition offers a different perspective on cognition. It sees the mechanism that any agent uses to determine its actions as specialised in the shape and form of the agent and in the specifics of the environment that it inhabits. This *context* includes all the real senses and actuators involved, and even the activations of the brain that could be only in part due to the perception itself, for example memory and emotive states. This takes away from the mental aspects of cognition, as the 'implementation' details of the activity are an integral part of the experience. In turn, this means that the peripheral issues cannot be forgotten but are essential.

Every human thought and action is adapted to the environment in which it is applied: "[W]hat people *perceive*, how they *conceive of their activity*, and what they *physically do* develop together" (Clancey 1999). Situated cognition sees knowledge as linked to the context in which it is acquired and applied: "Human knowledge development as a means of coordinating activity *within activity itself*." Even using knowledge, or thinking about knowledge or modelling knowledge changes the knowledge. The very act of creating and using a knowledge model leads people to reinterpret what they mean and to adapt the ideas to new situations. In some sense "every action is automatically an inductive, adjusted process" (Clancey 1987, pg. 238).

The neural structures and processes that coordinate perception and action are created during activity, not retrieved. Neural organizations arise in the course of activity, they are always new, and they are not retrieved from storage. According to situated

cognition, concepts, procedures, etc. are not neutralized, abstracted, de-contextualised pieces of symbolic information, retained in ones head and retrieved when needed.

For a knowledge system, a consequence is that knowledge loses its eternal quality. The knowledge that is developed and used is intrinsically woven with the social, cultural and material context in which it is employed and learnt. As the scientist finds the world changed by starting to measure, so the knowledge engineer can find the knowledge changes because a knowledge system is developed. This entails that knowledge is by nature and origin fluid and experiential, and that it can retain its quality only by continuous adaptation through application in the environment.

This resounds in the *Situated Cognition Premise* that “[H]uman cognition cannot be accurately modelled by context independent assertions”. Situated cognition (SC) claims that knowledge is context-dependent and that symbolic descriptions elicited prior to direct experience are less important than functional units developed via direct experience with the current problem. Proponents of situated cognition argue widely different views that can be roughly divided in two camps according to (Menzies 1998): Weak Situated Cognition and Strong Situated Cognition.

*Strong Situated Cognition* states that “since the influence of the environment is so great, we must use pure reactive systems that interact directly with the environment without reflecting over some symbolic descriptions”. This strong situated cognition position would seem to state that the relationship to the environment or context is so strong as to make it impossible or very difficult to use any symbolic processes. Explicit symbolic thought processes in themselves constitute a reaction to an internal subsymbolic context. From this stance come approaches to the development of intelligent systems grounded in their environment that do not employ any representation or symbolic reasoning. This excludes knowledge systems as resembling their current form.

This form of situated cognition therefore states that symbolic approaches cannot lead to cognitive models, and denies the physical symbol hypothesis. Proponents of the strong approach include Rodney Brooks (1991a, 1991b) and Drew McDermott (1987). A similar attitude is found in the critique of AI made by Dreyfuss (1979) and Searle (1980). They contend that a computer system can never be intelligent *like a human*, because they do not share the same context.

A compromise position is formulated as *Weak Situated Cognition*: “[t]he inferences of a symbolic model interacting with its environment are heavily constrained/controlled/changed by the inputs from that environment. That is, using knowledge in a particular context will significantly change that knowledge.”

This leads to a view of knowledge as a connected representation of solving problems, as they are experienced in a certain context and forms a consistent whole within this context. However, the knowledge changes when it is examined or applied. In other words, knowledge modelling is limited and knowledge models are inaccurate renditions of a temporary state of mind, which fluctuates even more when examined. Evidence of this would be that knowledge systems on static domains would still be seen to change, as they have been reported to do (Menzies 1998, Compton et al. 1992, Compton & Jansen 1990).

The situated premise states that human knowledge must take into account the context in which it originates. It says that all knowledge is context-specific, i.e. a rule is as good as the situations it has been applied to. The rule only contains knowledge as far

as no exceptions to it have been found, and (unexpressed) assumptions have not been falsified. It therefore does not propose that maintenance is an extra problem to be dealt with, but a fundamental issue. It is “an instance of the fundamentally fluid nature of knowledge” (Compton & Jansen 1990). It shows the direct influence of activity on the development of knowledge and how using knowledge changes knowledge. It illustrates that modelling knowledge requires constant synchronisation with experiences from applications.

Marvin Minsky has in a way defended the weak situated premise. He sees *symbolness* as a matter of degree. This can be taken to mean that symbols are a useful abstraction, and that neither is completely wrong or right. Different tools may be required on different problems. and where one sees a forest, others see trees. In many cases this means that for any connectionist system, an equivalent symbol system can be constructed which is a satisfactory model of an entity’s behaviour (Minsky 1990, 1988). This satisfies the physical symbol system hypothesis as an analytical perspective for an outside observer. Furthermore, it leaves the door open for connectionist, situated and other representation-less or *subsymbiotic* approaches.

### Role in Knowledge Systems

This philosophy for the mechanism of the agent is not used in many systems beyond those that employ inductive techniques and automated learning. A good example of these systems are Brooks’ *subsumption* hierarchies. In these systems a behaviour, say avoidance, is hardwired into an agent. After trying it out in a specific environment, other behaviours are grafted on top of the existing ones to modify and extend the existing behaviour. One of these behaviours cannot be taken from the agent and simply placed in another body, they are tailored to the one agent. Because these behaviours can also change the environment, certainly when populated by several such agents, the emergence of benefits from specific behaviour can also be due to the agent emergence and co-evolution with their environment.

Situated cognition is a challenge to current state-of-the-art knowledge system development strategies based on the physical symbol hypothesis. The weak situated cognition position indicates an alternate approaches for knowledge system development. Based on its assumptions, design and knowledge modelling becomes less important, because the situation changes as a rule. Grounded activity must lead to new knowledge and adaptation in existing knowledge. The context that the knowledge system inhabits leads to specialised knowledge models specific to that situation.

For example, a knowledge engineering methodology that acknowledges situated cognition must offer details about creating and changing a knowledge base. A review of the knowledge engineering literature suggests that most of the effort is in knowledge analysis and not knowledge maintenance (exceptions: XCON (Bachant & McDermott 1984) and Garvan ES-1 (Compton & Jansen 1990)). In the remainder of this chapter, the state-of-the-art in knowledge engineering is examined with this scientific controversy in mind and with a view towards making knowledge engineering a discipline directed towards learning and change.

### 2.1.3 Conclusions

The two different cognitive models presented above roughly conform to two attitudes in computer science. The symbolic approach is akin to ‘inventionism’ that argues that all information is interrelated and that any new information must be reconciled with

the existing: information has eternal validity. The situated approach has ‘adaptivistic’ tendencies and shares the attitude of full interrelation, but the latter recognizes that there is a growth process: new information may cause old information to be replaced (Jongeneel 1996).

Without siding with either camp, it is clear that considering the development of knowledge in humans, to assume that knowledge is static in knowledge systems is an indefensible position. When accepting the weak situated cognition, knowledge engineering must move away from design-focused approaches towards maintenance-focused approaches. It implicitly states that many real world domains may only be understood through exploratory methodologies. It is possible to adopt weak situated cognition and retain the physical symbol system hypothesis, but only if it can be demonstrated that the knowledge engineering approach can manage changes to our descriptions of knowledge with ease.

The controversy makes clear that the assumptions derived from the symbolic model of cognition should at least be questioned, and productive alternatives should be examined. The symbolic cognitive model is the foundation of many of the features of knowledge systems and the process of their development. This discussion has shown there to be room for other perspectives.

## 2.2 What is a Knowledge System?

This section examines in more detail what a knowledge system is. In part to provide some consistent terminology and definitions for later discussions but also to show the way that a knowledge system already incorporates a basic strategy to deal with changing knowledge. Furthermore, some examples of historical systems are provided and the advantages and disadvantages of knowledge systems are discussed.

### 2.2.1 Definition

A knowledge system was defined earlier on page 1 as “a software system that makes a body of knowledge operational to solve problems using a separate representation of knowledge that can be employed dynamically”. As this definition differs in some respects to other definitions, some explanation is given here.

To split the definition into its constituent parts, a knowledge system:

- ... is a software system (class),
- ... makes a body of knowledge operational to solve problems (function),
- ... has a separate representation of knowledge (structure),
- ... uses represented knowledge dynamically (process).

The first part of the definition states that a knowledge system is a software system. It is a set of software components developed to perform a certain task or function, using a computer of some kind. This part of the definition places knowledge systems within the category of software systems.

The second part of the definition says the knowledge system makes a body of knowledge to solve problems operational. In this context, *operational* means that it makes the knowledge able to be put it to practice immediately. No effort to absorb or integrate the knowledge on the part of the users is expected. Furthermore, the body of

knowledge is a consistent whole, specific to one or more related domains or tasks. It is not a collection of isolated elements. In this way, the knowledge is embodied by the system in such a way that it can be employed immediately by the system to solve a problem. This is the function of a knowledge system.

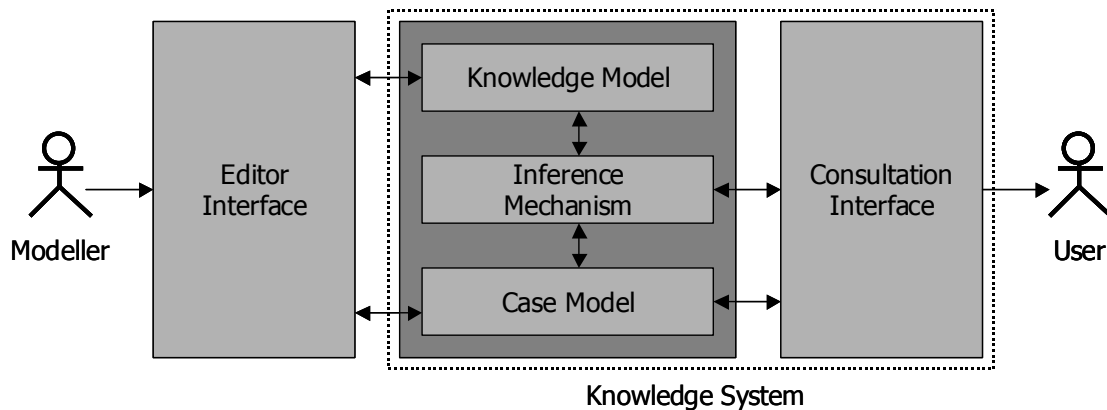
In the third part of the definition a structural difference is mentioned. As a software system the fact the knowledge is separated from the system proper sets knowledge systems apart. This does not mean other systems are not based on knowledge or have knowledge in them, but these have the knowledge enmeshed in different parts of the system. According to *software = procedures + data*, this would mean that non-knowledge systems have the knowledge in the procedures rather than the data. For knowledge systems the equation is different, *knowledge system = inference engine + knowledge model + data*. This describes the structure of a knowledge system.

Finally, the fourth part mentions a more important aspect of the separation of the knowledge, namely the ability employ the knowledge dynamically. The knowledge is represented as separate parts that can be used in many different combinations. In most cases, these parts have triggers that assert whether the knowledge applies in the current situation. This means that the knowledge can be employed specific to the situation. The use of knowledge emerges from the description of the problem situations, which creates the ability to work with a great deal of flexibility. It allows a knowledge system to approach a broad category of problems, without each of the problems being foreseen or enumerated in advance. This is perhaps the most important aspect of the knowledge system's ability to solve problems. While normal software is equipped with procedures and other deterministic means of problem solving, a knowledge system adjusts itself to the problem situation based on the knowledge it contains. In other words, "expert systems have capabilities and potential beyond those of conventional programs" (Crofts et al. 1989).

What the definition does not mention is the human aspect of a knowledge system nor does the definition use any anthropomorphic words attached to the processing of the knowledge. A knowledge system can be defined as "a computer system that emulates the decision-making ability of a human" (Giarratano & Riley 1989) to "... solve problems that are difficult enough to require significant human expertise for their solution" (Feigenbaum 1982). The human component is missing in the definition presented in this thesis not because it is irrelevant or otherwise negligible. The origin of the knowledge in an knowledge system is in the overall majority directly or indirectly human. Exceptions to this are few and far between, but advances in automated learning begin to show what arguably is machine knowledge. This however is not of interest to the question whether or not a system is a the knowledge system. If the knowledge derive from a non-human source, it would still be a knowledge system. The ability to understand the knowledge as it is represented is important (e.g. the Human Frame as suggested by van den Herik (1988)).

### 2.2.2 Architecture

In this section the architecture of the knowledge system is described. As the architecture is similar yet different for many researchers, this description attempts to sketch the default architecture while introducing some consistency in terminology.



**Figure 2-3 Knowledge System Architecture**

The figure above shows the basic *knowledge system architecture*. The knowledge system is divided into a *knowledge component* (the dark grey section) and its visualisation through the consultation interface on the right. The knowledge component is an abstract software component that performs computations but not necessarily has a visualisation. It holds the knowledge model, case model and inference mechanism.

The consultation visualisation provides the user with the functions of the knowledge system. On the left side the modeller using an editor interface makes changes to the knowledge component. This is the tooling side, in this thesis termed a knowledge system development environment. This may contain tools to develop consultation interfaces as well. In a knowledge system there often is a distinction between the representation and visualisation of the knowledge. The visualisation through the consultation interface can also allow another software system to employing the knowledge one of its components.

The most notable feature of the architecture of a knowledge system is the separation between the domain-independent inference mechanism and domain-dependent knowledge and case model. This is one of its defining features (Motta 1999). This enables the system to be changed by making adjustments in the reasoning knowledge without having to modify other parts of the system. Any system that uses the knowledge base component, will therefore be guarded from modifications in the knowledge.

### Knowledge Model

The *knowledge model* consists of individual quanta of knowledge that can be employed individually, in combination with others. It is separate from the rest of the systems, among other reasons because it is the part that changes most over time. The knowledge model contains representations of knowledge, used to assert new facts from the existing facts.

```

if temperature > 20 then heat := off
if temperature < 10 then heat := on

```

**Figure 2-4 Heating System Knowledge Model**

In a standard knowledge system, it contains production rules. An example is displayed above. It states that if the temperature in the room is higher than a threshold value, the

heating must be turned off. Conversely, if the temperature falls under a minimum threshold it must be turned back on. The model, implicitly contained in these rules, consists of two facts: *temperature* and *heat*. The total of the rules uses parts of the domain model, to define interconnections of the system.

### Inference Engine

The *inference engine* applies the reasoning knowledge to the facts to assert new facts, towards the determination of the end-goal. The rule in the knowledge base can be used in two main ways. When the temperature is higher than the threshold value, the heating can be turned on. Alternatively, to answer the question whether the heater must be turned on, a check whether the temperature is actually higher than the threshold value is required. The first approach is called forward chaining, based on the information that is already present we fire those rules whose condition is satisfied.

```

function Infer( Goal ): Value
if goal is asserted then
    result := goal
else
begin
    ruleset := find rules with goal in consequents;
    for all rules in ruleset do
        if all antecedents is true then
            fire rule; //asserts goal
    end

```

**Figure 2-5 Backward Chaining Inference**

The second approach is called backward chaining, where a goal is attempted to be proven, by checking whether rules exist that assert the goal, whose condition is satisfied. The latter is an approach used when a user must answer questions on the facts used in the conditions.

### Case Model

The *case model* contains facts that are asserted during a consultation to describe the current case or problem situation (Steels, 1992). A *consultation* is a single usage of the system to solve one problem situation.

```

temperature      :   Integer
heat             :   Boolean

```

**Figure 2-6 Heating System Case Model**

In many early systems, there is no explicitly modelled case model, but merely a storage where facts are asserted and retracted at run-time. This is why the discussions of the architecture does not feature as a separate part of the system, it is integrated into the inference mechanism. In these systems there is an implicit case model contained in the knowledge system, by virtue of the facts mentioned in, for example, the production rules in the reasoning knowledge (compare the knowledge model in Figure 2-4).



Many current systems use a more explicit model that constitutes part of the modelled knowledge. Knowledge of important concepts, their attributes and interrelationships between these concepts can hold important information. These models can be of different complexity. In some systems, they are as simple as a list of attribute value pairs as in Figure 2-6. Others have highly structured or object-oriented models organising concepts and their relationships. In this way, it contains knowledge about concepts, attributes and relationships in the domain. As the case model becomes more explicit and structured as well as more interlinked with the knowledge model proper, researchers tend to include the case model into the knowledge model. In the remainder of this thesis the term knowledge model will be taken to include the case model, and serve as synonymous for all the explicitly modelled aspects of a domain.

### **An architecture for change**

The lines drawn in this design are not always as clear as all this and different authors use different terms. However, the different responsibilities of each of the parts are discernable as a separation of concerns. The break-up of a knowledge system into different parts shows a design meant to introduce watersheds to deal with change by separating the mutable parts. The first division is to separate out the knowledge per domain into a knowledge model, while the inference engine remains fixed. The second division is to allow changes in the presentation of the system without changing the knowledge, by splitting the knowledge system into a knowledge component and a visualisation. This also makes it possible to present knowledge in a different way to different users. This improves the ability to bridge possible gaps between the user's background knowledge and the level of expertise expected in the knowledge model.

### **2.2.3 A Historical Perspective**

Over time, an evolution of systems can be seen that all conform roughly to the description of knowledge systems given above. Some important differences between them do exist however.

#### **Early Systems**

The beginning of knowledge systems can be traced back to the beginnings of AI itself. In a way, the General Problem Solver (GPS) system was one of the first to treat logic as knowledge that could be employed to state things about the world, namely state facts and prove or disprove new facts (Newell & Simon 1963). It was based on the assumption that much of problems solving could be attacked from logic and automated proof from first principles. It therefore implicitly emphasized a deep knowledge approach, but by its knowledge representation's limitation in representing different levels of knowledge, was crippled in approaching it in that way.

One of the first systems "that employed large amounts of domain specific knowledge" is DENDRAL (Feigenbaum, et al. 1971, Lindsay et al. 1980) originating in the Stanford Heuristic Programming Project of 1965. DENDRAL (DENDRitic ALgorithm), is a system for predicting the structure of organic molecules given their chemical formulas and mass spectrogram analyses. It reportedly "surpasses all humans at its task and, as a consequence, caused a redefinition of the roles of humans and machines in chemical research" (Turban 1990). Based on the industry standard of the production system it encodes its knowledge as production rules.

Another system also quoted as one of the first expert systems, when defined as a system that endeavours to act at a level of competence of an expert, is MYCIN (Shortliffe 1976). MYCIN is a medical diagnosis program. It diagnoses diseases such as meningitis and makes recommendations of treatment with antibiotics, based on examinations of symptoms evidence, test results and knowledge of treatments. By combining the knowledge of several experts in the field, it attained a level of competence similar to that of experienced physicians (Clancey 1985).

Based on the MYCIN system EMYCIN was extracted from the code to form an empty shell. The word *shell* denotes that the knowledge is meant to 'fill' the empty shell. The shell contains the inference engine and an empty knowledge model. Many other knowledge systems have been realized using EMYCIN and it was the inspiration for significant number of development tools. Many of its architectural and conceptual contributions can still be recognized in knowledge systems today.

### First Generation

The early systems and shells extended into what are now known as *first generation knowledge systems*. First generation systems employ shallow heuristic rules, using a clear distinction between the domain knowledge and a domain-independent reasoning procedure. For example, "If both rapid heart beat and fine finger tremor are observed in a patient, suspect the possibility of hypothyroidism with confidence 0.7 and proceed to ask for lab tests" (Shapiro 1987). This rule does not build on a complete causal model or different levels of reasoning, but instead uses a 'rule of thumb' or heuristic. The heuristic is sufficient to make useful conclusions, in about the same manner a doctor would. The domain independent inference mechanism merely needs to apply such rules mechanically. It can remain oblivious to the meaning of the elements mentioned in the rule, such as differences between symptoms and hypotheses.

The initial *transfer* assumption was that the experts could be asked to provide their heuristics which would then be transferred directly into the knowledge system. This transfer process assumed that the knowledge was already there and only needed to be collected. It turned out that the experts did not possess the rules such as those above in that form in their head, nor did they find it easy to formulate or formalize their knowledge as such. It became clear for the first time that knowledge acquisition was a difficult task.

Furthermore, first generation systems ran into trouble because of their reliance on heuristics. The rules failed or produced inconsistent results when drastically new or unanticipated situations are encountered, leading to a knowledge system's brittleness. In addition, they cannot explain their reasoning very well, but can only show a trace of their assertions. This arguably made them too weak to solve large and extended problems. First generation systems also suffered from problems in scaling from initial prototypes to actual systems, as they were hard to maintain.

### Second Generation

From the problems experienced by the first generation, a second generation arose. *Second generation knowledge systems* main distinguishing feature is the use of deeper knowledge models that utilise rich representations of the domain. They employ a model of their problem domain that is declarative as opposed to a procedural form

(van de Velde 1986)<sup>1</sup>. This means that the knowledge is approached as distinct from its use. The knowledge representation also distinguished between different types of knowledge, and makes the problem solving approach explicit.

Consider a system like MYCIN for medical diagnosis. A first generation system only has a collection of undifferentiated heuristic associations that it applies to arise at a conclusion. A second generation system distinguishes between problems, symptoms and hypotheses. Its inference procedure is specialised for the task type, for example a cover-and-test approach, explicitly using the symptoms to determine which of the hypotheses are plausible, by testing each of them. Within a second-generation system the problem solving approach is explicit and part of the description of the knowledge. In first generation is has been weaved through the rules, by the influencing of the inference process by tweaks such as clause order, usage of resolution mechanisms, etc. Making the underlying model explicit enables both understanding of the model, as well as allowing the model to be scrutinised for correctness and completeness. This generation of knowledge systems led to the explicit modelling of knowledge used to construct comprehensible, controllable and maintainable systems.

Attempting to by-pass the difficulties of knowledge acquisition, the second generation systems approach it as a modelling process. The modelling assumption asserts that the expert can not access all of his knowledge directly as it is hidden in his skills. This 'hidden' knowledge has to be built up and structured during the knowledge acquisition phase (Studer, Benjamins & Fensel 1998).

Because a second generation knowledge system has a deep model of knowledge it can also provide a better explanation of its reasoning. It can explain that its current question is required to prove or disprove a specific hypothesis, rather than print the current rule that is being processed. Furthermore, because it employs a deeper model, it can fall back on more general rules, whenever a situation occurs that is not within its scope, degrading more gracefully than failing completely.

In current systems, a combination of shallow and deep knowledge is seen. According to Clancey (1985) the distinction between deep and shallow knowledge is unconvincing. He instead states that examination of all knowledge as a form of heuristic classification brings clarity and comparability of different knowledge system approaches.

#### 2.2.4 Advantages and Disadvantages

The main goal for knowledge systems is to provide more effective and efficient use of knowledge in an organisation, as this is currently the base factor in determining the success of most companies (Schreiber et al. 1999).

The advantages for using a knowledge system are summed up in Table 2-1. The benefits of a knowledge system are different per situation, but are in many cases derived from the advantages mentioned in the table above. The main reason to employ a knowledge system is because the user is not capable of performing some tasks without this knowledge.

---

<sup>1</sup> Some have proposed parallel knowledge systems as the second generation of knowledge systems (van den Herik 1986).

<b>Uniformity</b>	The knowledge system will produce the same result in the same situation, because it is not subject to the preconceptions and prejudices that humans display.
<b>Reproducible</b>	The knowledge systems can be reproduced easily, by making copies of the knowledge system or reusing the knowledge model in another knowledge system.
<b>Effectiveness</b>	Where the knowledge may have been inaccessible or scarce it can now be made easily accessible, and duplicated. This allows the same knowledge to be used more intensively.
<b>Consistent, Transparent and Documented</b>	The inferences made by the knowledge system can be monitored, documented and explained in terms of the knowledge. The decision process becomes transparent to the user of the system.
<b>Efficiency</b>	A knowledge system can increase productivity and decrease costs by more effective use of knowledge in an organisation, resulting in a more efficient business process.
<b>Capture Scarce Expertise</b>	The knowledge system may be a way to capture and make explicit valuable and scarce expertise, which could otherwise be lost.
<b>Reliability</b>	A knowledge system is more reliable than a human, as it can be used 24 hrs a day, needs no sleep, and never gets sick.
<b>Education</b>	A knowledge system and the knowledge it contains can be used to educate novices in the field. It can teach problem solving methods, and deep domain knowledge. A knowledge system further has a training effect on users through usage.
<b>Breadth</b>	The knowledge of multiple human experts can be combined to give a system more breadth than a single person is likely to achieve.
<b>Timeliness</b>	Knowledge can be acquired just-in-time, instead of acquired (through training) just-in-case.
<b>Completeness</b>	The system can perform large numbers of tasks completely, where humans must limit themselves to a small number of examples, or to statistical analyses.
<b>Quality</b>	A knowledge system can be used to consolidate limited, low quality knowledge and provide a vehicle for improvement and knowledge development.

**Table 2-1 Advantages of Knowledge Systems (based on (Turban 1990))**

In most cases, the availability of the knowledge is the problem to be solved. Some experts exist that possess the knowledge, but these experts are hardly ever available. In a similar situation the expertise can be distributed among different experts and is therefore not accessible at a central location. The main problem here is not the complexity of the knowledge but merely the fact that some knowledge cannot be accessed in the support of some task. The users in this case are not experts in the task themselves but regularly require support from these experts. Features looked for in such knowledge systems are availability and reliability. In other situations a knowledge system may be developed because there is the need to collate, archive and inspect the knowledge to improve the quality of the knowledge. Features looked for in this type of knowledge system are consistency and quality. In either case it is not easy to enumerate the benefits in terms of direct financial benefits or savings.

The disadvantages attached to the use of a knowledge system are summed up in Table 2-2 below. Some see far worse disadvantages and question the entire enterprise of developing a knowledge system. Others see it as a doomed enterprise, never leading to any useful results. To take but one example, the Dreyfus brothers argue that knowledge systems could reach competence in narrow domains but could certainly not reach proficiency or expertise, which requires common sense and possibly 'holistic mental processes' (Dreyfus 1987).

<b>Common sense</b>	Knowledge systems lack common sense which makes them vulnerable when queried outside of their narrow scope. In addition to a great deal of technical knowledge, human experts have common sense.
<b>Creativity</b>	Knowledge systems lack the ability to improvise or come up with novel solutions.
<b>Learning</b>	Human experts adapt to changes in their environments and expand their knowledge automatically; knowledge systems must be explicitly updated. Some methods to incorporate learning into knowledge systems exist (e.g. case based reasoning and neural networks).
<b>Sensory Experience</b>	Human experts have available to them a wide range of sensory experience; knowledge systems depend on symbolic input.
<b>Brittleness</b>	Knowledge systems are not good at recognizing situations where no answer exists or where the problem is outside their area of expertise.

**Table 2-2 Disadvantages of Knowledge Systems**

Beyond the disadvantages mentioned, the cost and risk associated with knowledge systems are substantial. The cost is considerable because of the expense of the people involved. Both expert and knowledge engineer are expensive and the activity can be quite time consuming. The exploratory nature of the development means that the risk is great that no system will ever be produced. With unclear benefits and high cost and risk, the commitment from an organisation based on this kind of bottom-line is often ambiguous. This places the project under additional strain.

## 2.3 Knowledge Representation

Forms of knowledge representation are numerous, as many different approaches to represent knowledge are possible. Some of these are common to knowledge systems, such as procedural, logic and structured representations. Others appear predominately in other areas of AI, such as connectionist and case-based representations. The discussion in this section will start with the roles that knowledge representations play and should play in knowledge engineering. This is a starting point for an examination of knowledge representations found in knowledge systems and in other fields within AI. A succinct description of the representations is given, as more extended descriptions can be found in literature. Some specific representations with features relevant to the research in this thesis are given some additional attention.

### 2.3.1 Roles for Knowledge Representation

Knowledge representations play different roles in knowledge engineering and these roles have influenced the way that researchers and developers have addressed and used the different available knowledge representations. In (Davis, Shrobe & Szolovits

1993) five different but important roles for a knowledge representation are described. An additional sixth role is added as it is considered especially important in the context of this thesis. It is in fact an elaboration of the maintenance of knowledge systems and how knowledge representations support changes in the knowledge.

- **Surrogate** – it is a substitute for the world, which can be used to reason about the world rather than acting in it,
- **Set of ontological commitments** – what terms does the representation offer to describe the world,
- **Fragmentary theory of intelligent reasoning** – it tells us how the knowledge can be used, i.e. what inferences are sanctioned and what inferences are recommended,
- **Medium for pragmatically efficient computation** – besides being a representation of knowledge it must also serve as the basis of computation,
- **Medium for human expression** – the models made in the knowledge representation also serve to support the discussion of the knowledge itself,
- **Medium for learning and change** – the models constructed in the representation must be mutable/maintainable to allow for the dynamic nature of knowledge.

Any representation of knowledge must position itself along these six dimensions, and make the choices that are made one way or the other explicit.

### Surrogate

By surrogate, it is meant that the knowledge that is represented serves as a model of the world. Instead of acting in the real world, the model is used to make predictions about the possibilities to attain certain goals and examine the effect of actions in the real world. As a model, it is a replicate that is adequate for that purpose. In actuality, this is more the role that the knowledge plays, than the knowledge representation that enables it. Nevertheless, anything represented in a knowledge representation must fulfil this task.

### Set of Ontological Commitments

As a set of ontological commitments, the knowledge representation describes the types of elements that can be used to create models of the world. In a rule-based paradigm, the world consists of facts and rules defining relationships between those facts. The ontological commitments therefore limit the possibilities for modelling certain realities. In this way they also structure the manner in which such models can be made. This kind of ontological commitment is present in ourselves as well. The choice of knowledge representation “will bias our dispositions about which objects to consider more or less similar” and this “affects how we apply knowledge to achieve goals and solve problems” (Minsky 1990).

The ontological commitments therefore define what is recognized by the representation and what will be delegated to the background. With a hammer in your hand, you are more likely to appreciate the nail-like character of objects around you. Humans are predisposed to learn certain types of concepts and relationships better than others. For example, people are prepared to characterize colours in a certain ways, favouring certain break-up into predetermined categories. The cause of this lies in the biophysical makeup of the visual cortex and the structure of the visual brain centres (Wilson 1998). Minsky together with Papert calls this ‘prepared learning’

(Minsky 1989). This probably also extends to the concept of causality. Humans endeavour to find the causes of things, science being the pinnacle of this endeavour. Sometimes this search for causes goes so far that causes are invented or irrational causal links are proposed leading to individual and mass superstitions. A knowledge representation can function as a similar bias in the model's view of reality.

### **Fragmentary Theory of Reasoning**

A knowledge representation further embodies a theory of intelligent reasoning. It shows the possible use of the knowledge in the form of the inferences that are sanctioned by the theory, or those that the knowledge representation proposes. Some feel that such a theory must be psychologically viable, i.e. correspond to a model of human reasoning and have similar traits and flaws. Others generalise from this and search for viable models of general intelligent or rational action, without reference to the human context. Representation and reasoning are intertwined; they cannot be understood separately.

### **Medium for Pragmatically Efficient Computation**

One of the roles of a knowledge representation is the ability to execute or make operational models made in it. A representation must not be handicapped just for the sake of the speed of execution, but at the same time must not develop a representation that is too inefficient to compute. The critique by Davis is that the attention of designers of knowledge representation languages was in majority unconcerned by the efficiency with which calculations and inferences were performed. This is not always a concern in systems that aim at representing informal knowledge or knowledge in natural language (Lethbridge 1994).

### **Medium for Human Expression**

A very important role from the perspective taken in this thesis is the ability to express knowledge in a representation, to communicate this knowledge with others and with computers. This means that a knowledge representation can support the expression of concepts and relationships of different categories, or how easy is it for use to 'talk' or think in that language. Some knowledge representations exist that are not meant to be an expression rather a facility for storage of knowledge, in for example connectionist systems. It is tied in with the roles discussed before, but the 'distance' between a knowledge representation and human conceptions of a domain is directly related to the understandability of given knowledge representation.

### **Medium for Learning and Change**

The expressibility and understandability of a knowledge representation is very important in respect to the last role: the role of a knowledge representation as a medium for learning and its amenability to change. In part this is also where the support that a knowledge representation language gives to support the process of knowledge modelling, from inception and conceptualisation through development up to and including maintenance. It must therefore be a pragmatically efficient medium for expressing and maintaining knowledge models.

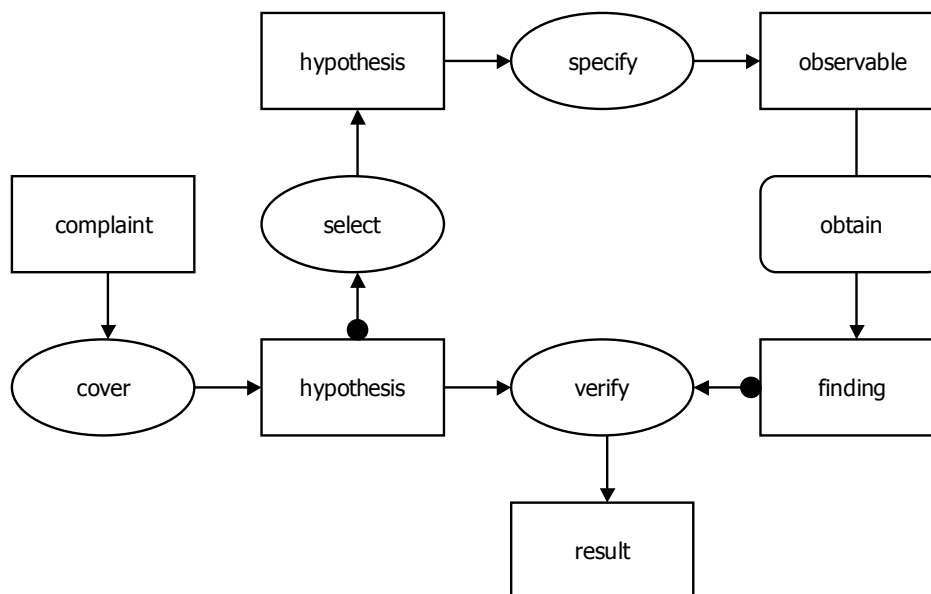
The representation must allow the user of the representation to easily change the represented knowledge without incurring too great a penalty. If this does occur then the representation is not suitable for representing knowledge that cannot be

guaranteed to be correct, nor can it be used in the development of systems in evolutionary methodologies.

To be able to change a knowledge representation, the represented knowledge must be understandable. It must support knowing what and where to change the representation. This is why a knowledge representation must also allow a global insight into its structural and specific content to those who attempts to understand and change it. If possible, the knowledge representation must actively support the creation of new knowledge by creating opportunities for the discovery of new insights.

### 2.3.2 Procedural representations

Procedural representations are not seen as a proper form of knowledge representation by all researchers. Some of the controversy that arose in the mid-1970s concerning whether knowledge is procedural or declarative is still felt (van der Velde 1986, Winograd 1975). Procedures contain knowledge in the form of a fixed control structure allowing execution of actions, making it possible to perform linear, cyclic or conditional actions. They represent an ossified solution such as a recipe or protocol. Such knowledge cannot be used dynamically, as the knowledge is compiled into a monolithical form. The procedure can only deal with aspects considered in the development of the procedure. Nevertheless, in knowledge systems procedural representations still have an important role to play. In approaches such as CommonKADS and others, they are featured as Problem Solving Methods (PSMs).



**Figure 2-7 Default Diagnosis PSM**

A PSM is a procedural form of knowledge that represents an archetypal solution to a specific kind of problem. For example, the PSM displayed in Figure 2-7 above is an inference structure for default diagnostic method (Schreiber et al. 1999, pg. 140). Dividing the domain into hypotheses, symptoms and tests is a perspective that can be used by many different diagnostic PSMs. PSMs are related to design patterns, as abstract solutions to a problem, without a reference to an actual domain in which it occurs (Gamma et al. 1994).

By classifying a task that has to be performed by a knowledge system into one of a hierarchy of abstract tasks, one of the PSMs methods associated with that task can be



used. The main division used for these abstract tasks is synthesis/analysis. Diagnosis falls under the category of analysis PSMs.

Beyond the representation of the global task that a knowledge system performs, such as in a PSM, a procedural representation represents tasks performed at lower level of detail within the knowledge model. For example, a procedure to optimise a combination of compounds to locate the best mixture is often more efficiently represented using a procedural representation. Some tasks performed by an expert are only known in a procedural form.

#### *Surrogate*

A procedural representation can make predictions about the world. It can embody a formula or recipe which when executed makes assumptions about the state of the world. As an algorithm, they are quite adept at describing behaviour of a system as a dynamic set of relationships between elements. The actual predictions depend on the quality of the total set of operations.

#### *Ontological Commitment*

Perceiving the world only through loops, conditionals and actions, procedural representations are myopic in respect to static relationships and concepts. Casting the world in a procedure to be followed can sometimes be the right perspective, but can also mean re-describing it in a form unsuitable or unnatural for the material at hand.

#### *Fragmentary Theory of Cognition*

Procedural representations form only a weak, partial theory of cognition, and are not focused on providing as a fragmentary theory of intelligent reasoning. Finite state machines deriving from procedures as a more general mechanism have been used for reactive agents however.

#### *Medium for Pragmatically Efficient Computation*

The efficiency of procedural representation is an important aspect to their use. Because the reaction in each situation is embedded and in many cases without requiring sensor input, it is clear which action to take.

#### *Medium for Human Expression*

It is quite easy to devise a procedure for a certain action, once the task is well-understood and there have been many experiences with the task. In many cases where knowledge is an issue, such a procedure is not yet available. To communicate with procedures is possible, but often this is not sufficient, as the procedure does not have the rationale behind it attached to it. Humans do use procedures quite often to communicate knowledge however, and are quite successful with it.

#### *Medium for Learning and Change*

Because of their ossified nature and lack of rationale procedures are hard to learn from. Additionally, without proper understanding of their context and intent, they can be exceedingly hard to understand (McConnell 1993). When procedures require change, the modeller is required to re-interpret the meaning of the algorithm, and based on that created understanding make the changes. Procedural algorithms do not structure or restrict their content; therefore, only domain independent and basically semantic support can be given.

Changing and gaining insight from procedures can be quite hard because they do not support jumping to conclusions. They have no direct support for learning and change, and go on the assumption that the formalised procedure is correct.

### 2.3.3 Rules

In this category fall approaches that have a fixed procedure for deducting facts using a separate representation for the rules that are processed. Each of these different rule types has a condition and a consequence part, i.e. a trigger and an action. Rules and related knowledge representations are all considered to be natural, precise, flexible and modular.

#### Logic

```
human (a) → mortal (a)
human (Socrates)
∴ mortal (Socrates)
```

**Figure 2-8 Socrates' Mortality**

Logic is the first and foremost representation of knowledge as knowledge. From axioms and basic propositions, additional propositions about the world can be proven using the rules of deduction. It was initially thought that such logic could represent any and all aspects of intelligence. The GPS was the first example of a system built along these principles (Newell & Simon 1963). Based in essence on positivism, it was thought that any knowledge could be represented in some form of logic. Nilsson offers John McCarthy (McCarthy 1968) as the first to propose that predicate logic as a representation of knowledge. Languages like Lisp and Prolog are extensions of this frame of thought.

#### Production Rules

Production rules are a more open-ended form of rules. Initially a production rule was an extension of procedural representations where a condition was a trigger for some procedural knowledge. This only allowed use in forward chaining inference. The procedure could assert new facts that would lead to other production rules to be triggered. Eventually more strict representations became the norm, only allowing the action to assert new facts, which allowed backward chaining approaches.

Special purpose extensions to these rules also exist. Two specific ones, ripple-down-rules and decision-tables are described in the next two sections.

#### Ripple-down rules

The ripple-down-rule approach is similar in way to the production rule system described above (Compton et al. 1992, Compton & Jansen 1990) The difference lies in the format of the rules. Instead of having a list of rules, the rules are organised in a tree. Each rule consists of a condition, an action, and an exception-clause. The exception-clause allows other rules to override the rule.

Ripple-down-rules are made for ease of maintenance. A system starts with a single rule. If a problem solution produced by the system is flawed according to the expert,

the last fired rule is the context to which knowledge to adjust the solution is added as an exception. This is in some way a reification of Minsky's exception principle (Minsky 1988).

This new rule is given the new problem and solution as its 'cornerstone' case, its origin. The cornerstone cases linked to each of the rules allow retesting the system against problems for which the solution is known in advance. This counts as verification and validation after a mutation of the knowledge. After each change, all the test cases can be verified, allowing for the detection of interferences. The approach has a drawback in that duplicate knowledge exists indifferent parts of the rule-tree. Changes to one rule are then not proliferated to the other. Only further experience will show from whether or not that would have been necessary.

### Decision-table

The decision-table format is a visual presentation of a number of related rules. Decision tables have a long history, albeit not as a knowledge representation format (Mors 1993, Verhelst 1980, Montelbano 1973). Among other purposes they were employed as tools in software engineering to formulate the different test condition and the associated outcomes. Additionally, they have also been used to produce test plans for electronic schema. Their use is not widespread but in knowledge engineering, there are some known uses in tools such as Wisdom, PrologA and AIONDS, as well as the two tools described in Chapter 4 and 5.

	<b>Wine type</b>	R1	R2	R3	R4	R5
C1	Main course	red meat	poultry		fish	ELSE
C2	Turkey is served	-	yes	ELSE	-	-
A1	Wine type	red	red	white	white	unknown

**Figure 2-9 An Example Decision-table**

A decision table consists of four types of elements: conditions, condition-alternatives, actions and action-alternatives. The conditions represent the relevant parameters that are the antecedents of the decision-table. In the above table, the conditions are `main course` and `turkey is served`.

The value of a condition is compared to the values represented in the condition-alternative. These values describe an exclusive and exhaustive domain of the condition. The condition-alternative where this comparison is true is the valid alternative. The sub-tree below that alternative is the chosen logic for the remainder of the consultation of the decision-table. For example, the `main course` in this setting is `poultry`. This is repeated for all the conditions until finally, the last condition-alternative is chosen (R2).

To finish, the actions in the chosen column perform the assignment with the value in the action-alternative. In this case, the left-most column only contains a single action `wine type` and the value inferred for the wine type in this decision-table is `red wine` by the action alternative.

The condition-alternatives under consideration at any time form an exhaustive and exclusive set of values. This assists in the verification of the completeness and correctness of the rules represented in the decision-table. Missing alternatives become

clear as they do not appear in the condition-alternatives and action-alternatives with no values can be clearly marked as gaps in the knowledge.

### *Surrogate*

Rules define relationships between symbols, with a connection to objects in the real world. These relationships can be calculated with and the symbols manipulated. The rules of deduction, or of plausible inference, thereby allow predictions about the world to be made. This can be used to derive new facts and predict new information. Depending on the perception of these rules, these new inferences can be provably consistent and correct, but in many knowledge systems, such guarantees are neither required nor supplied. The heuristic value of the relationship is sufficient in most cases.

### *Ontological Commitment*

In their base form, rules divide the world into facts and relationships between those facts. These relationships between objects are either seen as representations of real laws governing nature, or as useful abstractions enabling pragmatic action. Regardless, in either view they see the world as defined by causality. Logic brings a view of the world as consistent and correct governed by absolutes. A normal production rule states a relationship that should always hold, except when rule with higher priority exists. Ripple-down rules divide explicitly between the norm and exceptions, whereas decision-tables present knowledge as internally consistent, in exclusive and exhaustive form. Each of these forms brings a different slant to the world view, but at their basis lies a view where each result has a cause.

### *Fragmentary Theory of Cognition*

Rules are direct descendants of the symbolic theory of cognition. The theory that they profess is that of symbol pattern manipulation as necessary and sufficient for any intelligent system.

### *Computational Efficiency*

A rules computational efficiency is dependent on the complexity of the rule base, and the possibilities of the inference mechanism. The main problem is to locate the rules affected or required for a certain fact, but the Rete algorithm provides a good efficient solution to locate the set of rules that should fire under certain circumstances.

### *Medium for Human Expression*

Rules can be easily recognised as a human form of expression. Legislation is but on form where principles and laws are laid down for all to comprehend and implement. In the form of logic, it has provided an almost disambiguated form of communication. Humans have found it natural to express themselves in terms of rules and logic. Especially in the form as rules of thumb, the ability to use heuristics usefully was not lost on mankind, as is evidenced by our use of proverbs.

### *Medium for Learning and Change*

The inherent modularity of a rule based representation means that it is quite simple to add, remove or change an individual rule. Ripple-down rules incorporate support for this intrinsically into the representation. Decision-tables offer similar features, but extend this by providing opportunities for discovering new insights from the representation of knowledge itself. Inconsistencies such as a missing alternative are easily spotted, leading to immediate feedback on the represented knowledge.

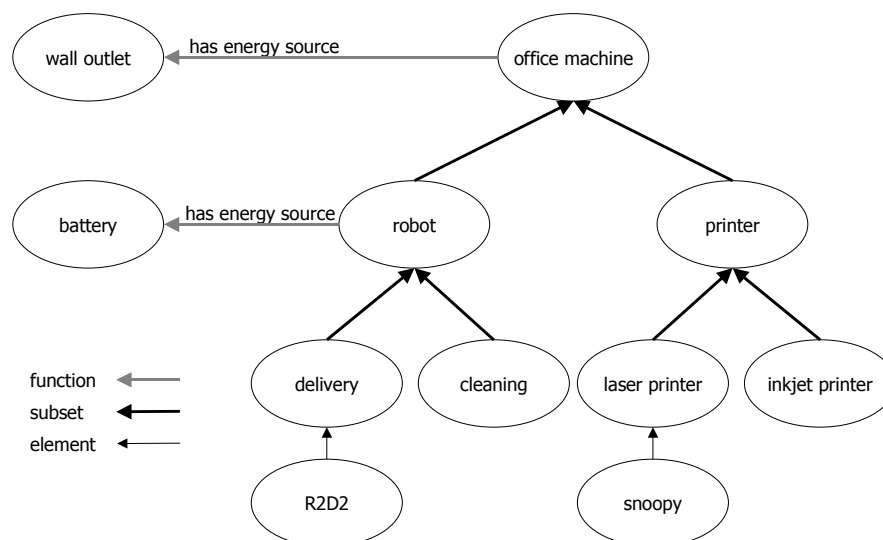
The problems lies in the scalability. It is impossible to extract a complete view of the role of a single rule within a set of rules. The interactions between rules can not be seen in the content of the rule itself. The sum of the rules is larger than the value of the individual rules. It becomes even more complicated when the rules are implemented to take advantage of proprietary behaviour of the inference mechanism (e.g. (McDermott 1982)). At that point it is only possible to understand the set of rules by analysing its response to large numbers of test-cases, as incorporated in the ripple-down rule approach.

### 2.3.4 Structured representations

Under structured representation fall a number of techniques that have influenced each other quite heavily. The three major ones used in knowledge representation are semantic networks, frame systems and object-oriented representations. The structural representations are different from procedures and rules, which allow the formulation of specific relationships between the values of certain facts. Structural representations impose structure on the facts themselves, by allowing the introduction of relationship between two facts, other than the heuristic association. They employ the principle of *localisation*, which means placing related items in close physical proximity to each other.

#### Semantic networks

A semantic network aims to represent the concepts between symbols by mapping the relationships in which they feature. A network made up of nodes joined by links, where nodes represent concepts, like a STONE, and the links represent relationships like PART-OF house or HAS-COLOUR red. The meaning of stone is derived from its relation to other concepts. Semantic networks may be loosely related to predicate calculus by substituting terms by nodes and relations by labelled directed arcs. Initially, semantic networks were employed to drive language understanding systems, now their role is seen as more general.



**Figure 2-10 A Semantic Network for Default Reasoning (from Nilsson 1998)**

Their nature as a network allows them to be visually represented and certain consequences can be arrived at from examining the diagram. For example in the network in the Figure 2-10 above, a kind of diagrammatic reasoning is possible: what

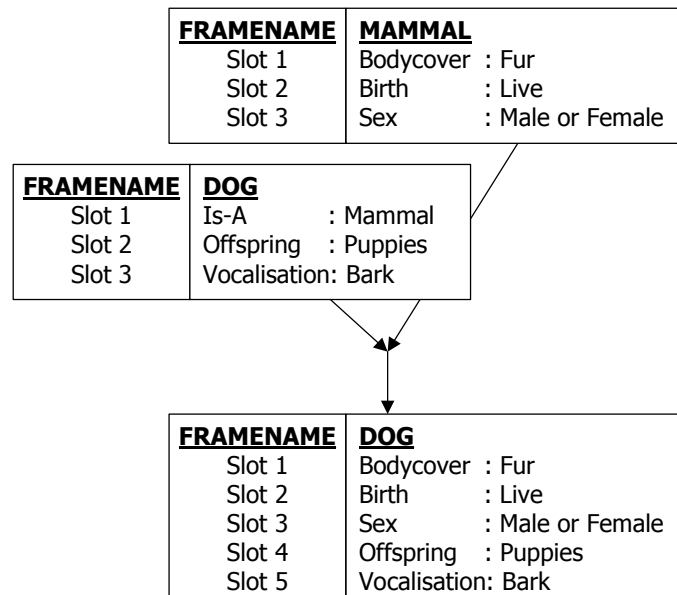
kind of energy source does R2D2 have, or what kind of robot is R2D2. By following the different links, meaningful inferences can be made.

The labelled links such as IS-A and PART-OF can be ambiguous especially if the same word is used in different ways. For examples, R2D2 IS-A DELIVERY means that the individual robot R2D2 is a delivery robot, a type to individual relationship. However DELIVERY IS-A ROBOT means that DELIVERY is a subset of ROBOT, a type to type relationship. This controversy and other have been the source of many different formats of semantic network with sometimes very little in common (Ringland & Duce 1998). The format above for example includes different types of IS-A as well as a version of override-able default reasoning. The network described that all office machines have a wall outlet as their source of energy, except for robots that derive their energy from a battery.

### Frames

The frame representation originates in a single paper by Marvin Minsky (Minsky 1975). Although little of the intent of the frame systems survived in the frames as they are presented here, the legacy of the paper is evident in many parts of knowledge engineering and other areas. Object orientation itself can trace its origins back to that paper, and so can case-based reasoning, both discussed in later in this section.

Frame systems and semantic networks have more than surface similarities. Frames or *schemata* are structured objects consisting of a collection of related facts, or slots as they are known as in frame terminology. A slots is a name-value combination and can contain a default value. Slots can also contain links to other frames in relationships such as IS-A and HAS-A. An example of a frame is shown below. Frames employ the principle of *encapsulation* which means the packaging of a collection of items. This takes localisation to the next level.



**Figure 2-11 A Small Frame System (Ringland & Duce 1988)**

Frame-based representations encourage jumping to possibly incorrect conclusions based on matches, expectations or defaults. They therefore are not proven to deliver correct actions. A prime example is the form of concise notation used in frames by leaving out those elements shared by two frames that have an IS-A relationship, as

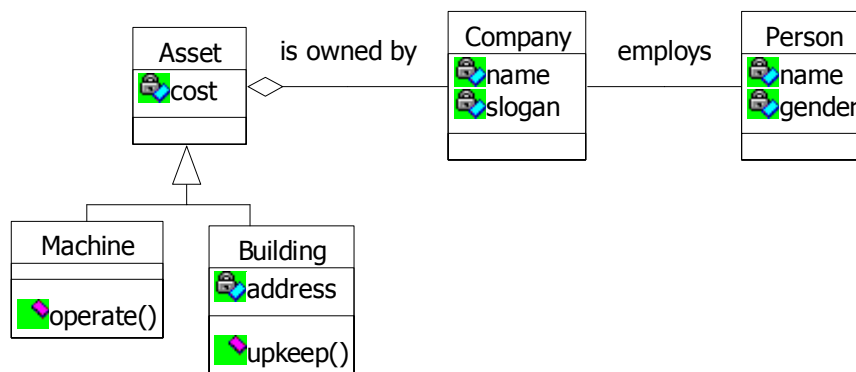
illustrated in Figure 2-11. This is also called inheritance. The child frame, the Dog in the image, is assumed to have the same content as any mammal, except when stated otherwise. Therefore a dog can be queried as to its *bodycover*, yielding the answer *fur*.

Another example, frames employ default values that apply *unless* they are displaced by information to the contrary. The normal colour of an elephant is grey, but white elephants also exist. In understanding a sentence like “At the circus, the elephants were made to walk on their hind legs”, there is no reason to assume that there is anything exceptional about the elephants, so the defaults apply. Logic representations do not handle such defaults very well, as it means that a value once was determined ‘grey’, is displaced by ‘white’. This requires extensions on first order predicate logic.

The force of the statement is “in the absence of any evidence to the contrary assume the elephant is grey” (Ringland & Duce 1988). Frame systems emphasize a practical aspect of reality, namely that it has norms and exceptions, and that it is practical to make use of norms, even though this may sometimes lead to errors. In the long run, the strategy works efficiently. These systems make these assumptions explicit and can reason about them, whereas many other (e.g. logic) systems do not.

### Object orientation

Object orientation describes the world as a collection of objects. An object is “[a] discrete entity with a well-defined boundary and identity that encapsulates state and behaviour” (Rumbaugh, Jacobson & Booch 1998). A set of objects with the same structure and behaviour is described using a class. For example, different persons all have a name, a gender and other attributes. Many people are employed by a company. There are many different types of companies. A company in turn has a name, a slogan and owns a set of assets, like machines and buildings.



**Figure 2-12 Assets, Companies, and People in UML**

In an object oriented model, this is visualised as in the above Figure 2-12, following the Unified Modelling Language (UML). The triangle shape indicates inheritance, meaning that a Machine also has the same attributes as any other Asset. This is analogous to inheritance in Frames. The diamond indicates aggregation, meaning that the Assets like Buildings belong to the Company. The 0..\* indicates 0 or more assets are owned by one Company. The line connecting the Person and the Company has a label ‘employs, which means that this user-defined association needs to be read as ‘a Company employs many Person’ or equally ‘a Person is employed by a Company’.

In many ways object orientation is the culmination of semantic networks and frame systems. Object oriented models also consist of concepts linked together through

different relationships, denoting inheritance, ownership and user-defined associations. Object orientation employs and extends the modularity of frames and its inheritance. It transformed a variety of different yet equal relations between concepts from semantic networks to a number of predetermined and user-defined relationships. Each of these types of relationship has limitations and specific semantics attached to it.

One of the important aspects that object orientation adds to encapsulation is *information hiding*, meaning that certain aspects are hidden from outside parties. Sometimes encapsulation and information hiding are seen as the same thing, but hiding adds concepts such as private and public parts of a class interface. This separates *what* an object does from *how* that is achieved. This form creates the ability to for *abstraction*, which is a mechanism for focusing on the essential details of a concept, while ignoring the inessential details. For example, a modeller is allowed to a Building and a Machine object as the same kind of object at the level of Asset. Both have cost as an attribute.

Taken together this allows object oriented models that offer primitives to express structured concepts, attributes of concepts, instance-type relationships, part-of, and kind-of relationships. Object orientation stresses the modelling aspect of representation. Beyond a receptacle a model also is meant to function similarly to a specific viewpoint on reality. The models that can be constructed have a high degree of match to the perception that a person has of a domain.

Object orientation is a direct representation, representing concepts that are also found in reality. In addition, when an element in a model is acted upon in some way, it is supposed to react analogously to a similar act on a similar element in reality. Insights gained through simulations with the model then have a matching effect in the real world.

### *Surrogate*

A structural representation has most often a direct resemblance to a domain, as seen from the perspective of participant in the domain. It emphasises the relevant concepts and leaves out residual issues, that do not feature in the perspective. It therefore strongly emphasis the surrogate aspect, especially seen in object orientation.

### *Ontological Commitment*

As an ontological commitment a structural representation shows a world consisting of concepts and relationships, i.e. a conceptual view of the world, as position opposite from the procedural knowledge. It describes static relationships, and separates the world into elements with boundaries and identities. A concept is a composite of facts, with state and behaviour, that derives much of its meaning from the relationships it has to other structures.

### *Fragmentary Theory of Cognition*

In part, the structured representations also stem from the symbolic theories of cognition, as increasingly more defined symbols and concepts. Seen as such they perceive concepts in the mind as structured and related. Navigating and examining these structures, new facts can be derived through examination of the concepts and relationships. A very strong role is played by reasoning based on norms and exceptions, which is treated as a natural, even if not always guaranteed to deliver correct results.



*Computational Efficiency*

The incorporation of structured representations into AI and software engineering has shown that they can be realised quite efficiently and can provide further efficiency for other representations incorporated into it. The decomposition allowed by structural representations, allows division of the set of all rules into smaller sets localised within each construct. This provides far more efficient derivation of new facts.

*Medium for Human Expression*

Structured representations are quite good at representing human forms of knowledge. Concepts and relationships come easy to them as abstractions or models of reality. Abstracting away from unnecessary detail is perhaps one of the most human cognitive capabilities. Many forms of model used to describe artefacts or constructs are made in this form. Especially object-oriented diagrams have proven their worth as means to communicate complex ideas succinctly and understandably.

*Medium for Learning and Change*

The direct connection to a certain reality in structured representations allows the modeller to go back and forth between the model and reality. Therefore, a missing concept or inconsistency between the model and reality is easily spotted. This aids the formulation and understanding of such models, the localisation of where such changes need to be made and the discovery of new insight.

The decomposition and encapsulation assist by dividing the problems that exist. This helps to focus the effort, and localises any problems that are found. Changes can be limited to within a certain class for example, and because of its encapsulated nature the *how* may be changed, without affecting *what* the class does.

**2.3.5 Case-based reasoning**

Case-based reasoning is an approach to reasoning that emphasis the use of memory. “A case-based reasoner solves new problems by adapting solutions that were used to solve old problems” (Riesbeck & Schank 1989). A doctor who diagnoses a patient by recalling another patient that exhibited similar symptoms is using case-based reasoning.

Case-based reasoning has been formalized as a four-step process (Aamodt & Plaza 1994):

1. RETRIEVE the most similar case or cases
2. REUSE the information and knowledge in that case to solve the problem
3. REVISE the proposed solution
4. RETAIN the parts of this experience likely to be useful for future problem solving

A case-based reasoning system has access to a collection of cases, describing previously experienced problems and their solutions. Facing a new problem the system locates the case that is most similar to the current situation. In many cases this means calculating the similarity between the current problem and each of the cases in memory. The best case is used to provide the solution, which is adapted to the current problem situation. This new experience is stored in memory, so that it can be used again.

The initial concepts that form the basis of case-based reasoning originate in Schank and Abelson (1977) who looked at the human mind as composed of memories of previous experiences. They rejected the hypothesis that human cognition depends on a set of reasoning principles or rules and introduced so called 'scripts'. The scripts constitute much larger chunks of knowledge to be used as primary knowledge structures. Such a script can describe an event like 'visiting a restaurant' or 'waiting in a doctor's reception'. Thinking then means finding an available script to use, that is right for the current situation and applying that script, rather than generating ideas from first principles.

The cases are described in most case-based reasoning systems in ways similar to frame- and object oriented systems (Hammond 1989). As sets of attribute-values pairs a case merely describes a series of facts. For example, a medical system might represent its cases as patients with certain attributes like age and gender as well as the symptoms and final diagnosis. Systems using monolithic structures like frames are very common, but composition of case structures are also known (Redwood 1993).

If sufficient similar problem situations occur repeatedly over time, the experiences can collate to form more general or abstract experiences, referred to as prototype or *ossified* cases (Riesbeck & Schank 1989). These general principles are also thought to form part of the way experiences are retrieved (Kolodner 1993). They can be perceived as a set of norms or rules, but are formed as an aggregation of similar cases, grounded in a stream of experiences.

#### *Surrogate*

A case-based reasoning system assumes the world to behave as it did before. It predicts new facts about the world based on previous experience. The system is therefore as good the memory of its experiences. In many cases, this provides an adequate model of reality.

#### *Ontological Commitments*

From the viewpoint of data, a case-based reasoning approach divides the world into structural entities, like frame-based and object oriented approaches. These describe how experiences are recognized and stored. From the point of view of reasoning, the only inferences that are sanctioned and proposed are those that can be supported by previous experience. These inferences are not provably sound, and may be inconsistent. They are however, tied to the actual behaviour of the system in reality.

#### *Fragmentary Theory of Cognition*

Case-based reasoning proposes a theory of cognition that centres on retrieval of previous experiences from memory. It explains problem solving as based on a heuristic relationship between surface similarities describing individual cases. This does not require a deep model consisting of rules, but merely exploits history's tendency to repeat itself. As a theory of cognition it also integrates certain rule-based aspects, which it incorporates as ossified cases.

#### *Computational Efficiency*

The efficiency of a case-based reasoning system can be limited, as the complete case-base needs to be examined for similarity in basic schemes. This makes the efficiency of retrieval fall with increasing experience. This is an active area of current research (Smyth & McKenna 2000).

*Medium for Human Expression*

From the point of view of expressing knowledge, there are few approaches that are more natural to humans than discussing and describing individual cases. As medical case-histories, jurisprudence, or war-stories to be found in almost all walks of life, humans like to express themselves through examples and analogies.

*Medium for Learning and Change*

This also means that case-based reasoning systems are quite easy to change as it only requires addition of new cases. Changing old cases may not always be as easy. Extracting more meaningful insights from the model requires additional facilities, akin to data-mining. It therefore does not automatically lead to new insights.

**2.3.6 Connectionism**

Connectionism is a blanket term that covers a number of approaches characterised as stimulus response systems. Neural networks are perhaps the best known examples of these systems. Neural networks have been given their name because of their relationship to biological neural systems, but the semblance to the natural form is superficial at best.

Connectionist systems calculate the output from input signals modified by a set of adjustable weights. Adjusting the weights will give a different result when given the same input. This allows a connectionist system such as a neural network to be tweaked to give a specific output for a certain input configuration. Different algorithms exist which allow such a configuration of weights to incorporate a mapping between many input-output pairs. After exposure to a set of samples problems paired with the output that would be appropriate for each the inputs such a system can be said to have learned a mapping between an input and an output.

Most connectionist systems have characteristics that allow them to have a good probability to give an approximately correct answer when confronted with unseen input patterns, when their training set is representative of the inputs the system is likely to receive. The essence of these systems is a stimulus response approach in which the knowledge, if one can speak of such a thing, is represented in a distributed form over the different connected parts (i.e. the weights) of the system.

These systems cannot be constructed based on an account from an expert, but have to learn the relationships from actual data examples. While this is also not considered to be a knowledge representation in a true sense, because it leaves the knowledge implicit, it can fulfil the same functions as the other representations.

*Surrogate*

Connectionist representations such as neural networks specifically make predictions about the world, based earlier experiences and on a current description of the surroundings (inputs and outputs). As a black box they may deserve the term surrogate, but their internals do not allow inspection and do not symbolically represent the problem..

*Ontological Commitments*

Analogous to a case-based reasoning system, all its knowledge is experiential in nature, deriving purely from its training set. Connectionist systems cannot be described in term of ontological commitment, unless one attempt to appreciate their

subsymbolic nature as mapping simple input signals to output signals. There is no representation of intermediary concepts or inference process using the representation to plan ahead or make conjectures on evidence. The system acts on signals in ways it has been taught to act.

#### *Fragmentary Theory of Cognition*

As a theory of intelligent reasoning it stays close to the strong situated form of cognition, where the entire system arranges itself to the current situation, reconstructs itself if you will, and through its activity allows intelligence to emerge. A connectionist system is a representation modelled on the same implementation hardware as humans.

#### *Computational Efficiency*

As a truly compiled form of knowledge or information, a neural network, is extremely efficient in use. It replaces all kinds of inference process, by a simple additive and multiplicative formula, from which the answer is derived.

#### *Medium for Human Expression*

Direct manipulation of the distributed representation is not possible, or at the very least impractical. Therefore, its expressiveness is limited to the provision of a training set. This means it is not in a true sense amenable to human expression of knowledge.

#### *Medium for Learning and Change*

A connectionist system is in its configuration a black box. Although it is trained to operate based on different cases presented to it, the exact training of a system remains to be an art form, and is not yet an exact science. Changes to its training set may have the desired effect, but the exact effects are hard to ascertain.

The maintenance of connectionist systems can be different for each of the approaches. Most approaches apply some form of off-line learning, whereas others apply on-line learning. The craft of training connectionist systems is still a dark art at times, as experienced people are needed to configure the system. Semi-automated approaches to this configuration problem also exist (Spronck 1996, Boers & Kuijper 1992).

### **2.3.7 Conclusions**

This section has shown a variety of ways to express, make operational, communicate and learn knowledge in different forms of representations. Many seem to have a certain degree of overlap but several categories can be discerned. Each of these representation arises from its own particular background which means it focuses on one or more of the roles to the detriment of others. All six roles feed into the ability of a knowledge representation to embody certain quanta of knowledge. The roles also make clear that even though some of the representations may be equivalent, even provably so, this fails to capture the 'mindset' and the insight behind such an approach.

Special attention was given to the aptitude of knowledge representations to support expression and change. This shows that these representations are perhaps good for expressing knowledge, but do not always have sufficient support for changing and extending knowledge models. This can be taken as further evidence of the attention of knowledge engineers and researcher for building knowledge systems rather than

changing and maintaining them. The focus seems to be on representing existing knowledge without attention to discovering the knowledge in the first place.

The authors of the original article only discussed five roles: Surrogate, fragmentary theory of reasoning, ontological commitments, medium for expression, medium for learning and change. The sixth role is missing because the authors approach knowledge as static even though they are quite explicit about their view of knowledge as dynamic. Even they are susceptible to the assumptions of knowledge representation as an 'after the fact occurrence', without truly appreciating its ability to empower human to discover knowledge.

From the examination of the different knowledge representations used in the development of knowledge systems one can conclude that the major representations used are made for expressing knowledge are logical oriented and structured. There is much attention for the analysis of that knowledge as to its correctness and completeness. These representations heavily influenced by symbolic assumptions. They aim to embody knowledge as is and demonstrably verify and validate that it is what was required in the first place.

Two knowledge representation techniques fared especially well in terms of supporting the development of new insights as well as their changeability: decision-tables and object-oriented models. They would fit well within a situated approach to knowledge system development.

One aspect has remained outside of the discussion until now, which is the aspect of fitting knowledge representations to the knowledge. Each of the knowledge representations was approached individually. However, no single representation has the flexibility to represent the full spectrum of human knowledge. To enable the representation to remain transparent, to show and make clear the content of the knowledge, it is probable that different representations have to be used. An integrated framework for different knowledge representations is required to capture to full extent and intent of our knowledge. Therefore a knowledge development requires integrated knowledge representation architectures, or risk mangling certain aspect of a domain to fit the chosen representation. The further a representation of some knowledge distances itself from reality the harder it is to derive new lessons from it that also have meaning in the world they represent.

## **2.4 Engineering Knowledge Systems**

This section describes the history and state-of-the-art of knowledge engineering as the process of knowledge system development. It starts by an overview of the history of knowledge engineering. This is followed by a general introduction into the different tasks that are found in knowledge engineering. It then goes on to discuss how these are employed in the different approaches.

### **2.4.1 From Art to Discipline**

In the early years of knowledge system development ad-hoc and rapid-prototyping approaches were the only approaches known. This allowed early knowledge engineers to create systems where little pieces of knowledge are added piecemeal. It is even questionable whether at that stage it could be dubbed engineering. The systems were

developed as throw-away prototypes<sup>2</sup>. Like early plane builders, a new contraption was built, pushed off a cliff, and after watching it crash, the process was started over again.

*Rapid prototyping* is an approach characterized by short iterations over a system, each time adding a new feature or some new knowledge. The goal of rapid prototyping goal is to produce practical results quickly. Based on the result of each cycle, the direction for the next cycle can be determined or the result of the last cycle can be reversed. In a number of iterative cycles, the improvement in each cycle is based on the lessons learnt in the previous cycle. This makes it highly adaptable and experimental. It works well in exploratory development methodologies, and prototypes are still developed to gather requirements for an eventual product.

Early knowledge systems were very successful in applying rapid prototyping, because of their fixed shell vs. flexible rule-set architecture, enabling this quick turn-around time. Where software was being created through an arduous process, knowledge engineers were blessed with tools that enabled very fast development of a working system. Adding a single rule was often enough to create a basic system. Adding more rules was the trick. This approach was also seen as a benefit to extract the relevant knowledge during each iteration, until such a time that the knowledge was considered sufficient for the target system. From conception to adolescence, knowledge systems are typically constructed through an iterative, exploratory process. “While other large software systems may also grow by accretion, for expert systems this [iterative] style of construction seems inescapable.” (Buchanon et al. 1983).

After initial successes and increasing numbers of failures, a need was felt to convert the dark art into something more tangible, i.e. manageable, plan-able and predictable. In the same period when DENDRAL and MYCIN came into existence, the term software engineering was coined and advances were made in controlling the process of software development (Endres 1996). The notion of software engineering implied that software manufacture should be based on the types of theoretical foundations and practical disciplines that were established in the traditional branches of engineering. This need was motivated by the so-called ‘software crisis’.

Moreover, people started to recognize that there is more to knowledge system development than just knowledge acquisition. This was the start of approaches that attempted to copy some of methodological developments in software engineering. The art of knowledge system development was beginning to show progress towards a discipline. Examples from software engineering that inspired knowledge system development were the waterfall- and later also the spiral models of software engineering (Boehm 1988). Adaptations were needed because traditional methodologies were not entirely suited to the special needs of knowledge systems, just as software engineering found that there were important differences between traditional engineering and software engineering. This led to methodologies specifically suited to the development of knowledge systems. Knowledge acquisition is of course one of the main differences with standard software engineering.

This is the time that second-generation knowledge systems came into existence. Attempting to solve the problems with knowledge acquisition based on knowledge

---

<sup>2</sup> Quite literally, in (Buchanon et al. 1983) there is a maxim for constructing a knowledge system that says “Throw away the Mark-I [prototype] system”.

models: increasing the model complexity to include more elaborate modelling constructs and by offering far more elaborate inference capabilities. This mirrored some of the advances made in software engineering using design based approaches to create complex artefacts.

### 2.4.2 Development Tasks and Phases

The advent of second generation knowledge systems as gave rise to the use of development methodology, i.e. a more structured process and ordered task-structures. Different researchers have different approaches to the development life-cycle. The iterative character of development is found in all approaches, but centres on different aspects. If we examine the partitions used by other developers of knowledge systems we see a number of recurring types of tasks over time. These are used to give a general view of the development process.

The tasks in the development of a knowledge system are split by (Hayes-Roth et al. 1983) into: Identification, Conceptualisation, Formalization, Implementation, and Testing. As a corollary, they state that “[a]n expert system evolves gradually”. This in turn means that they perceive this development as a design-based iterative process. In fact, the direction of this work is towards a final goal, which is approached iteratively.

Steels (1992) sees Preparation, Construction, Product Development and Evaluation. This division is different because of the explicit attention for the development of a product based on a knowledge model, separating the concern of the system from those of the knowledge acquisition and representation.

CommonKADS has the most elaborate of phases: Analysis, Design, Implementation, Installation, Use, Maintenance, and Knowledge Refinement (Schreiber et al. 1999). Of these, CommonKADS seems to spare most room for pre-knowledge system and post-design activity. Another important aspect is the iterativity of the development, through their use of a spiral development model.

<b>Phase</b>	<b>Description</b>
<i>Inception</i>	Identification of the problem, description of the problem, examination of possible problem solutions.
<i>Design</i>	Creation of model of solution and acquisition of required knowledge.
<i>Implementation</i>	Transformation of model to implemented system using the design as a specification of the target system.
<i>Deployment</i>	Deploying the system and training the users of the system towards use.
<i>Maintenance</i>	Adjusting the system for flaws and errors, perhaps slightly extending the system’s functionality.

**Table 2-3 Development Phases**

This simplified division is not to propose a new development phasing, as it is more general than one for actual use, it serves to elicit and equalize some of the concerns of the different approaches. In a way, this is a counterpart to the earlier architecture of a knowledge system, which is a structural sketch, whereas this is a behavioural sketch. It serves as an analytical tool. These phases may occur in a straight waterfall model, or in a spiral development methodology.

### **Inception**

In this phase the problem is identified, an analysis of the problem is made and possible solution methods are examined. This examination is not necessarily part of knowledge engineering, but is more properly placed under knowledge management. This is also the phase in which choices are as to what functionality to add, or in a spiral model, what part of the knowledge model to change and expand.

### **Design**

In earlier years, knowledge systems were mainly seen as stand-alone systems, performing a single task. Later integration of such a system with existing infrastructure became more important, eventually leading to the use of knowledge technology as a component of other systems. This makes the design of the systems and its relationship to other systems more important.

While other concerns therefore can play a considerable role in the design of a knowledge system, what sets the design of a knowledge system apart from that for a normal software system is the knowledge acquisition. The modelled knowledge can be seen as a specification for the implementation at a later date. This division can be seen as analogical to split the knowledge level vs. symbol level.

#### *Knowledge Acquisition*

The internals of a knowledge system are to an overwhelming degree a result of knowledge acquisition. The basic model for knowledge acquisition has been one where a knowledge engineer is a mediator between the expert, and the knowledge model. The knowledge engineer elicits, encodes and refines the model in collaboration with the expert to achieve acceptable performance. This makes *knowledge acquisition* the process of examining knowledge sources or collaborating with experts to devise a knowledge model that reflects an operational perspective on the knowledge required to perform a certain task. It has been found to be hard to extract knowledge in a computer usable form existing knowledge sources, such as experts. The knowledge acquisition process is therefore seen as the bottleneck in the construction of knowledge systems (Feigenbaum 1977).

There are many different tools at the disposition of the knowledge engineer, to assist in teasing out the knowledge. Some are structured, whereas others are more freeform. A number of examples are:

- Interview-techniques
- Brainstorm-sessions
- Case-research/-sorting
- Protocol analysis
- Structured walkthrough
- Thinking-aloud sessions
- Data-mining
- Observation
- Prototyping
- Repertory Grids

These methods aim to explore the knowledge of the expert where he is incapable of expressing it himself directly. Beyond this the knowledge engineer and the expert can use the knowledge model and discuss it together and come to changes and new



insights by discussing it. This requires the models to be transparent to the expert. In addition, by examining the operation of the system of different cases, the expert can ascertain the quality of its solutions, providing further insights and localize possible flaws in the reasoning. This is what can be achieved by prototyping.

In the early days of knowledge system development, knowledge acquisition was seen as a process of *transfer*. The knowledge engineer took the knowledge from the head of the expert in the form of rules. The knowledge engineer would literally ask the expert for rules that would apply in certain situations. These rules would then be translated into a formalized computer representation, ideally based on some form of logic. Experts were however unable to produce useful rules in this way. This difficulty was initially explained as a communication problem between the knowledge engineer and the expert, and a mismatch between human and computer representations (Buchanon et al. 1983).

In software engineering, design and role of methodology increasingly enabled successful development in the face of the problems experienced during the software crisis. Models were used to create a vehicle for discussion as much as to develop requirements for an eventual system. In reaction to the problems and due to influences from software engineering, knowledge acquisition evolved into a modelling approach. In the modelling view of knowledge acquisition, the role of the knowledge engineer is to structure the domain knowledge, to identify and formalise important concepts and relationships. The knowledge engineer creates a theory of the domain and then makes that theory operational. This constructive process is supported by the expert that can inform the knowledge engineer. The expert can be presented with specific cases and is queried for the solutions for such specific problems.

The change in attitude to knowledge acquisition has affected it much, as the knowledge model is seen as separate from the way a human expert would approach it. The psychological adequacy of the approach is relinquished, and there is no assumption that the knowledge in humans is representational as well.

Many of the problems that make knowledge acquisition hard still persist. The first problem is the tacit nature of much of our knowledge. Tacit knowledge is often highly personal or weaved through the cultural background. Such knowledge is almost impossible to explain or impart via typical teaching methodologies. Usually it is conveyed through apprenticeships, where a student can absorb the knowledge as it is being applied and expanded, thus developing ownership of the knowledge through use. Knowledge acquisition is therefore in many cases concerned with turning the unspoken into explicit, formalised axioms. Understandably this can be hard to impossible to accomplish in a single movement.

A second problem is the lack of knowledge on the part of the knowledge engineer. The initial vocabulary used by the knowledge engineer is inadequate to discuss the problem solving and the knowledge engineer. This causes many communications problems for the knowledge engineer in his role as mediator. It also means there is a considerable lead time before the knowledge engineer can hold meaningful discussion with the experts.

A third problem lies in the fact that experts employ the notions and ideas about their domain in a certain way, which may not be easily translated to a formal representation. The conceptual difference can be great, when considering the formality of the rules that can be employed. A model constructed on a certain domain

must contain conceptualisations that are recognizable to an expert. It must become in a way a map of the territory. Although Clancey is determined that the map is not the territory; the map must resemble the territory in recognisable ways for the expert to tell where it is wrong, or to translate insights gained when developing the model back to the expert's own knowledge of the domain.

This is also related to the tools and technologies used, which are often limited in the number of knowledge representations that are offered. The knowledge representations used can be limited in expressive capabilities, which make it more difficult to translate the knowledge in a knowledge model to an implementation. This increases the mismatch between an expert's view and the knowledge model's view on the domain. This can make the model truly unintelligible for an expert, making it impossible for him to comment upon the content directly.

Knowledge acquisition itself is in the view of most researchers an iterative process itself. The knowledge acquisition process must create the knowledge model with the expert, by an interactive synthesis. The constructions devised by the knowledge engineer, are scrutinized by the expert, based on whose comments adjustments may be made.

Some solutions have been examined to solve the problems. Reuse of knowledge is an important approach to offset the cost and risk of knowledge acquisition. Reuse is another approach originating in the static and absolute nature of knowledge. It enables the cost of development to be shared over difference knowledge systems making the enterprise more economically attractive. To some the development of reusable libraries is a "*conditio sine qua non* for improving the state of the art in knowledge engineering" (Schreiber et al. 1999, pg. 124). But even in software engineering, the benefits of reuse are yet to be clearly demonstrated (Menzies 1998, 1997, 1995), especially reuse without adaptation or change. There is certainly no reason to assume that the situation is more amenable for knowledge systems.

The reuse of knowledge is most clearly seen in the CYC project, which attempts to make a multifunctional 'cyclopaedia of common-sense knowledge. This large-scale knowledge base is meant to be reused by many different systems, and has been under development for over 10 years (Lenat & Feigenbaum 1989). It seems that the company is currently poised to present the knowledge base to potential customers. It is currently being used to power a new search engine for the Internet. The goal of the CYC project is to break the 'software brittleness bottleneck' once and for all by constructing a foundation of basic common sense knowledge.

Among other solutions explored is automatic knowledge acquisition, as in TEIRESIAS (Davis 1980) and the SBF framework (Marques et al. 1992). Here the knowledge system automatically queries for new knowledge when an inconsistency or hiatus is found. Beyond these examples there the approach is not found in any practical knowledge system implementations.

Machine learning is seen as an alternative solution, e.g. (Langley & Simon, 1995). By placing less reliance upon the existence of human expertise and the competence of knowledge engineers it has the potential of being a very powerful solution indeed. The ability to create a knowledge system directly from experiences and automatically keep it up to date would most certainly. To date not all fields have been amenable to such approaches, which limits the applicability of machine learning.

### **Implementation**

In this phase, the design is translated into a working system. The design includes any formalisations of knowledge that may have been devised through knowledge acquisition. Whereas in software engineering this may mean the actual implementation and the first time a working system may be present, in knowledge engineering things are a bit different. Many shells and development environments exist that allow the designs to be executed immediately. These may have been shown to the expert and some of the intended users. This was seen in the rapid-prototyping approaches earlier, but such systems are still available.

The higher regard for design and proper methodology has meant that such tools are not the immediate choice. Approaches to knowledge system development therefore separate the implementation from the design more strictly. For example, CommonKADS has seen some development of tools that allow the models to be operational. It describes an implementation as a structure preserving transformation to some system. It seems to propose an implementation in any substrate, be it a standard programming language or a dedicated environment for developing knowledge systems, such as AIONDS. Implementation can therefore mean many different things.

As seen in Steels subdivision of tasks however, there can be considerable work to be done to translate the knowledge model into a software system. This can entail integration with other systems as well as fitting the knowledge system with an easy to use, appealing graphical user-interface. There is not always very much attention to this, especially when these systems are made as prototypes or proof of concept systems. The technical possibilities to add user interface niceties are not present in all tools that are developed. Or it is implemented in such a way that there is a marked difference between a knowledge system and a standard software system, for example by realising the user interface with a proprietary set of buttons and forms. The cost of not developing a system customised to the needs of the users will be directly noticed in the phase deployment following implementation.

### **Deployment**

In this phase, the system is deployed to its users. Education may be provided to introduce and train people in using the system. The introduction of the system is a short but difficult period, where the users are for the first time confronted by the system. In earlier phases, some ambassador-users may have been present in the development of the systems content and form, which can greatly improve the quality of the system and the acceptance by the users. Deployment also occurs when an updated version of existing software is published to the existing user community.

Deployment is a risky enterprise as the ideas of the development team are pitted against the expectations of the client. It is very hard to change the preconceptions based on a first encounter with a system. The time it takes for a development team to produce their first release version is constantly at risk of being killed as soon as they unveil their creation. In most cases working towards an earlier release has considerable payoff.

### **Maintenance**

The maintenance phase is where the system is in use and feedback from the users can be expected. While many of these bugs can be logged and used to steer a following phase of activities, some of them may require immediate action, to repair possibly

critical errors from the system. This is normally seen as a phase where the system is not entirely in line with the requirements. There will be errors, which when discovered need to be repaired, and there may be some extensions which may have to be added to the existing knowledge.

Maintenance is stated by many researchers to be time consuming, perhaps even more so than the development itself. In the early days of KADS (the predecessor of CommonKADS), the problems in maintenance were attributed not so much to the frequent changes in the knowledge itself, but due to the lack of transparency and structure. These make it hard to know where the changes and enhancements should be made (Breuker & Wielinga 1986).

Reports of fielded and maintained knowledge system are rare (Menzies 1998). Two systems have been described in detail as to their maintenance and operation over a significant period. XCON has been deployed at DEC for several years. The effort in building it stands in bleak comparison to the effort need to keep the system operational. Over half the rules in the system are replaced on a yearly basis (Bachant & McDermott 1984). The second system is the GARVAN ES-1 system developed using ripple-down-rule technology (Preston, Edwards & Compton 1993, Compton et al. 1992, Compton & Jansen 1990).

A further example detailing some of the maintenance activities and employing the CommonKADS methodology is FraudWatch (Killin 1993). There are no known follow up articles however. Killin first mentions the need for maintenance to improve the effectiveness of the system (capture more fraud, reject more genuine transactions). This goes into deepening the models classification hierarchy and acknowledges the drift in fraud patterns through changes in the environment.

There really is or should be no real difference between knowledge acquisition, and maintenance. The tasks and purposes are pretty much the same, and to make an explicit distinction is counter-productive. Stefik (1995) concurs in detecting what he calls the *hand-off assumption*. A strict product-oriented view of knowledge system development characterises the process as development followed by maintenance, where a completed system is handed over to a maintenance team. This maintenance team adjust and extends the system over time. He goes on to state "It is more useful to this of the process as including a structured transition from a phase where the system is undeployed and definitely under development, through a phase where the system is deployed but still undergoing important changes in response to initial experiences, to a phase where the system is undergoing constant and sometimes large-scale evolution".

### 2.4.3 Conclusions

On the one hand, there is a history of successes gained by rapid prototyping approaches. The problems experienced are the knowledge acquisition problem and the gap between prototypes and professional knowledge systems. On the other hand, there is an increase of design and structure in the methodological approaches to knowledge engineering. One question that we should ask ourselves is whether we can now build systems that are more complex. Another question that is equally valid is can we now build simple systems easier, faster and with less effort. Do current mature approaches improve upon our ability to develop knowledge systems, compared to the earliest rapid prototyping approaches?

All in all, considering early rapid prototyping approaches and current spiral methodologies, there is an increase in model-based development, and structure-preserving implementation, whereby the earlier prototype can be retained. Iterations are dispersed differently, as more attention goes to getting the design right, than getting the system right. Nevertheless, there seems to be less feedback from this approach, and the time from first rule to working system is greater than before. The appreciation that knowledge is gained through experience is not lost on these scientists, but they consider the interaction with the knowledge source to be the experience. Now the system is not thrown aside, instead the design is changed to accommodate the adjustments.

The development of knowledge systems is model or design based with the clear purpose of delivering a product, when looking at either the rapid prototyping approaches of old or the mature methodological approaches currently in vogue. Knowledge systems are approached as software products, that can be designed, constructed and maintained as software. So knowledge systems are perceived as artefacts that are designed similarly like software, and can be constructed by an iterative process and finalized. Any remaining activities such as adjusting for flaws and coping with incompleteness in the knowledge model are seen as the purpose of the maintenance.

To illustrate the limits of what the current methodologies see as an iterative development strategy, an example. A description of a realistic case on a knowledge system for nuclear reactor noise analysis featured in the last textbook on CommonKADS shows how the development iterativity 'quite typically' works (Schreiber et al. 1999). While the first phases are used to incrementally develop the design, '...[f]rom the third cycle onward, the project life cycle became rather predictable, and could be more and more managed in the waterfall form...'. This shows the proclivity to use many cycles for design and then go directly for waterfall strategies.

## 2.5 Problems and Challenges

This section examines the discussions in the previous sections to summarize the problems. The question being asked here is what are the problems facing knowledge engineering and what causes these problems. From a practical perspective, the direct economic aspects of knowledge systems play an important role. From a more scientific perspective, these practical problems are influenced by other problems intrinsic to knowledge engineering. These problems are discussed after which a number of conclusions are stated.

### 2.5.1 The Bottom-Line Problem

Knowledge systems are both commercially and practically successful, and have been said to be AI's most successful spin-off. Some knowledge systems have shown to deliver enormous benefits, for example XCON (Bachant & McDermott 1984) which has saved DEC many millions of dollars. But despite these successes, which some researchers quote to numbers above and beyond a thousand successful knowledge systems (Lenat & Feigenbaum 1989), many war stories talk of prototype knowledge systems that never got the chance to prove they could also provide such benefits (Crofts et al. 1989). Yet, there are no known knowledge systems to date that rate in the category of killer-apps, used by a great number of people.

A practical threat to knowledge system development is the high cost and risk associated with knowledge system development and a severe lack of clarity when it comes to the return of knowledge systems. The cost and risk originate in the intrinsic difficulties of knowledge system development and accidental problems exacerbating these difficulties. The benefits of such a knowledge system is often vague or incomplete. The goal of knowledge system development is to increase the effectiveness with which knowledge is used in an organisation, and people involved in these projects seem to see this as a realised benefit in many cases (Schreiber et al. 1999). But the return is in many cases only in part measurable, because the return is often found at the level of the organisational efficiency and sometimes even stated in humanitarian grounds (e.g. more interesting jobs fit for people). In addition, in the current approach the development of knowledge system, they require a considerable investment upfront.

This creates a situation of uncertainty, which is not the best basis for the development of a system as its economic benefit is the reason for its existence. In the case of XCON the lack of visible return and the cost of the system's development has led to the project being cancelled three times before reaching its current status (Bobrow, Mittal & Stefik 1986). Currently however, XCON is seen as a great cost saver. It now generates returns in excess of its initial investment. PROSPECTOR (Duda et al. 1979) helped find in 1983 a deposit of molybdenum in Washington State, and then another in Alberta worth \$100 million. In most cases the actual return of a knowledge system is more indirect, hard to measure and far more difficult to predict in the inception phases of a project.

This *bottom-line problem* prevents organisations from seeing knowledge technology as a viable approach to solve their problems. The cost, risk, and lack of a clear return cause them to label knowledge systems as experimental technology, as it does with many techniques in AI. This limits the application of knowledge systems and makes companies refrain from using knowledge systems in mission critical applications. XCON in this sense is a tremendous exception, but also one that shows what the benefits can be when that risk is taken.

This is a predicament that knowledge engineering finds hard to get out from. The problems have not merely consequences for knowledge engineering's applications, but also negatively affect the science behind it and the cognitive research that lies behind the conception of knowledge systems. Changing this situation must be one of the most important goals for knowledge engineering research, from both a commercial and a scientific point of view.

Some of problems that underlie the risk and the cost of knowledge system development have been discussed shortly in the previous sections. In the following section, these problems are summarized and discussed as to their cause.

### **2.5.2 Knowledge Acquisition and Maintenance Problem**

“Because the physical symbol hypothesis works so well it has become not one philosophy among others but an absolute viewpoint among AI workers. Thus the conventional explanation of the difficulty in acquiring knowledge is that experts don't communicate the knowledge very well.” (Compton & Jansen 1990).

The difficulties associated with knowledge acquisition and maintenance have been mentioned above. In the basic model for knowledge acquisition, a knowledge

engineer fills a role as intermediary, a scribe between an expert and a knowledge mode. The difficulties are attributed in part to the communication and the representation mismatch between human and computer representations.

Knowledge modelling itself seems to function best if approached as an iterative process, using a construct-review-feedback cycle. There seems to be no way around this inherent incremental nature. The feedback and ability to attack the defining of the knowledge by constructing and reconstructing is an essential part of the development. By perceiving the development of the knowledge model as a process of learning, the perception of the effort and tasks changes. When humans learn it is also in small increments. The piece of knowledge is integrated into our existing knowledge and makes us better at what we do and provides a stepping-stone to more knowledge in that field. It is appreciated that the majority of knowledge, and more so, the knowledge that makes us true experts is acquired through application of knowledge, through practical experience.

This also explains why the acquisition and maintenance of knowledge systems is such an arduous task, requiring a great amount of work. The nature of knowledge does not allow knowledge models to be created into existence but requires them to be rediscovered and explored as it were. This apparently is even necessary when the domain itself is static. Most of the development strategies mentioned seems to take the view that at a point in time, the knowledge system is completed and the remainder of the effort can be approached as maintenance. Studies of systems that have monitored and studied the maintenance of the knowledge models show that this maintenance changes and expands the knowledge considerably, beyond reach 50% of the total knowledge in the system a year. This indicates that contrary to the thought of maintenance as a period where flaws are removed and incompletenesses adjusted for, it constitutes the period for a knowledge system where the actual effort is only just begun.

This can be a considerable effort when this is compared to the effort of for example XCON. More than half of the thousands of rules are changed every year (Soloway, Bachant & Jensen 1987). The adjustments made to XCON are in part necessary as the environment changes. New products require new rules and adjustments and extensions to existing ones. New situations require different and novel solutions. The evolution of the environment must be reflected in such a knowledge system. Changes in that environment will also have their effect on the configuration of existing products. This will be true for all knowledge systems, and that type of infrastructure will be necessary for such systems. This also lends support to the perspective of a learning process.

The GARVAN ES-1 system shows a different principle at work (Preston, Edwards & Compton 1993). Working on a static domain, the knowledge systems rules keep expanding and evolving. This system shows proof of the principle that using knowledge changes knowledge. These are not errors that are taken out of the knowledge system, but are categorised as knowledge discovery.

Knowledge systems require a support structure that will update and maintain the knowledge system. This situation can be compared to that with databases. A database is not just created and then left to its own resources, it is maintained and kept in line with reality, in part automatically by its use. Special personnel are dedicated to the guard the quality of these systems, to ensure its consistency and quality.

In any case, the result from both knowledge acquisition and maintenance both show a problem with creating correct knowledge model, which has over time necessitates the use of iterative strategies. The assumption that the root cause of this must be found in the nature and origin of knowledge seems to be warranted. It would seem that current approaches have not paid this sufficient heed.

### 2.5.3 Gap Problem

A further problem lies in the development of user-friendly, industrial strength knowledge systems based on a knowledge model. There remains to be a great gap between a prototype and a commercial knowledge system, especially in the visualisation of knowledge systems, as compared to what users have become accustomed to in standard software (Crofts et al. 1989). This gap problem is a very serious concern for knowledge systems.

Even after a successful knowledge model has been developed, the ability to translate this model into an application acceptable to the end user is tenuous. Besides misconceptions and neo-Luddite technophobe sentiments, the users often find the level of such applications to be below par. The acceptance of knowledge systems is not easy, as they do not restrict themselves to providing a service, but effect changes in the organisation and balance of knowledge. For knowledge systems this is perhaps even more true than for normal software systems.

“It is too easy for us, the [knowledge-based systems] community, to bask in the technical ingenuity of our problem-solving solutions, and to forget that the whole system may be judged a business failure if the printer doesn’t work, or the communications software is incorrectly set up.” (Killin 1993). The quote nicely sums up what the critical nature of this gap is. Integration is one part of this equation where as the user interface is another important aspect.

User interfaces are very important in knowledge systems for generally they are highly interactive and can involve clients with little or no computer experience. The design of the interface in consultation with experts and clients is an important part of knowledge acquisition. Since most knowledge is fairly transient, knowledge bases are important aspects of system operation. The interface for those maintaining and enhancing the knowledge model also needs careful attention.

The effort to develop user-interfaces of a quality that most users have grown accustomed to and inclusion of auxiliary facilities such as integration with existing information systems can require even more work than the total effort of developing the knowledge model. This is discussed in literature (Crofts et al. 1989) but also has been observed in applications developed in practice in the context of the Knowledge Based Systems Department at TNO. The percentages seem to show that this additional development constitutes from 50% to 70% of the total development. The attention given by CommonKADS in modelling the environment as organisation and the technical environment is therefore certainly not misplaced. Usage of knowledge systems as components of larger systems is in this regard also an important aspect.

The design- or model-based approaches mentioned earlier make it difficult to build systems fast, as a long track must be completed before the users can view the system and make comments on its form and content. This has as a side-effect that knowledge systems are cancelled before they can even make clear some of the benefits that they could have, before even a single user has seen it. The term ‘time-to-market’ is perhaps



seen eyed with suspicion as it may seem only a commercial concern of developers, but it is nevertheless one of the important factors determining the potential success of failure of any software system. Moreover, as will be discussed later it has consequences beyond the commercial.

This lack of acceptance and the long period before a first system or new increments are complete complicates the realisation of a learning process, as it increases the distance between the model on which the system is based, and the actual experiences that might be gained by the application of that model. This is in addition to the problems that arise from the dated knowledge when maintenance is not performed continuously that affects the quality of the system and therefore the use of the system.

#### 2.5.4 Brittleness Problem

The brittleness problem is a further challenge in the development of knowledge systems. A knowledge system performs badly when confronted with problems just outside of its domain, and does not have the knowledge to handle. The knowledge precipice means that the system performs well enough on the plateau, but very badly just outside of that. Some think this is caused by lacking depth in the knowledge that these systems have, others consider the missing part to be a large store of common knowledge.

The brittleness of knowledge systems, as explained as the fact that a knowledge system's performance drops dramatically when confronted with problems outside of the normal domain of the system. This is seen as a difficult problem, for a knowledge system as application but also for the cognitive questions this problem raises. This problem does not affect the economics of knowledge system development in an important way.

This brittleness displays the way knowledge systems are built, they contain sets of heuristic connections that allow conclusions to be reached based on known and inferred values. For example, if the Gram stain of an organism is negative then the bacteria causing the disease are encephalitis. This rule shows a bit of compiled knowledge. There are underlying causes why a stain is evidence for that type of bacteria. Nevertheless, the causal connection displayed here enables a medical diagnostic system to operate quite well. However, if the stain would be of another type or of an unknown type, a system employing these kinds of heuristic classifications (Clancey 1985) would not be able to dig deeper.

One of the solutions to the brittleness is the creation of deep models of knowledge. This was already discussed in relation to second-generation systems. These systems get extensive models of their domain, going beyond the heuristic association. This enables them to fall back on such reasoning when necessary, and further allows them to explain their inferences in more detail. An extension of this solution is to create a large base of common sense knowledge. During a lifetime, a great deal of knowledge is acquired, to an overwhelming degree through hands-on experience on different fields. At many points in life, this exercises our commonsense. Therefore, through our application on many different fields do we acquire deep knowledge, grounded in large amounts of common-sense knowledge that connect the different areas of expertise.

A variation on other reuse strategies, the CYC project aims to provide a knowledge base filled with all things both trivial and important that can be used as common sense. CYC is an approach to enable this deep reasoning, by providing a vast

knowledge base of facts and relations about all kinds of conceivable common sense reasoning (Guha & Lenat 1990). For example the statement ‘when it rains wear an umbrella’ implies that ‘the performer is not dead’ or that ‘the performer is human’ (Lenat 1998). This is a form of default reasoning seen earlier in frame systems. Knowledge was entered by “handcrafting and spoon-feeding CYC with a couple of million important facts and rules of thumb.” By virtue of this knowledge, any system could access this knowledge base and assist the system in understanding certain things, proving a broader basis for action and understanding.

Commonsense knowledge is created and required when one system crosses from one domain into another. Then such overlapping knowledge becomes more important. Therefore, this knowledge is learned as well and in this way. Could we function without this knowledge? On narrow domains, we probably can, and this is shown by many expert systems and by systems currently competing in Turing tests. Therefore, as knowledge systems start to cross domain borders, the need for such knowledge may become indeed crucial. Even then, it is added because it is required, not because it is a requirement for all systems.

Thus, for symbolic approaches to knowledge engineering the brittleness of knowledge systems is seen as a problem, which it aims to solve by constructing deep reasoning models and/or immense caches of knowledge containing the largest amount of trivia ever compiled. From a situated approach, one could safely say that it is a non-problem, as the environment to which the knowledge system is coupled may indeed require it to delve into issues previously not part of the systems range. This indicates a reason to change and add the missing knowledge, to let the system rise to the occasion later.

### 2.5.5 Complexity Problem

It was already said that knowledge systems are built to have “capabilities beyond those of conventional software” (Crofts et al. 1989). Knowledge systems operate on problems and support tasks that would require humans to have considerable training and experience to perform. Taken a different way, this means that knowledge systems beyond the problems already mentioned, fall prey to the fact that in many cases the problems they are required to support are complex.

The complexity may surface in different ways, but there is not a set solution to be followed for the system, otherwise a knowledge system would not be required. The knowledge in many cases is incomplete or susceptible to change. It may not be possible to make clear-cut or consistent choices. This may even be due to matter that are not easily captured in fixed structures, such as company policy or political considerations.

The problem may also have special requirements on what constitutes a good solution. For example, medical diagnostic systems such as MYCIN need to provide guarantees as to the quality of their diagnosis and cannot be seen to make lethal mistakes. A system to support that task with a 80% hit rate may be acceptable in some situations, but not in these situation. Other such requirements, such as the timeliness of the solution, further complicate an already difficult job.

In addition, the fact that these system cross a border in tasking on roles normally reserved for humans, makes the need to tread carefully in developing the system and its presentation to its users all the more important. Furthermore, because of the nature

of the problems that these systems support the system are often one of a kind, with idiosyncratic issues and project specific aspects. This adds to the difficulty already experienced.

### 2.5.6 Conclusions

The economic and fundamental problems play havoc on the field of knowledge engineering, which until now has not been able to truly cash in on its success and enter the mainstream of software development. Each of the fundamental problems plays its part in the economic aspect of the problem. The main culprits are the knowledge acquisition and maintenance bottleneck, and the gap between a successful knowledge model and a professional, industrial strength knowledge system. These two problems define the majority of the risk and cost associated with knowledge system development.

The approaches found in knowledge engineering literature show that most of the attention to these problems has been at the level of methodological solutions. This is concentrated on attacking the economic problem directly. This has led to solutions primarily directed to the method of knowledge engineering. By use of proper methodology and design, the problems may not be solved, but can be rendered controllable and predictable. It allows spreading the cost and risk, and thereby creates at least more clarity in the investment side of knowledge engineering.

Approaches to solving the fundamental problems have been singularly unsuccessful in reaching any results but to rein them in. The knowledge acquisition and maintenance problem is treated as an intrinsic, even defining, part of knowledge engineering. Problems with translating a knowledge model to a successful system are seen as accidental, not the responsibility of the science of knowledge engineering but that of the tools builders.

The conclusion is that the usage of the symbolic model of cognition combined with the product-based engineering view on the development of knowledge systems are complementary in their needs and qualities, but both run counter to dynamic nature of knowledge and its origin in learning. Current approaches to dealing with these problems are directed towards overcoming this character rather than indulging it. They orient themselves towards solutions of a pure engineering character, by employing schemes though risk-driven iterative methodologies.

## 2.6 Summary and Analysis

This section summarises the different aspects of knowledge engineering that were presented. This shows the ingrained nature of the use of the symbolic theory of cognition and the engineering view of knowledge system development. The problems of knowledge system development are however not solved by current approaches stemming from this background. This section therefore proposes an alternative direction based on this analysis of the problems.

### 2.6.1 State-of-the-art

The view given by the previous shows a current set of best practices that combined a theory of cognition, an architecture, a series of knowledge representation techniques and a process for developing a knowledge system. Two main assumptions lie behind this, first the physical symbol systems theory of cognition, second trust in engineering

as a provider of trustworthiness, efficiency, effectiveness and professionalism. These two assumptions determine much of what is now the state-of-the-art in knowledge engineering.

The basic architecture of the earliest and first-generation knowledge systems was constructed for change, compartmentalising the changeable part. The architecture used for many of the current second-generation systems is no longer based on this division, but is created per project, much like conventional software. Ever more like software engineering, each of these systems is created based on a knowledge level model, independent of the implementation platform. This further distances the design from the realisation.

Analysis of the knowledge representation techniques that are used show that the techniques used are well suited to building knowledge systems. They provide support for creating and developing different models, and formalising concepts and ideas. Support for changing the knowledge represented in these different forms is limited, and in many cases they do not scale well. Some examples of alternatives that do provide this are given but these are outside of the mainstream. The representations allow formulation of knowledge and really only allow for intermittent removal of errors, not large scale change and evolution. The representations are also deficient in the support for the discovery of new knowledge. The focus clearly is on modelling an existing, static situation, rather than supporting a quest for improved understanding.

An additional complication here is the use of a singular knowledge representation technique forces translation of concepts as seen by the expert into an interpretation in that one format. This forces the use of a professional proficient in that representation, and excludes direct involvement of domain experts in discussing and improving the knowledge model.

Knowledge engineering approaches show a progression from ad-hoc and rapid-prototyping towards methodological sound procedure. This coincides with the move towards second-generation systems. In part, this move was pushed through a crisis in knowledge system development. On the other hand, developments in software engineering already had taken this path, stressing methodological control and specification of software using design. The field of software development had gone through a period of crisis too, and had taken the ideals from other engineering fields to improve the product that it delivered and the process used to develop it to change into software engineering (Endres 1996). Much faith is placed in ability of methodological advances to solve the problems that are experienced.

The current state-of-the-art in knowledge engineering has seen limited success in solving the current problems, and subsequently has failed to make significant inroads into mainstream software development. The main practical problem is the precarious and unclear economic situation concerning the development of knowledge systems. This problem hinders the use of knowledge systems as viable technology for many organisations. This problem is caused mainly by four fundamental problems in knowledge system development, namely the knowledge acquisition and maintenance problem, the gap between a successful knowledge model and a professional industrial strength knowledge system, the brittleness of knowledge systems and the intrinsic difficulty of many of the tasks they are employed for.

The brunt of the effort to overcoming these difficulties in knowledge engineering addresses the practical economic problem. The measures are aimed to improve the

bottom-line by reducing the cost and limiting the risk. These measures derive from approaches to solve similar problems in other engineering disciplines. Introduction of prefabricated models and stressing reuse of knowledge aims to cut cost of modelling. Skilled professionals produce the models and system effectively and efficiently. Improved development methodology aims to make the development more standardised, transparent and most of all, manageable. Risk management strategies, such as risk-driven incremental development, limit the effects of the risks that are experienced. These measures are in part inspired by the physical symbol systems hypothesis, and partly by engineering disciplines, especially software engineering.

Coming to a final identification of the problem, it would seem that the total of assumptions that underlie the current state-of-the-art in knowledge engineering have been held for fact for too long. But rather than faulting the researchers involved in ardently trying to further the science and practice of knowledge engineering, the problem seems not to lie with one single element, or with the combination of each of the individual approaches.

Rather, the assumptions underlying knowledge engineering that inspire the current explanations of the problems and the solutions derived from them are at fault. Knowledge is treated as alike to the natural laws of the universe. It is absolute, precise and unfaltering. Once jotted down, the can remain much the same. This is not an explicit statement on the part of the professionals but stems from their reliance on the symbolic theory of cognition, which defaults to such assumptions. A moving target also falls outside the parameters of most engineering, which tries to fix as much as possible.

When knowledge system development is treated as the development of an artefact, for example a building, as a design based engineering approach, then these assumptions come as natural. The development of a building, where an architect spends many hours creating designs and drawings, talking with the client, introducing vision and using models to strengthen the conviction of the client that this is indeed that what is required. Building such a house, introducing technical finesse and other realistic concerns, leads to a finalised product that can be kept in a good state of repair for years to come. This is how knowledge system development is treated today. Of course, new additions can be made to the house. Making changes or creating a framework where additions can be made over time is also possible, as far as specific types of change can be predicted. However, prediction requires knowledge of the future and such knowledge is in many cases in short supply during the design phases of a knowledge system. Perhaps, the development of a knowledge system is not comparable to the development of other engineering products.

### 2.6.2 Alternative Directions

An alternative perspective is provided by the situated cognition which combined with a de-emphasising of the engineering gives a view of knowledge system development more akin to the kind of development found in science. The development of a model as a current best theory on knowledge is approached as a never-ending exploration and search for more insight. The scientific theory is a vehicle for clarifying the inner working of a domain as much as it provides a basis for products as embodiments of that theory. “Constructing a knowledge [model] is often a scientific effort of empirically constructing and testing models” (Clancey 1997, pg. 39).

The direction that a solution according to this perspective must therefore be sought is an approach that allows the nature and origin of knowledge as dynamic and open-

ended to be indulged rather than overcome. In other words, knowledge engineering must be made into a learning process. It is not possible to transpose knowledge out of one knowledge model into another one, without changing the meaning of that knowledge. Therefore such an act creates new knowledge in a sense unrelated to the previous incarnation. The consequence of this is that the assumption that any knowledge existed a priori must be relinquished, and that every new knowledge must be learnt anew. A part of this can be finding a brand new form for tacit knowledge based on the bias already present in the explicit knowledge at that point.

Perceiving learning as a requirement for the creation of any knowledge model and placing this at the forefront of any approach to knowledge engineering created new possibilities. For one, the possibilities to exploit this character of knowledge are opened up rather than solely constituting an opposing force to be overcome.

According to Menzies (1998, 1996a) weak situation cognition means a knowledge system development methodology must focus on how knowledge models are built and changed. This opening move perhaps is in the right direction but not strong enough. Knowledge engineering must be seen as steering and managing a knowledge acquisition process, with the occasional release of knowledge system as a product, but the actual product in the making will be an evolving model of knowledge. The knowledge system then becomes only the medium by which this model is made available to its users. The focus must be on making knowledge engineering a process of insight, reformulation, change and application leading to further insights. Knowledge engineering must be inspired by another kind of metaphor, towards the discovery of new insight. This should be the global direction for a solution to the problems, at all levels in knowledge engineering.

While this involves some major adjustments, these adjustments are start in adopting a different perspective. Many of the techniques that are employed to create models of knowledge are in principle sound, but they may have to be employed in other ways. This may well have consequences for current research approaches into reuse based strategies and design based approaches. Beyond discussing how these models can be called forth to bolster the productive in constructing knowledge models, it is also necessary to examine how such initial structures and designs can be adapted and evolved to follow the change of the tide.

Further adjustments are necessary in the methodologies used, to strengthen their learning character, in the roles that are played by the different people involved in knowledge system development and the tools that are employed. This to allow them to not only express knowledge but also change these models over time, reach insights into the modelled knowledge and develop professional applications with them.

The situated nature of knowledge is only a possible explanation for the problems that are currently experienced. Accepting it entails a different outlook on knowledge systems and a different approach to knowledge engineering. To differentiate between these two perspectives it is necessary to be able to verify the correctness of the proposed explanation of the problems. To do that it is necessary to ascertain whether an approach to knowledge engineering based on insight, learning, and science enables the circumvention of these problems.

## 2.7 Conclusions

This chapter realises the first step in the program from the first chapter: identify the problem and provide a direction for a solution. In the previous sections of this chapter, the field of knowledge engineering was reviewed, discussing cognitive models, elaborating the properties of knowledge systems, examining knowledge representation techniques and exploring approaches for knowledge system development. In part, this enabled a discussion of the problems in knowledge engineering, and supported the analysis leading to an explanation of these problems. In addition, alternative views were voiced to prepare the formulation of a direction for a solution to these problems.

The main culprit behind the problems that are experience is seen to be the reliance on an engineering metaphor to inform and inspire the field of knowledge engineering. This causes knowledge engineering professionals and scientists to continue to employ approaches from software engineering and other engineering field to solve the problems to do with the bottom-line of knowledge engineering. The underlying causes are fundamental difficulties with the nature and origin of knowledge that disallow approach that attempt to create knowledge into existence. Therefore, a solution is proposed that aims to augment the accomplishments of the engineering metaphor with those inspired by a scientific metaphor, to make of knowledge engineering a learning process.

# Chapter 3

## Towards Continuous Knowledge Engineering

---

*One cannot attain the limits of craftsmanship,  
And there is no craftsman who acquires his total mastery.*

*– Ptahhotep, c. 2350 BC*

The previous chapter identified the problems of knowledge engineering and targeted the underlying engineering metaphor as a common thread in these problems. Instead, it offered an insight discovery based on science metaphor as a suitable replacement. This chapter describes the continuous knowledge engineering approach inspired by that metaphor.

The first section covers the use of metaphors, and its role in knowledge engineering. Section two gives an outline of the proposed solution showing how it aims to generate and support the discovery of insight, by proposing a number of guiding principles. The third section describes the path from this global view to a number of realisable measures. It asks what the consequences are for different aspects of knowledge engineering. Continuous knowledge engineering changes the roles of people involved, the view on the character of the project, the generated products, and the manner in which the development is structured and managed. This leads to a number of requirements for the development tools, to support in making these changes practical and viable. Following in the fourth section, techniques and technologies are described that can be used to implement these changes at the tool level. The fifth section describes an evaluation program to enable a validation of the method through a demonstration in a number of case studies. The last section summarises the subjects in this chapter, paying special attention to the elements that will be the basis of the next chapters.

### 3.1 Metaphors

Science uses metaphors frequently to facilitate explanation of new concepts or even steer the incursion into new territory. This section discusses the role of metaphors in knowledge engineering and explains the difference between the engineering metaphor and the scientific metaphor for knowledge engineering.

#### 3.1.1 Role of Metaphors

In a metaphor such as ‘the world’s a stage’, the statement modifies the normal meaning of ‘world’. The metaphor focuses the attention on certain prototypical aspects of the modifier and projects them onto the subject. This concentrates on their intrinsic similarity or asserts that a similarity exists. By the juxtaposition of concepts a metaphor conveys a shared body of knowledge by adjusting the meaning normally associated with the words. This facilitates the rapid transfer of information. Perhaps for this reason, humans are quite adept at using analogy and metaphor, expressing themselves in sentences that convey more than their literal meaning. This implies a strong, shared context.



In general, the power of metaphors is that they are direct and can be grasped as conceptual wholes. The unspoken implications of the analogy facilitate explanation of ideas as a whole. By application to known elements to create a they focus on a particular aspect, such as 'men are pigs'. They can function as an analytical tool to bring certain important aspects to the fore, by a simple modification of an original subject.

A metaphor also allows new unknown ideas to be expressed, like a 'horse-less carriage', explained in terms that are known. This assists in the communication and helps understanding such new ideas. In fact, in this role, metaphors are also used to structure and inform new approaches, as is evidenced by the early beginnings of the development of flight. The metaphor of the bird inspired the first flying machines, although now there is only a superficial semblance between airplanes and birds. In the beginning however, different inventors and scientists would not hear of any attempt of flight that did not share significant features with birds. The inspiration of the bird was an important factor in the starting the eventual development of flight, by providing preliminary answers to questions that were unanswerable at the time.

In software engineering and knowledge engineering alike, metaphors have also been used to make sense of areas that are little known and have only tenuous connections to concepts familiar to the normal physical world. The role of metaphor in this context has been subject of many discussions. "A software metaphor is more like a searchlight than a roadmap. It does not tell you where to find the answer; it tells you how to look for it. A metaphor serves more as a heuristic than it does as an algorithm." Software engineering for one has been compared to penmanship, building, farming and oyster farming (McConnell 1993). Even the term software is a metaphor (Bryant 2000a).

The strength of a metaphor depends on a number of things. First, there is the number of features that the metaphor shares with the actual subject. Secondly, a metaphor can be especially appealing. Finally, merely repeating a metaphor can turn it into a self-fulfilling prophecy. This further depends on the measure that a metaphor can support the rallying of people under a common flag. The strongest metaphors are those that relate well-known things to things that are relatively unknown. These allow making immediate jumps towards new insights.

Sometimes areas of inquiry are proposed that turn out to have been misleading, in which case the metaphor has been overextended. Of course, a metaphor should not be treated as complete literal truth, as without attachment to reality it soon acquires the features of a myth. This can occur by mere repetition or appeal. Metaphors can also start to lead a life of their own where their function as a guide into the unknown is starting to be taken literal, evolving to ever-greater rigidity (LaFrance 1997). A metaphor used for a long time can seem like truth, familiar as it is in connotations and implications.

Therefore, while a metaphor can be a strong tool, and provides for easy communication, focus on important aspects, and informs approaches into the unknown, they also pose a danger. The momentum associated with a metaphor as inspiration for an approach can create a long lasting allegiance to the metaphor. This may lead a metaphor to be followed well after its time is up.

The history of science can be described as a series of substitutions of one metaphor with another one (Kuhn 1970). This substitution is not one from wrong to right, but changes from 'worse' metaphors to 'better' ones, towards more inclusive and

suggestive ones. In fact, many models that have been replaced by better models are still useful. Engineers still solve most engineering problems by using Newtonian mechanics even though, theoretically, Newtonian dynamics have been supplanted by Einsteinian theory (McConnell 1993).

### 3.1.2 Engineering Metaphor

Artificial intelligence relies heavily on metaphors, both to explain the conceptual approach underlying different techniques and to inspire new approaches to computational intelligence. Metaphors are productive because many concepts in AI are hard to explain and understand. Often based on biological counterparts, like neural networks, artificial immune systems, genetic algorithms, etc. (for example, see (Timmis 2000)), it provides a grounding for such new approaches.

Knowledge engineering also is fraught with the use of metaphor, because the terms used in their pure sense mean so little to others. Even the term itself contains a metaphor with the engineering known from other disciplines. Knowledge acquisition has been likened to mining (Hayes-Roth et al. 1983), knowledge system development equated to the crafting of a ship (Waters & Nielsen 1988) and a knowledge system compared to an expert in a box or on a disk (Laurie 1992).

Engineering is designing and making artefacts to have desired properties (Simon 1982). The design entails a detailed specification of an artefact that conforms to the specification in a justified manner, including matters of economics and practicality. Sound engineering principles mean to ensure that the specifications are correct and that the development of the artefact based on the specification remains to be justified and of good quality.

In engineering fields such as structural engineering, a thorough analysis of the intended project leads to the creation of a detailed design. This complete design is a blueprint that specifies every nut, bolt, dimension and material for every part of the artefact to be developed, including detailed projections of the entire construction process. These blueprints are created before one nut is attached to a single bolt, and are considered fixed throughout the realisation of the plan. This way of creating an artefact based on detailed specifications is not amenable to dealing with change, or support for discovering the actual specifications. The process of creating such a detailed specification is a very lengthy and expensive process, and can consume as much (or more) time and money as the actual construction of the system.

Furthermore, the work towards this artefact is finite, and is performed by professionals that are specialists in their engineering field and have knowledge of these engineering principles. Different specialists may be involved in different stages of the project. The process of development can be incremental in approach, and apply risk-driven methodologies, but this is a measure solely aimed at dividing the work in controllable parts and compartmentalisation of the work that is done. Verification of the products quality is in relation to the specification, as its objective measure of quality.

What this shows is an engineering practice that is skilled at creation, but ill-equipped to manage change. In the meantime, the world turns, and directions and needs may change. The engineering approach is therefore less suited to mutable specifications, and changing conditions in the world. It most certainly does not fare well when the

operations used to map requirements onto specifications that were good yesterday, are bad tomorrow.

In the previous chapter, different aspects of the practice and science of knowledge engineering were discussed. The conclusion was reached that there is an engineering metaphor that underlies the current state-of-the-art in knowledge engineering. The cognitive models, architecture of current systems, aspects of the knowledge representation techniques and the view of the process of knowledge engineering all clearly have this aspect. There are many shortcomings in using the engineering metaphor to develop the methodology for knowledge system development. While some of the problems may be perceived as intrinsic or peripheral, within the view provided by the metaphor, they change in character.

The term knowledge engineering combines scientific, technological and methodological elements (Hayes-Roth, Waterman & Lenat 1989) derived from software engineering. Software engineering has mimicked many of the attributes of other engineering fields such as structural engineering (Endres 1996). Even software engineering is experiencing the backlash of adopting pure engineering measures to an artefact that is not analogous in form, function and development needs (Fowler 2000, Bryant 2000a, 2000b). This has led to several lightweight methodologies that aim to follow the change in requirements, and use it to improve the product, rather than resist it and force development in a preset direction, e.g. DSDM (Dynamic System Development Method) described in (Stapleton 1997), Extreme Programming (Jeffries, Anderson & Hendrickson 2001), etc.

Differences in the character of a knowledge system when compared to other man-made artefacts show it up-to-now to be far less clear what the requirements are. This makes it hard to describe them upfront, and likely to be far more susceptible to change. Within the engineering metaphor view, knowledge is treated as a static commodity that needs to be used as a semi-manufactured product. The engineering metaphor's interpretation of a knowledge system starts to break down, and measures to attack the problems in knowledge engineering that are solely inspired by engineering metaphor and associated thoughts, will not be guaranteed to solve these problems. The shift away from the engineering metaphor is rendered difficult because of the pervasive assumptions regarding the nature of engineering and the role of the engineer.

### 3.1.3 Scientific Metaphor

In a simple view of science, scientific models formulate experiences and observations about the world. These models are valuable because they describe observed patterns and predict future phenomena in detail. Furthermore, they have engineering value, enabling us to devise artefacts with known attributes. Therefore, an important aspect of a scientific theory is to allow development of the theory, permitting inspection, questioning and changing of these scientific models. The value of a theory is not simply in how well it corresponds to past practice, but in how well it serves to guide and develop future practice.

Learning is changing ones behaviour based on previous episodes to improve performance on later episodes. It is considered an important aspect of intelligence, and some it equate with intelligence. Much of what humans do is arranged to create opportunities to learn. Science can be seen as a human invention to support learning, and catch faults the justification of our actions based on current understanding.

Science learns by modifying theories that try to explain the world around us, and that are tested against the environment. A change in theory allows new predictions to be made, leading to new tests to verify it further and drive its development, in a form of co-evolution. While some tests lead to minor change, like adding an additional decimal to some physical constant, the most important ones advances are made through the discovery of a new insight. Testable theories are required because they incorporate learning opportunities explicitly, at the current boundary of knowing and predicting.

**in-sight** (in'sit') n.

1. clear or deep perception of a situation
2. a feeling of understanding
3. the clear (and often sudden) understanding of a complex situation
4. grasping the inner nature of things intuitively

**Figure 3-1 Dictionary Entry for Insight (WordNet 1997)**

The word insight is almost strange in this context and has connotations that are related more to religion than to science. An insight is the feeling, the process and the result of coming to understand something. Insight provides a consistent and complete view of a system from seeing only some of the parts. It constitutes a moment of clarity, as a sudden spark of immediate comprehension, recognising the inner consistency of things. Insight is one of the fundamental building blocks in the development of human understanding of its surroundings and itself. Scientific progress is dependent on new insights to originate new avenues to explore, to create the very stepping-stones for generations to come.

A new insight then means discovering new abilities and understanding, a deepening of knowledge. Perhaps more even than undirected creativity, a sudden insight can provide many new lessons. Learning continues from recognition of events, of realisations of some structure or pattern. Those expectations realised and those failed, from the perception of anomalies and incongruities, of exceptions and serendipitous events. The discovery and generation of insight requires a continuous process of successive adjustments and critical feedback (Kuhn 1970).

This can be taken to mean that a scientific theory only provides for insight discovery when it is falsified or justified. This view does not explain why theories are sometimes supplanted by others theories identical in predictive strength, but considered more elegant than its predecessor. Rather the theory that provides most opportunities to move forward, to gain additional insight will be favoured. Every phase in the development of a scientific theory, not merely the moment of falsification can give rise to new insights. Opportunities to discover new insights are present throughout. Multiple viewpoints on the same material can clarify inconsistencies.

Scientific models thrive through application of their ideas, both to justify and falsify their content as well as to derive practical value from them. A scientist examines the results of these applications and is able to verify the scientific theory used. It can be adjusted when predictions made by the theory do not fit reality. When sufficient adjustments are needed, a restructuring of the model may be necessary. From the failures of the model as well as the subsequent adjustments and restructuring, a new

insight will spring forth that must be incorporated and justifies the restructuring as well as give it its future direction. In this way, old knowledge gives way to the new.

## 3.2 What is Continuous Knowledge Engineering?

In the previous section the scientific metaphor and the notion of insight discovery was deepened. From this continuous knowledge engineering is posited as an alternative approach to knowledge engineering inspired by this metaphor, based on a number of guiding principles.

### 3.2.1 Interpreting the Metaphor

Since applying the scientific metaphor to knowledge system development implies several things at once, this section attempts to unravel some of the more important aspects. At the same time, some of the intrinsic aspects and requirements of knowledge system development are addressed.

Science is gathering knowledge, derived from observation, study, experimentation and the development of artefacts with the intent of determining the nature or principles of the subject that is studied. It is a continuous process of successive adjustments and critical feedback (Kuhn 1970). The purpose is to increase insight and understanding of the world that surrounds us in the form of a theory, that others to its veracity can inspect and provides a basis for further improvement. From this understanding, artefacts based on this understanding are created, for example through engineering. Beyond the practical use of these applications, this can also be a source of new insight, feeding into the theory.

Applying the scientific metaphor to the development of knowledge systems changes the perspective from devising an artefact, to facilitating the development of deeper understanding and putting this into a model that can be applied. In this view, knowledge system development is as a learning process, starting from nothing. The knowledge model is made from scratch, with knowledge discovered through insight, deriving from working with and applying the models. To embrace the character of knowledge, the development of knowledge system therefore needs to be arranged as learning a domain anew, grow from nothing to competence, and then maintain this level of ability.

In effect, this means turning knowledge system development into sustained knowledge acquisition. It makes knowledge engineering a process of learning, by supporting the ability to discover new aspects of knowledge and model a changing corpus of knowledge. The insight discovery aspect of the scientific metaphor makes knowledge engineering a process of successive change and application with the intent on generating feedback for further changes. Knowledge in this view is not excavated, captured, courted, or created, but discovered. This involves the development of knowledge models which, if they are to explain inconsistencies which exist with respect to earlier knowledge models, must go beyond existing knowledge and therefore require a leap of the imagination. With every change, a possibility to learn more is created, and provides for the discovery of new insights. Every failed test can provide new knowledge.

Voices from the past also go towards what is being expressed in this chapter. Bachant and McDermott (1984) argue that an “expert system will never have all the knowledge it needs and that it is essential to introduce a system into routine use in

order to uncover the inadequacies in its knowledge”. Lenat and Feigenbaum come to similar conclusions: “...standard software design methodology cannot build a program ‘in one pass’ to perform the task. However, as the developing [knowledge] system makes mistakes, the experts can correct them, and those corrections incrementally accrete the bulk of the hitherto unexplicated rules. In this manner the system incrementally approaches competence and even expertise” (Lenat & Feigenbaum 1989). These statements were made before the second wave in knowledge engineering was felt.

The consequences from the metaphor also influence the perception of the knowledge system itself, changing to a mere application of modelled knowledge. In this view, a knowledge system is more properly divided into the knowledge model and the knowledge system that applies the knowledge. The knowledge model is the abstract, formalised, current state of knowledge on the domain, as an operational theory. The knowledge system allows the application of that knowledge model; the hypotheses and assumptions contained in the knowledge tested each time the system is used. Any comment on the system, immediately is a comment on the knowledge model and an opportunity to learn and move forward. Furthermore, the knowledge system then also has two kinds of users, the system-user who apply the knowledge to a task using the knowledge system, and the model-users who use the knowledge system to further their knowledge and improve the knowledge model. This requires the knowledge model to be both operational and remain comprehensible.

### 3.2.2 Principles of Continuous Knowledge Engineering

The alternative approach to knowledge engineering proposed in this chapter concentrates on continuity, which it perceives as the essence of this interpretation of the scientific metaphor. Full understanding cannot be attained at once. From inception to decommissioning, a knowledge system should be successively adjusted and reviewed critically. Each successive adjustment should accommodate the latest insights. These changes should be open to critical review, both from the side of the model as well as the application. Therefore, from early on a model should be created and made operational in a knowledge system. Verification and testing can then be used to make sure knowledge does not get lost rather than validate the knowledge, although a use it or lose it principle does apply.

While the metaphor inspires this approach, the needs and requirements of developing a usable knowledge system in an economic fashion, to a number of desired properties also plays an important role. To make the approach viable and practical, an attempt is made to use current techniques and technologies but within an alternative perspective. The concept of continuous knowledge engineering combines some of the notions from scientific method and other learning systems with the current understanding of knowledge engineering and software engineering.

Continuous knowledge engineering is a common set of ideas that facilitate a learning process. It involves a change of perspective on the deployment of current methodologies, tools and techniques. The goal of this approach is to change the conceptualisation of knowledge engineering from a constructive process with a fixed end goal, to one of continuous development. The intent is not to replace current practices, but to augment them and deploy them in a different way.

### 3.2.3 Cyclic Development

Engineering often applies divide-and-conquer techniques on tasks by splitting them up into meaningful parts and planning with these smaller elements. Different approaches exist to spread the effort in any project over several different phases, each with its own timeframe and goals. The basic models of software development are code-and-fix, waterfall, transformation, evolutionary, and spiral models of development (Boehm 1988). The methods differ in the way that they divide the work and order the different kinds of steps.

The essence of the approaches can be divided into *incremental* and *iterative*. Incremental approaches divide the task into smaller sections that can be seen as independent additions, e.g. waterfall approaches. Iterative schemas go through all the motions each iteration, e.g. evolutionary approaches. In the latter category, the difference is in the size of each iteration. The spiral model of development is an approach that mimics the other models, using a risk driven approach and based on the dynamism of the situation.

Iterative development means that each step is based on a previous step. Lessons learned in the current iteration better enable and inform the tasks in next ones. This is the reason why many so-called evolutionary or incremental software engineering methodologies favour it to enable risk- or functionality driven approaches (Jacobson, Booch & Rumbaugh 1998, Stapleton 1997). As was discussed in the previous there is already a tendency to employ iterative strategies in other approaches to knowledge engineering. However, in these approaches the iteration mainly takes place in the design phases, and not through cycles of implementation, application and design.

The principle of cyclic development means each of the changes made is translated to the knowledge model, and immediately deployed into the knowledge system. The term *cyclic* is introduced to denote approaches that are capable of running through all the motions for each single change, leading to many, very small increments. In many situations a solution to a problem cannot be created, but must be revealed through experimentation and exploration. In addition, many solutions are formed in reaction to practical problems. These problems give focus and direct the development of knowledge through experience. Furthermore, only when the knowledge is made explicit can certain epiphanies take place, clarifying and illuminating old problems in a new light. Only by positioning the knowledge models in the real world, where they are confronted with real problems and real solutions can we be assured of the quality of that model and gain experience leading to novel insights. Anything less than this may provide a candidate with which we can delude ourselves. These statements hark back to the grounded program defined by Brooks (Brooks 1991a).

The ability to incrementally develop and ripen a knowledge model to increasingly incorporate new knowledge and improve on the knowledge, as it is already present in the system is the pivotal element in the approach. Knowledge builds on knowledge, like in real-life. True insights can only be gained working and testing our incomplete models of the world and discovering imperfections in its understandings. The focus of continuous knowledge engineering is therefore on fast and frequent delivery of the knowledge system. In this way, the approach becomes based on the knowledge model and the knowledge system, rather than on requirements or models. This is more flexible as the system does not have to constitute a complete solution, as long as it demonstrate evolution and can be used a basis for evaluation of the progress and

direction of the development. To converge on an accurate solution it is necessary to use iterative and incremental development strategy.

To work with confidence in a cyclic strategy it is important that changes that are made can be reversed. It must be possible to accept sometimes that a wrong path has been taken, and that it is necessary to return to an earlier point in the development. In some cases, this may lead to a concern that much work is discarded. The safety of being able to return to an earlier point in the process creates a freedom of movement during development that encourages experimentation, exploration and discovery. In addition, because of the relative regularity of the increments, the size of the setback may be small. This in particular will ensure that only recent work needs redoing.

### 3.2.4 Active Participation

The principle of active participation means that the community of stakeholders, i.e. the experts and users, are involved directly in the development of the knowledge model. Therefore, it moves away from development solely by trained professionals, such as knowledge engineers are. Rather than transfer the domain knowledge to a knowledge engineer, it transfers knowledge engineering knowledge to the modellers and users.

Active participation focuses on getting the experts to be actively involved in modelling knowledge. This can be in unison with a knowledge engineer, but the ultimate intent is to get them working directly on the knowledge model. The wish to have experts to participate in the knowledge engineering has existed for long as it represents certain financial and practical realities (Shapiro 1987). The idea itself is as old as McCarthy's advice taker (McCarthy 1968).

Being involved in the modelling of knowledge yields additional insight in that which is modelled. The current person engaged in this role is the knowledge engineer. The practical experience of the expert should give the expert a better basis to learn and discover new lessons. It makes the expert the person best able to appreciate the possible insights to be gained from the development and application of the knowledge model. The expert is grounded in the field and brings practical experiences to bear upon the model. The expert will also gain the most benefit from interacting with the model. Making the understanding of the task and the domain explicit will aid the expert and deepen his own knowledge.

The knowledge engineer should focus on the structuring of the knowledge system and facilitating the development of the system. Furthermore, the cyclic way of working would require the knowledge engineer to be either constantly connected to this single project. This would be possible if the effort was small but regular or when the knowledge model would be large enough to warrant such singular attention. This would not constitute effective use of such high priced and rare individuals.

Knowledge modelling by the expert is not a common approach in AI. The work on RDR systems mentioned earlier comes closest. To a great degree, the experts have maintained these systems, for a few minutes each day. They were involved in all activities concerning knowledge formulation and formalisation (Compton et al. 1989; Compton & Jansen 1990, Compton et al. 1992; Preston, Edwards & Compton 1993). A similar endeavour plays a part in the Spark-Burn-Firefighter (SBF) experiment (Marques et al. 1992). The Spark system selects a task framework with the user. The Burn system then specialises the framework and places default knowledge into the



tasks of the framework, based on choices made by the user. The Firefighter then tries to acquire new knowledge from the user and locate errors to iterate further with Spark and Burn. The whole SBF framework aims to ease the task of the ‘non-programmer’ to create and specialise programs and maintain them.

### 3.2.5 Direct Application

The principle of direct application means two things: a knowledge models should be consultable at any moment in its development and the knowledge system developed around a knowledge model should be a professional, industrial strength solution.

In the first sense, this means that at any point the knowledge system must be able to be consulted, to verify the system through testing and running cases through it. The second sense is quite comparable, but directed towards development. Then direct application means that the knowledge model can be deployed to its intended users early in the development and after that with each new increment. Simply said, this means that a system based on a single rule should be both consultable and deployable with ease, and that changes to the knowledge model are published in rapid succession. Furthermore, the quality of the systems should be such that it is acceptable to its users.

One of the main priorities in development of knowledge systems is that someone will use the system – it has, at least, to meet with interface standards that end-users are expecting from traditional application programs (Crofts et al. 1989). The fitness of the knowledge system to the expectations of the users is an essential criterion for the acceptance of the knowledge system and the knowledge it contains. This kind of focus on ‘business purpose’ also features in other incremental or evolutionary development methodologies such as DSDM (Stapleton, 1997). There is no reward for delivering the best product, just for the delivering the right solution.

### 3.2.6 Practical concerns

When discussing the approach based on a scientific metaphor and detailing the principles on which it should be based, this was mostly forwarded from the concept of applying another metaphor. Other, more practical concerns also play a role. The question is whether a continuous approach to knowledge engineering is better capable to deal with the character of knowledge in real life, as well as with the practical and economic concerns of knowledge system development.

The continuous knowledge engineering approach favours easing into development. The knowledge model can be development through exploration and discovery. This supports the formulation and formalisation of knowledge. This can even be true for static domains of knowledge (cf. GARVAN ES-1 (Preston, Edwards & Compton 1993, Compton et al. 1992)). Acquisition of knowledge is eased considerably as it is spread over a longer period. It is based on needed adaptations, rather than obscure completeness goals.

As was mentioned considerable effort goes into developing the visualisation and auxiliary facilities for a knowledge systems as well as integrating the system with existing information systems or as a component of a larger system. Both of these benefit from a cyclic approach with direct application. First of all, systems can be developed without these facilities at first. Secondly, introduction of these facilities can

be done as needed, in an incremental fashion. Especially interfacing with existing systems can benefit from an exploratory approach.

The experts are more likely to remain involved and feel less threatened by the system. It also can improve the quality of the expert, as they adopt a more analytical attitude towards the domain and can become more proficient. This may provide to be a very important benefit, as the assumption that the knowledge system can exist without an expert is left behind. The users will also be more easily and more effectively trained in the use of the system. By being confronted with the system early and by getting new features in small increments, the users themselves easily grow accustomed to the system. Starting with simple, basic functionality they grow with the system as it gains new functions with time. As these functions are more likely to be based on user wishes as they have more time to propose new additions.

One of the criticisms that is aimed at most iterative and by extension cyclic development strategies by “often-unrealistic assumptions that this operational system will be able to accommodate these changes, without addressing long-range architectural considerations.” (Boehm 1988). This is an important concern, which the remainder of this thesis must address.

### 3.2.7 Conclusions

From the previous, it is clear that a scientific model for development runs counter to an engineering model, as it changes the perspective on different aspects quite drastically. From the initial realisation that knowledge system development concerns the development of an open-ended, dynamic artefact, other consequences flow automatically. Such an artefact is not created into existence; it will be developed over an indefinite time period. In that case, it requires a permanent organisation of stakeholders to guard it, rather than a temporary team of professionals. Furthermore, approaches must be found to deploy such as system into use early on and with each increment, rather than intermittently. Continuous knowledge engineering implies continuous development, continuous participation, continuous application and continuous testing. Each of these forms a high-level abstract principle completing the description of the approach.

By keeping the system small in size initially, it does not require a great investment. When the system is seen to incur a certain profit or becomes more intensively used, the organisation can use this information to determine whether they should continue the development and direct the development towards what is considered essential features. These investments are based on the benefits already seen, rather than those predicted or supposed. This solves the bottom line problem.

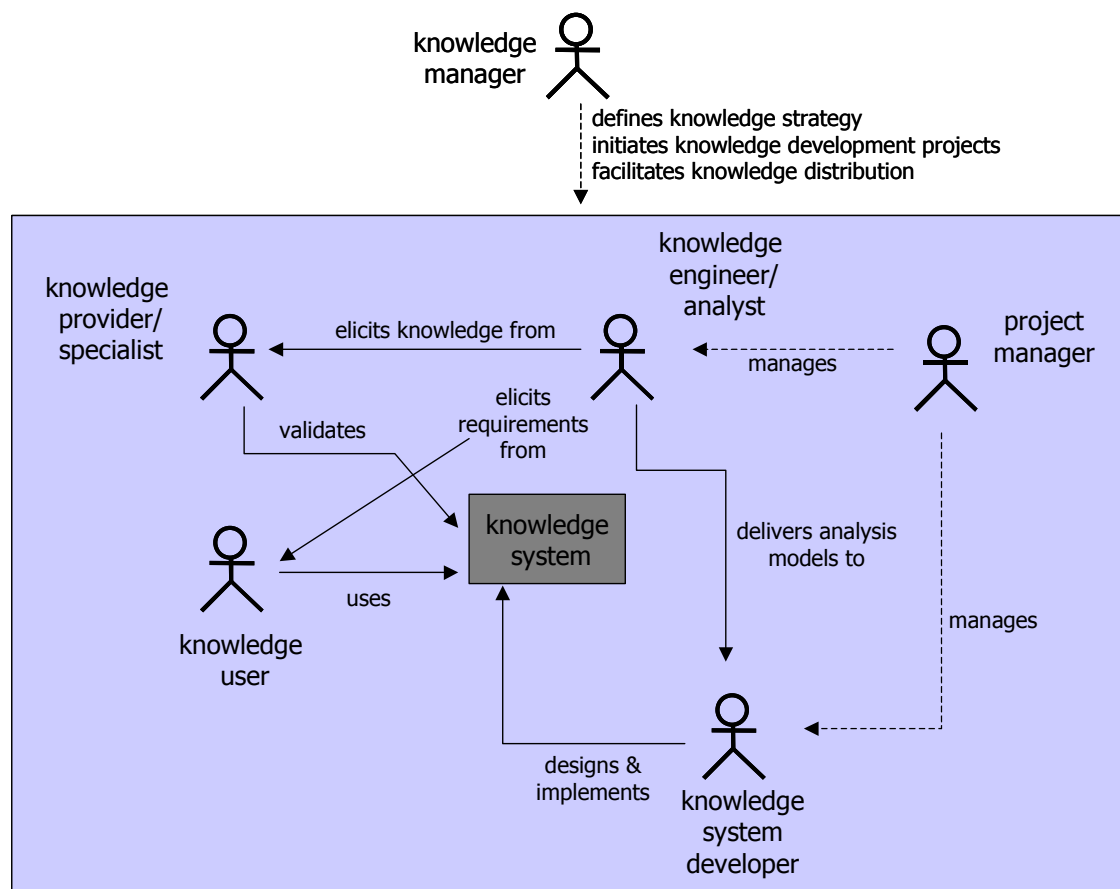
A further practical benefit is that the risk of building the wrong system is reduced. By constant testing of the products, the direction of the development is verified and the focus is on developing a system for the users. Instead of conforming to a set of formalised requirements, it focuses on added user-value. The final system is therefore more likely to conform to users expectations and therefore will be more acceptable to the users. In addition, as the system’s development is influenced by their comments, they are more likely to assert ownership, and see the system as something they have participated in.

### 3.3 From Principles to Requirements

The next question is what the effects are of these principles of cyclic development, active participation, and direct application. The continuous knowledge engineering method affects many aspects of the development of a knowledge system. The remainder of this section discusses these effects according to a categorization used by Jacobson, Booch & Rumbaugh (1998). This discusses these effects as they apply to people, project, product, and process, before going into the tools support that this approach requires.

#### 3.3.1 People

The people are those involved in different roles in the project, towards the development of the knowledge model and knowledge system.



**Figure 3-2 Roles in Knowledge Engineering (Schreiber et al. 1999)**

The image above is a graphical view of the six process roles in knowledge engineering and management from CommonKADS (Schreiber et al. 1999, pp. 21). It shows the six main roles that can be discerned in a knowledge engineering project. If one removes all references to knowledge, it becomes clear that this is basically an engineering approach to the development of a system for a non-specialist client for non-specialist user by trained a number of specialised professionals.

A scientist is an expert in a domain first, followed by being versed in the method of science. Philosophers or others that generally know a great deal about scientific method, do not perform the task of building scientific models based on the expertise of scientists in the field. Scientists do so themselves. Taking this idea, the proposed

change for continuous knowledge engineering entails moving from a team of professionals making use of clients and experts, to a team of stakeholders supported by professionals. Continuous knowledge engineering therefore entails significant changes in the roles adopted by different categories of people. The roles themselves will also change, as there will be a shift in responsibilities and intensity of involvement of the roles.

### **Knowledge Provider/Specialist**

This is the 'owner' of the knowledge, in other words the expert. The expert is a specialist in the domain or has access to part of the knowledge that is required for the knowledge system. This is the role most seriously affected by the changes.

Normally, the expert is a subject for study. In the new setting, the expert is involved in the content of the model as much as the knowledge engineer, be it with a different intent. The expert must extricate new lessons from the knowledge modelling effort, and be directly working with the model. It is his role to come to changes and adaptations. While the expert may not necessarily be the party responsible for making the changes (see the role of knowledge system developer), it is considered positive for his ability to locate new insights if he does.

Changes are made continuously to the knowledge model, either in response to feedback on the knowledge system or from personal insight into some matters in the knowledge model. These pass to the user of the knowledge by deploying the changes. Again, this may be performed by mediation of the knowledge engineer, or directly by the expert.

### **Knowledge Engineer/Analyst**

The role of the traditional knowledge engineer is split in two different roles in the model in Figure 3-2, the knowledge analyst and the knowledge system developer. The knowledge analyst is normally responsible for developing the knowledge model based on investigation of the domain, and acquiring the required knowledge from the 'knowledge owner'.

As a support for the expert, the activity 'elicits knowledge from' changes to 'supports'. The role of knowledge analyst may still include developing high-level analyses and designs, but these are made in cooperation with the expert. One could also stay within the division proposed by the model, and merely place the expert in the knowledge analyst role. This would not make it possible to make the activity of support explicit, nor would it make clear that the communication between the 'provider' and 'analyst' is reversed to a degree. The expert is no longer an actual provider for another party.

The role of the knowledge analyst changes as less work is done in the deepest recesses of a knowledge model. Instead, more effort must be given to structuring and aiding the development process, meaning that many actual modelling activities are placed outside of this role. This is either by preparing developments in the domain, or by guiding the developments made by the knowledge owner into more professional levels, as a mentor to the development of the knowledge specialist.

### **Knowledge System Developer**

The knowledge system developer is ordinarily responsible for the development of the knowledge system based on the specification of the knowledge analyst. This is a role

that is comparable to a software engineer and in smaller projects is often also performed by the knowledge analyst.

In a continuous setting, the knowledge system developer is more likely to be a role played by the expert. Sharing this role with the knowledge analyst is also a possibility. In addition, the knowledge analyst may still provide guidance and prepare designs in unison with the expert. In part this is to make a cyclic development process possible, and in part because the expert needs to be involved with the model to acquire new insights.

#### **Knowledge User**

The 'knowledge user' uses the knowledge system. In traditional software engineering and knowledge engineering methodologies, the user is only a part of the development, during introduction and afterwards originates much of the effort in maintenance. In recent approaches, the user is much more involved in the development, e.g. DSDM (Stapleton 1997) or Rational Unified Process (Jacobson, Booch & Rumbaugh 1998). The role of the ambassador-user is one as part of the development team.

In continuous knowledge engineering, as in other iterative development strategies make a finished system available to the user as early as possible. All users will gain access to the knowledge system earlier than normal, and will have to deal with an application that may be incomplete but can be steered by them towards further completeness.

#### **Project Manager**

The project manager normally manages personnel developing the system during a finite amount of time, measured in months or years, after which a product is relegated to a maintenance organisation.

In continuous knowledge engineering there is no final moment of development, or in another sense, the system goes into maintenance almost from the beginning. The length of involvement, although less intense, is much longer. A project manager may have many different small-scale knowledge systems in development with only a knowledge provider present to build first prototype system. This may evolve into a larger scale effort, where many people are involved in updating and maintaining the quality of the system. Such efforts therefore run counter to the expectations of project manager today.

#### **Knowledge Manager**

The knowledge manager is not part of the project team, per se. This person is in charge of knowledge strategy for the organisation, and initiates different projects to solve problems in this area. This may include the initiatives for knowledge system development, but other organisational measure may also be part of the initiatives undertaken by the knowledge manager.

The knowledge manager may see the different continuous knowledge engineering driven projects as a development similar to venture capitalist efforts. One of out ten knowledge systems of smaller scale may proof their worth and develop into a large-scale mission-critical application. Others can at least prove worthy of keeping up to date and proliferation to a limited population.

### Development affects people

The development process affects people as their accomplishments are measured using the standards set by the process. People involved in a project require understanding what the project is all about. This becomes easier when each increment is smaller, and by getting frequent feedback from the cycles of the project, a sense of accomplishment is created. They can discuss the schedule of their task that they can understand and influence. The effect on morale of these approaches was signalled as early as in Brooks (1987).

The tool support for non-computer scientist to participate in modelling is limited. In most cases, the tools are created for professionals that have gone through extensive training. For a change in roles to take place, either the tools need to support the users in adopting different roles, within the process, or dedicated solutions must be created within the project to allow them to participate. The first will require changing in existing tools to be usable by non-specialists. The latter entails customisable versions of a knowledge system development environment, specific to a certain project.

### 3.3.2 Project

The project represents the undertaking as a whole, consisting of a mission and resources with which to attain that goal. The project is the organisational element that manages the development of the knowledge system. In most cases, a project is a temporary affair, charged with the realisation of a goal, for example, the construction of an office building or the development of a new car. Afterwards, the project is dissolved, and control over the people, products and resources is relinquished. The results of the project are transferred to others for upkeep and maintenance, and often this is then seen as distinct from the project. When the development effort becomes a serious extension or revision of the system to warrant the existence of such an organisational element a new project is started.

When one goes from product based projects to continuous knowledge engineering the notion of a project changes in character. As a long-lived and continuous organisation, it is responsible for the existence of the system and its continued upkeep and upgrading. This requires an organisation comparable to that now in play for large-scale information systems, also seen in financial institutions. In these cases, the product drives the project, rather than the other way around.

It is the question whether the term project should still be used, as this term is normally associated with temporary efforts. The project assumes the guise of *stewardship*, placed in charge of the affairs of the knowledge system for its lifetime; supervised, kept from harm, watched over and protected. The project exists just as long as the system does. This requires continuity of commitment and operation. This precludes the use of temporary professionals, but long-lived participation from different parts of the organisation, incorporating the expert and the user. Affected parties and stakeholders must play a meaningful role in setting goals, defining priorities, and crafting policies. Consensus building is a slow and arduous process, but more time and resources spent up front can mean greater savings and real solutions further down the road. Also, compare this to recent developments where users are included in the design and development of software such as DSDM and Extreme Programming.

These project organisations require plasticity as their composition may change over time. They need to be conscious of changes around them to align the project to

perhaps changing objectives emanating from a fluctuating organisational context. Furthermore, they must remain informed about the status of the knowledge model and knowledge system.

This stewardship can also be taken up for more than one knowledge system. Families of knowledge systems could be created when the infrastructure for their development and introduction has been set-up. This would even out the initial cost of that infrastructure. From such families of knowledge systems flows a more evolutionary thought. Such approaches may lead to the development of venture knowledge systems. This inept term aims to describe the development of several knowledge systems, to locate the one killer-app between them. Although the term venture may seem to mean that each should be positioned as a commercial product, one could see such an approach working at smaller scales as well.

#### 3.3.3 Product

The products are the artefacts created during the lifetime of a project. While the main product is the system that is the goal of the project, this can further include models, source code, documentation, etc. Some approaches also include trained users as a product, and within this context that could also include an improved expert. The goal of the project is to create the target product. It is often described using a series of requirements to be pursued. This is similar to the state of affairs in software engineering, where the outcome of a project is seen to be a released product, followed by maintenance track (Jacobson, Booch & Rumbaugh 1998).

One of the aspects that often surfaces in software- and knowledge engineering is that the ‘users keep changing their mind’. This recurring change of heart originates in two factors. First, as time elapses, things change, the environment and organisation align to new concerns. Requirements therefore have an inbuilt expiry date. Secondly and more importantly, when users are involved in a project they are educated during the project to their wishes. The users become better users. This is in fact one of the most important results. The product in a standard method is an end-goal to be attained, not a running goal to be guarded. Science is not interested in finding the ultimate theory, but in improving understanding along the way. It pursues a running goal.

The products developed using continuous knowledge engineering (or other incremental, participatory approaches) are approached differently. One starts with simple systems and tries to evolve them to larger systems. This growth factor will initially allow system development with minimal means. Because their growth is continuous, the investment done is more gradual and can follow the impact and importance of the system. In this respect, it is important that those involved in the decision-making should have an extended perspective on these systems, as their true value may only show after considerable time (see stewardship).

A consequence of this is that there may be products, but there are no finished products, no final version, much like there is no final scientific theory. The knowledge model is always under construction. The users work with the most recent release of the system, not just a final version. On the other hand, this means that there should always be a working product. At every moment in time, during development, testing, etc. the knowledge system should be consultable.

From a pure product-based perception of a knowledge system, these issues are difficult to unite. A change of perspective on the knowledge system and knowledge

model is therefore required to shed some light on the matter. The knowledge system is treated as a viewer for the knowledge model, a medium for its publication. Much like an Internet browser, it is able to show and provide access to a store of information, a knowledge system browser a specific knowledge model. The Internet changes every day, but the browsers stay relatively static.

This view allows for the inherent dynamism of a changing knowledge model, and the idea of a fixed yet adaptive means of getting access to the latest knowledge. And while the knowledge model may be under development, this is not a reason for it not being used. The Internet is not put on hold, while one of the users updates a part of it.

### 3.3.4 Process

A knowledge engineering process is a definition of the complete set of activities needed to transform users' requirements into a product. A process is a template for creating projects. It describes the different phases that are distinguished and the manner in which they are traversed.

Many different approaches exist. Linear processes go through each of the phases one by one, while an incremental process divides each of the phases into increments. An iterative process goes through all the phases for a single increment and then repeats. A cyclic process is a specialisation of iterative processes, where each small change goes through all the phases. As the process becomes cyclic, it is best characterised as a feedback loop. The tasks seen in most development projects over a period of months could occur within days. This means the tasks such as described in the previous chapter all have to be performed frequently and in a relatively short time-span.

While the frequency of these tasks normally would be overwhelming, the work associated with each step is much smaller than usual. Accidental problems may make the passing through each of the phases in rapid succession a problem, but these can be amended. In a normal process, one of the main aspects of these phases is providing documentation of the process. The documentation and design models must be kept in line with the changes that are made. Alternatively, the development can follow a design-less approach, abandoning control and making management of the process hard or impossible. However, other possibilities may also exist, in finding a way to automatically generate documentation and update the design models used, like for example with executable specifications.

In addition, most processes are geared towards *forward engineering*, transforming models into implementation. This makes turning back or *reverse engineering* very difficult. Moving backwards is not possible. In fact, in many situations, documentation and product are not in line and the source code or knowledge model is the only accurate description of the system (e.g. McConnell 1993).

The most technically demanding question is the frequency of the updates that have to be made available to the users. At every cycle, a new product is released onto the users. This can prove problematic, as this is most definitely not standard. The common form for a knowledge system, as for other types of software, is as a standalone application deployed when a release candidate is created. The number of releases versions is kept down because distribution of the software and installing the new version involves considerable overhead.

Schemes that are more flexible exist. An example of this could be the automatic updates used by virus checkers. This update requires access to the Internet. Even these



update their system only on a monthly basis. Other possibilities also use the Internet as the medium to reach the users. This is in most cases a scalable solution. What is needed is technological support to make this both feasible and practical.

One of the most important challenges is the ability to test each deployed versions. Because of the frequency of deployment it is essential that this is both easy and cheap. Most likely approaches are corner-stone cases such as those in ripple-down systems (Compton et al. 1992) or automated testing such as the unit testing found in Extreme Programming (Jeffries, Anderson & Hendrickson 2001)

### **3.3.5 Conclusions**

The impact of the scientific metaphor in the incarnation of a continuous knowledge engineering method pervades many of the aspects of knowledge engineering. The roles of people change and require stakeholders to perform tasks normally in the hands of trained, skilled professionals. The timeframe of attention to the development switches from a temporary job to a long-lasting stewardship effort. The perspective on the product of knowledge engineering is transformed from an artefact to a model of knowledge for which a knowledge system is the medium for its communication. The process by which this product is updated and improved is cyclic, meaning many small sized changes need to be released regularly to the users.

## **3.4 Fundamental Method, Essential Tools**

The intention of this section can be found in this quote from (Jacobson, Booch & Rumbaugh 1998): "... to develop a [method] without thinking about how it will be automated is academic. To develop tools without knowing what [method] (framework) they are to support may be fruitless experimentation. There has to be balance between process and tools." The aim of this section is therefore to examine the way that some steps can be made to go towards continuous knowledge engineering by developing the method and the tools in balance. Method has a profound effect on the perception and perspective that one has on what one is doing, and why. Many methods share certain tasks and models, but differ in their perspective and intent.

The software tooling that is used to automate the activities defined in the process is an essential factor in making the transition to continuous knowledge engineering possible, viable and practical. In real world situations, the support from the tools determines the practical interpretation of the method. Tools make a method viable and practical. This section continues where the previous discussion of the effects left of, by enumerating requirements for tools and method along the division of the principles of continuous knowledge engineering. Although some of these requirements support more than just one of these principles or aspects, an attempt was made to create a meaningful categorisation. These requirements strive to propose a path towards realisation of the continuous knowledge engineering method by defining the interface between the method and tools that could support it.

In another man's words, "[t]his support must provide facilities to make it convenient (reasonably easy, safe and efficient) to use that style. Such a tool does not support a technique if it takes exceptional effort or skill to write such programs; it merely enables the technique to be used. For example, you can write structured programs in Fortran and object-oriented programs in C, but it is unnecessarily hard to do so because these languages do not support those techniques." (Stroustrup 1991, p. 14).

Therefore, this section concerns the changes that should be made to the aspects people, project, product and process and the requirements this places on the tools to provide active support for the continuous style of knowledge engineering.

The converse of the statement made by Stroustrup is also true. A tool focuses to support one or more styles of use. Each of these is weaved as an idea through the presentation of the facilities. While C++ allows its usage as if only C was present, this does not constitute best usage. Many of the aims and intent of a tool or language can be lost in that way.

### 3.4.1 Participatory Modelling Requirements

The following requirements concern the knowledge modelling capabilities and the ease of use of these representation systems for non-specialist users. In the list some of the requirements overlap, but they each have different shades of meaning that are valuable.

#### Understandability

Each of the knowledge representations, individually and in conjunction, have to support the ability to understand the knowledge model. The knowledge has to be readable. It must for example, not be necessary to have knowledge of the entire model to understand one particular facet. For instance, the intent of an individual rule can readily understood looking at the rule, without regard to other rules. However, when adding a new rule, it is unknown how this will influence the conjunction of rules. This makes it hard to ascertain the impact of such a change and makes it hard to understand a model's behaviour.

Often when a design model, the implementation requires a translation. This can remove the natural structure of the design with one optimised for a computer. All the names of symbols in a knowledge model can be renamed to the form M0008, and the system will still be functional to its users. To the modellers, a significant function of the model will have been lost.

Understanding is the pivotal element for verifying, locating, expressing, changing and testing, in other words, all the activities involving the representation of knowledge. An expert should be able to verify the knowledge as to its correctness and completeness, but a layman or novice should also be able to understand the working of the represented knowledge even if the reasons why, behind the represented knowledge, are beyond him.

#### Locatability

One of the aspects of understanding a knowledge model is to provide a notion of navigation and locating parts of the knowledge model. The structure of a knowledge model must allow whoever is making changes or merely navigating the model to comprehend it, to find what they are looking for. Related parts of the model should be in the same physical proximity. When a certain aspect requires verification, change or extension it should not require an inordinate amount of time to find that particular aspect.

**Expressibility**

An untrained person must be able to express knowledge in the formats chosen, and be able to know what the effect is of changes made to the representation. Besides the earlier ability to understand the model to a level, making it becomes possible to express to the knowledge engineer what should be done with the knowledge, the expert should be able to create and develop a knowledge model without the knowledge engineer. The expression of knowledge should provide support for finding inconsistencies and incompleteness if possible.

**Changeability**

The aspect of changeability concerns the extent to which a knowledge representation remains mutable. Knowledge models can ossify, because certain changes cannot be made without incurring a great or overwhelming cost. This causes incorrect or incomplete structures to be tolerated and worked around in new additions and subsequent changes. Eventually this leads to an un-maintainable knowledge model, where the only avenue of approach remaining is to start over.

One of the reasons behind a lack of changeability are the hidden assumptions and implicit knowledge in a knowledge model, which is weaved through a model and provides an important support structure. These cannot be easily removed once they are in place and make it impossible for any individual part to be used separate from the remainder of the knowledge model. An approach that requires the plasticity of its components requires separation between the elements of which it consists.

**Extendibility**

In order to grow a knowledge model into a level of maturity and allow it to follow the needs of the times, the knowledge model will have to be extended from time to time. Extension means providing functionality beyond that described in the initial specification. In some languages, it is possible to distinguish extension points in a model, where the design has prepared for some aspects to be elaborated in the future. This kind of extendibility is very important. Another form of extension is required where functions and system or model behaviour that was not predicted in advance have to be added. This can be harder and is connected to the changeability discussed before.

**3.4.2 Project Stewardship Requirements**

These requirements specify the support that can be given to the continued stewardship of a knowledge system development. The steward is the guardian of the knowledge system, keeping the knowledge model and knowledge system from harm, managing further development, monitoring the status and taking action when necessary. The term steward implies care and responsibility, but of property that does not belong rightly to the stewards.

In industry and other parts of society, the responsibility of the producer of certain products with an environmental impact has been extended beyond the moment at which the product is delivered to include usage and decommissioning. This translates within knowledge systems to form a development process that incorporates usage as a step in the development.

This requires the released product to be somehow still under the control of the producer, after deployment. For example, the project management could know each of the users, and maintain a record of their usage information. This and other user monitoring may violate common rules on privacy, which must be resolved. However, this would not go far beyond what is considered normal in monitoring Internet users.

As a requirement for tools, this is a weak form of requirement. It would be served by a centralised form of deployment, instead of a distributed, non-integrated form such as deployment through stand-alone software. As such the stewardship aspect has a limited effect on the tool requirements, but parties, involved in the management and monitoring of a knowledge system development, do require information from the system and model development as well as usage. The development information may provide insight in realised functionality, in addition to the status of known issues with the knowledge model. The latter may for example include metrics, bug reports, etc. Other software can also provide this kind of functionality.

### Usage Information

Usage information is required to know the status and provide feedback on the usage of the knowledge model. If possible, such information should include data on the usage of different aspects of the knowledge model. This can give insight into the areas of the knowledge model that require fleshing out.

The case histories that are created thanks to the use of the system also provide an essential source of information. This can show hotspots, i.e. areas that are used predominantly by the users, and dead areas. It can also provide feedback on the quality of certain consultation results. This may show areas where knowledge is lacking or provide information on lacunas in the required knowledge. Data-mining approaches may also be able to glean important patterns in the usage. The results should at least provide information to ascertain the quality of the solution being offered and critical feedback on the knowledge exercised by the different consultations.

### Metrics

Metrics are widely professed in software engineering as means to understand and monitor complexity of systems as well as measure progress. The most well-accepted are probably Lines of Code and McCabe's cyclomatic complexity measures (McCabe 1976). More recent ones aim at object oriented specific metrics, and concepts such as function points

Most metrics can be divided into three classes: raw measures of size, measures of various attributes of complexity, and compound measures intended to assess productivity. Metrics can be used to:

- Estimate completeness of a knowledge model, or part of a model.
- Judge the overall volume of knowledge in a knowledge model.
- Indicate how difficult a knowledge base might be to navigate or modify.
- Enable subjective comparison of knowledge models to see how domains differ, in order to inform the estimation of future knowledge modelling and development tasks.

These types of measurements can give an idea of underlying subjective phenomena and can provide an approximation of the core notions of interest to developers,

managers and researchers. Within knowledge engineering, such approaches are less common, a notable exception being (Lethbridge 1998, 1994).

Such information can be provided in part by the development tools that are employed. While these can be used to compare different knowledge models developed using the same tool, across tools such comparisons are much harder. This is caused by the lack of uniformity of representation found in these tools.

### **3.4.3 Application as Medium Requirements**

These requirements detail the need to be able to develop professional industrial strength knowledge systems, and the auxiliary issues that are associated with this. An important aspect originating in the continuous knowledge engineering method is the view of the knowledge system as a medium for the communication of knowledge.

#### **Operational Knowledge Models**

As the knowledge model is continuously changing it is constantly under construction. For the knowledge system, it is important that the latest available knowledge is used. To have a continuously operating system, it is therefore not acceptable to be un-operational, and when lots of small increments are added, it is essential that they can be deployed to the user as soon as possible. Therefore, the requirement that is placed on the tools is to have an operational model at all time.

In a sense, this is a common requirement for knowledge system development environments. Rule-based systems often conform to this requirement. Adding a new rule does not require reworking the whole system.

The requirement makes it possible to create a working system directly from a knowledge model, and with little effort. A trivial system, e.g. a system that consists of a single rule, should be usable at a minimum in effort and auxiliary issues. Even though this may not produce a useful system, it illustrates the need for being able to create a tiny system with little or no fuss. There should be no threshold for knowledge systems to be developed based on a knowledge model.

Design-to-implementation strategies are therefore not acceptable as-is. With respect to cyclic development, it is important because the transformation of a change in a design-model to a change in the knowledge system as presented to its users is normally involved. Testing of a change can only occur after implementation of the feature. This longer turn-around time is detrimental to change, and therefore an impediment to cyclic development.

#### **Incrementally, Modifiable User Interface**

It must be possible to change the user interface of the system to create an easier system to use, and one that is more palatable to the user. This ability should be given in such a way that it is possible to create these user-interfaces in small bits, rather than require a complete effort, as soon as one deems it necessary to develop or adapt the user interface. It should not be necessary to develop a user interface upfront.

#### **Differential Visualisation**

A knowledge model must be usable by several user groups, with different skills and facilities. Based on a single model, it must be possible to create different visual representations of the knowledge system. This may go as far as offering different

functionality to mediate differences in the knowledge level of the user. This requires allowing different user interfaces working with the same knowledge models. This requires similar facilities as the previous, but must also allow different visualisations to co-exist and be developed in parallel.

### **Dedicated Solutions**

At the other end of the spectrum is the development of a dedicated solution. A dedicated solution means being able to create system especially for a specific purpose and customised to its needs and requirements. It must therefore be possible to use a knowledge model in a variety of ways and create a system tailor-made, using advanced visualisations techniques and with the possibility of integrating the system with many different information and knowledge facilities.

### **Integration**

The integration of knowledge systems will become more important and more an issue for knowledge system development over time, both in terms of process and of product. Knowledge systems may currently be seen as primarily standalone applications, this will change over time. In part, knowledge technology will become as special but unified technology within software engineering. The development processes used require that their modelling languages and those employed in knowledge system development match. This is a similar situation to that in database technology, these occupy a special position within software engineering and efforts are made to allow software engineering processes to discuss structure and access of databases on the same footing as other systems within a product.

This is also true for the development of a knowledge system itself. Initially, the need to integrate a knowledge system with other systems may be latent but as a knowledge system matures, integration becomes a paramount issue. This goes in three directions: integration into other systems, incorporation of other systems and communication between systems as equals.

### **3.4.4 Cyclic Development Requirements**

The requirements aim to realize the ability to operate continuously and in a cyclic, iterative manner on the development of a knowledge model and knowledge system.

#### **Start Minimal**

The first system that can be developed and released onto the user community ought to be realisable in a very short period. This minimal system should present as small an effort as possible and still be meaningful to the user. The size of the first increment determines the threshold that exists between not having a system and having a system.

#### **Small increments**

This requirement concerns the granularity of the changes and extensions made. The changes that can be made and have to be made and can be deployed in the content of a knowledge model have to be able to be small in granularity. It must be possible to deploy a single change in a rule, with little or no cost. Within Extreme Programming<sup>7</sup> these are called “small atomic bits of functionality” (Jeffries, Anderson & Hendrickson 2000).

**Revoke changes**

It must be possible to revoke changes to a knowledge model, and return to an earlier version of the model. In part, such functionality can be realised using version control systems, therefore this criterion only concerns the development within the modelling environment, for example an undo/redo mechanism.

**Scalable from Small to Large**

The system has to support the scalability of a knowledge system from very small scale to larger systems. This means that the method and tools must allow sustain small systems, just as well as medium and large systems, and support the transition through each of these phases.

**Gradual Integration**

Another consideration in this respect is the ability with which a system can integrate other knowledge models, and how easy a developed knowledge model can support its integration. In many growth scenarios, the communication and linking to other systems becomes more and more relevant, where this is often a minor concern in the early stages of development.

**Easy to make changes**

The representations used must allow for changes to be made with ease. This would mean changing the model, within the purpose and structure that it has, to repair or adapt the model to a better implementation of that purpose. There must be no penalty for making a change, beyond the effort of that change itself. Furthermore, to deploy a changed model must not be penalised either.

**Easy to extend**

A related requirements to the one above, this requires that the knowledge model must allow extension, i.e. incorporation of new features and functionality, and this extension must be possible without great effort. Considering the time-frame in which development may take place, this can mean that the purpose and structure of the knowledge system can change considerably over time and that such changes need to be accommodated by the process and the tools. This requirement therefore requires support at a different timescale than the previous one.

**Easy to deploy**

A new system or increment should be easy to deploy to its users. A modeller should be able to send an update to the users after validating the changes. Current default software deployment techniques are not sufficient for this purpose.

**3.4.5 Conclusions**

The requirements above are quite broad and in some cases show more of the requirements to the process than the tool, but both are united in their purpose. In some cases, the effects or benefits that are expected are mentioned. In part, these requirements form an evaluation method for tools and development methodologies; if they conform to some degree to the requirements mentioned in this section, they can be said to support the continuous knowledge engineering approach. This will be subject of discussion in later sections. However, in line with the principle mentioned

by Jacobson at the beginning of this section, they will likely to work better if both are dedicated to a unified purpose.

### 3.5 Architectural and Technical Concerns

Considering the realisation of the requirements discussed above something can be said about the architectural concerns and techniques that may be employed. These are discussed here because the next chapters are about the actual implementation of some tools, and not all these techniques will be employed in each tool, although all are relevant. Furthermore, discussing them here on an equal footing will enable a focus a better focus on specific concerns in the chapters on tools.

#### 3.5.1 Vivid Knowledge Representation

Direct, analogical representations or as described by Levesque (1986) as *vivid representations* represent knowledge as a model of reality. This may sound trivial, since one could assume that any knowledge representation is a model of a domain, which is not true. A knowledge representation is not a model of its domain if it does not present a surrogate of it, like a neural network. But even representing a domain in logic or rules does not it mean it is by definition also a model. For example, compound assertions such as:

```
Jack is married to either Jan or Jill
Nobody is married to Jan
```

These allow the conclusion that Jack is married to Jill, but this does not capture any essentials about the domain. Also, the rules are encoded in such a way that there is a limited context dependent use for them, while any useful assertions based on such rules in other contexts can soon become intractable.

A vivid representation evokes the same kind of imagery the domain itself does. A knowledge representation is vivid if:

1. There will be a one-to-one correspondence between a certain class of symbol in the [knowledge model] and objects of interest in the world.
2. For every simple relationship of interest in the world, there will be a type of connection among symbols in the world such that the relationship holds among a group of objects in the world if and only if the appropriate connection exists among the corresponding symbols in the [knowledge model].

This means that the world and the model share important basic features, and to find a relationship in the knowledge means that such a relationship should also exist in the world. Vivid knowledge representations are analogues of the domain, we can operate on them as if they were the domain itself. Direct calculations on a vivid model should be able to determine what is true in the domain itself. For example, on a map one is able to find out the distance between two cities by using a ruler to measure the distance between two cities.

To answer the question, what age is Jan, the information should be ready to use. The information could be stored in the knowledge model as a fact, or there could be a proposition telling that Jill is married to Jack. The example is a puzzle, which could be direct. This means that base facts about the world should be easy to answer, without the need to infer anything. If in the world it is easy to determine whether Jack



and Jill are married, then the knowledge representation should allow this to be as simple as a database look-up.

For knowledge engineering vivid knowledge representations have an important secondary effect, which means that as lessons are learned from making new discoveries in the knowledge model these will translate to lessons learned on the domain. This implies that vivid knowledge models are exceedingly useful in supporting the discovery of new knowledge.

### **3.5.2 Visual Knowledge Representation**

Making sense of the environment by provided visual or graphical models enables certain easy to comprehend representations of sometimes very complex and interrelated associations between elements. Related to graphical or visual programming, it receives much attention from researchers (e.g. Kremer 1997, Menzies 1996b). The attraction of the saying that one image can say more than a thousand words is quite strong. Software and knowledge remains are inherently difficult to visualise properly (Brooks 1987).

In previous chapters two representation techniques that are coupled to highly visual counterparts were presented among others that will be examined in following chapters on the tools to be developed. The decision table leans quite heavily on the fact that it can be presented in such a way to make the relationships between several related rules clear. Furthermore, inconsistencies or incompleteness within the decision table are quite easy to locate. As such, it is a good contender for knowledge representation that is easy to understand and has good characteristics concerning its changeability and extendibility. From the standpoint of participation of stakeholders, it therefore seems a good candidate.

Object-oriented models are to a degree forms of visual modelling. UML and its predecessors gave much attention to the visual nature of its models (Rumbaugh, Jacobson & Booch 1998). By virtue of the direct support for localisation, change and extension, object oriented models are quite good at communicating ideas from software professionals to stakeholders and vice versa. They allow them to comment on the model, and provide feedback that goes directly to the design. The vividness of the model allows parallels with the experts' own understanding of their domain.

Other issues such as the principles of encapsulation, abstraction and information hiding are also important to the conciseness and modular nature of the models, having no small impact on the readability of an object-oriented diagram. Another aspect to object-oriented representation is integration. It plays an important role when using object-oriented knowledge representation, as many software developers use object-oriented languages and design specifications. Speaking the same language makes it easier to integrate these concerns.

### **3.5.3 Component Based Development**

From the previous, a need to deliver one product in several formats was seen. As a stand-alone application, integrated into an existing piece of software, with more than one visualisation of the same knowledge model, and perhaps also in other forms. To be able to offer such a solution an important concept is component-based development (CBD). This is an approach to system development whereby systems or parts of systems are developed through assembling, configuring and combining

different reusable software components. These components are autonomous units that constitute reusable software units offering certain functionality and are designed to be employed in flexible configurations of other components and applications. It is often seen as complementary to object oriented development, although some confuse it as being identical.

Component-based development allows the development of different applications and systems reusing the same components. CBD embodies the 'buy, don't build' philosophy advocated by Brooks (1987). CBD is also referred to as component-based software engineering (CBSE) (Brown 1996). It allows for the development of, for example, different development environments based on the same components. In this way, both standard and dedicated version of such an environment can be developed with ease. Java Beans for example, fall into this category, where an entire architecture is divided into autonomous software units. The ties between the different components are kept as minimal as possible. CBD gives the possibility of developing a knowledge system shell but with the freedom to integrate that shell with any regular programming language and providing any type of user interface that is wished for.

### 3.5.4 Model-View-Controller Framework

The Model-View-Controller (MVC) framework is prototypical for model-based applications, which makes a strict separation between data and use of that data, or model and view of and control over that model. Its origins can be traced to its inclusion in the Smalltalk IDE (Integrated Development Environment) as a primary architectural construct by developers of Xerox PARC (Krasner & Pope 1988). Several versions of it exist, each attempting to take some of the negative points away from its realisation. The Swing classes, which provide the Java with extensive User Interface capabilities, are based on MVC. The Document/View separation in Microsoft Foundation Classes (MFC) is also inspired by it. Several current day tools are flaunting it as part of their support to building ever larger and more complex systems, like VisualAge (Stanchfield 1999).

The framework separates the system into three distinct parts:

- Model**        Responsible for access to the data, responding to mutation requests and maintaining data consistency.
- View**         The elements showing or presenting the model to the user. For knowledge this is known as a mediating presentation. These can also present access to the control capabilities.
- Controller**    The controller maps abstract actions undertaken by the view to the actual actions necessary on the model.

The view can perform actions on a selection of the model through the controller and is informed of changes by any party to the model. In this way, the necessary updates are made to all views showing the model. In a structured, more complex model, the controller and viewer focus on part of the model, rather than the whole. In many implementations the controller and view are joined together, as the viewing and editing functionalities are often integrated into one class or element.

This gives the design the following features:

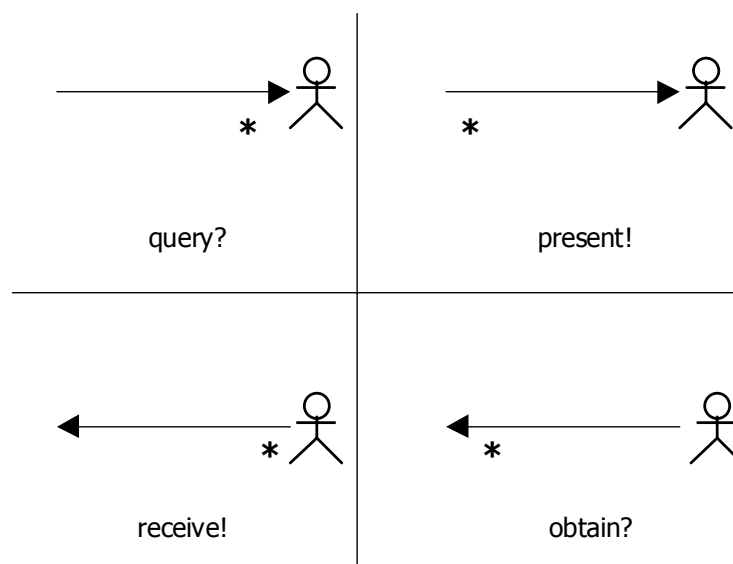
- Clarity of design
- Extendibility
- Maintainability
- Exchangeability

Comparing this to the requirements set out earlier shows that this architectural choice fulfils many of the essential functions. This separation of concerns enables the replacement of any of the components by equivalent counterparts (cf. CBD). As an example, and a preview to later discussions, another model may replace the (knowledge) model, while keeping the same control and view elements.

As a concrete example, we use a tree or composite. A tree can be represented as a node that can contain other nodes. This can be presented as a tree in user interfaces in several ways. In addition, some trees are mutable, where others are fixed in nature. This may affect their use but not their inherent data structure. Therefore, a viewer can be constructed that can show any element implementing a Composite interface. Any controller that gives access to methods to add, modify, or remove nodes can be used to modify the composite. Therefore, whether the model holds a directory tree or a family tree the application can operate satisfactorily. Services added to this, such as undo-facilities are implemented once, and can be reused many times.

### 3.5.5 Abstract Communication Model

In this section, the need for an abstract communication model is discussed. An abstract model of communication, i.e. one without designation of form or visualisation, allows customisability and presentational diversity. These are both requirements mentioned earlier. It is also a prerequisite for CBD, and application of the MVC framework. In short, it separates the visualisation of communication from the message communication action itself.



**Figure 3-3 CommonKADS Communication Modalities**

Figure 3-3 shows the four modalities that exist in system/user communication. The communication model above is taken from CommonKADS (Schreiber et al. 1999).

- Present** The system shows information to the user. For example a picture, an intermediate- or end-report.
- Obtain** The system requests some information from the user. This is the typical question to the user.
- Receive** The user provides information to the system unprompted. This may be entering some information describing the problem or for example some general information to differentiate the consultation from another one, merely for administrative purposes.
- Query** The user asks the system for some information. This can be a question based on the status of the system or the start of an evaluation.

The asterisk \* indicates on which side, knowledge system or user, the initiative lies. The communication direction is indicated by the arrow direction, from the system to the user or back. The question mark shows whether a response is expected, an exclamation indicates that it is a message or report, without response to be expected.

```
(MSG
  :TYPE query
  :QUALIFIERS (:number-answers 1)
  :CONTENT-LANGUAGE kif
  :CONTENT-ONTOLOGY (blocksWorld)
  :CONTENT-TOPIC (physical-properties)
  :CONTENT (color snow ?C))
```

**Figure 3-4 KQML Message**

Other models and standards for communication also exist, for instance KQML, the knowledge query and manipulation language could also be used (Finin, McKay & Fritson 1992). KQML is influenced by logic and logic inspired languages such as LISP, but it allows use of other underlying representations. Queries are sent to ask questions (color snow) or assert new beliefs for the agent. KQML knows many more types of communication than those proposed for CommonKADS, and it is not meant here to equate them or size them up for comparison.

Another example, which is directed towards knowledge interchange is KIF, the knowledge interchange format (Genesereth & Fikes 1992). Also based on logic as an underlying representation, this format has been criticized for attempting to formalize a common basis for communication when most of the questions needed to develop such a standard remain unanswered as yet (Ginsberg 1991). KQML and KIF show through the need to communicate and transfer concepts and beliefs as symbolic structures. Considering earlier comments, it remains to be seen whether such approaches will reach the levels of effectiveness that are necessary, and that such communication does not hinge on the acceptance of static, standardised and therefore stale ontologies and knowledge structures.

The approaches mentioned here have different, complementary purposes. CommonKADS treats communication as tasks, like knowledge goals (cf. planning), whereas KQML attempts to define a protocol for doing this and KIF proposes itself as a possible format for the content of the messages sent through KQML (also see the CONTENT-LANGUAGE in the figure above).

These types of protocol can also be added as additional layers and services to an abstract communication-model. MVC normally only allows for those actions where

the initiative lies with the user: query and receive, and only sends informative messages when something is changed in the model, but has no facilities for explicit communicative actions. This means that services have to be extended to allow for the knowledge model to initiate communication.

### **3.5.6 Centralised Deployment**

Normal deployment is quite costly and requires a concerted effort. Because of this, it is something that occurs only a couple of times per year. Otherwise, other schemas are more effective. Examples of this are systems that check with regular intervals for updates, such as virus-checkers. The mutable part of the application in these cases is known in advance.

To enable the deployment of changes and new systems to users, a different approach than standard deployment such as CDs or Internet downloads is required. With software that is continuously changing, and requires frequent updates, centralised schemes are more appropriate. One of the possibilities is the use of Internet as a means to access the developed applications distributed while using centralised deployment. Deployment of a change or iteration then means that only single central location needs to be updates, whereby all users will have the latest version of the system as their disposal. Minimally the knowledge model is centralised while the knowledge system may be distributed to its users, going on the assumption that the latter is relatively fixed. This may be complementary to the component-based development described earlier. A component for knowledge systems may be augmented with a visualisation engine that dishes out web pages, instead of the normal forms. Especially for the standard question-answer interaction, this can be accomplished quite easily.

Distributed applications such as these web-applications are becoming more and more commonplace, which eases the introduction of these facilities for knowledge systems. These types of systems offer great advantages. The very low deployment cost and effort are important. They constitute an essential factor for any cyclic development process. Another benefit is that the knowledge does not leave the confines of the company or the country, but only the questions and answers do. Normally such data would require encryption and this is only a limited protection.

A further advantage lies in scalability; the effort to allow one user is the same as for thousands or tens of thousands of users. The only additional effort may be the installation of extra server machines, when such numbers of visitors is reached. Furthermore, the required infrastructure can be shared over different developments.

Information gathering on the usage of the knowledge system and the feedback from the users is essential for the verification of the systems quality. Not only does it give information on the impact of the system, the case history can also provide important insight into the usage of the system, and the hotspots and dead-areas. This information can be used to steer further development and sell to management that the knowledge system is essential.

### **3.5.7 Executable Specification**

Many development environments use textual code that is compiled or interpreted, where this code is the result of a previous analysis and design in some design method's modelling language. The path from conceptualisation through design to

implementation can be a long one. On modification of a system's functions, theoretically, the design is changed and these changes are subsequently implemented in the system. To temper some of the cost, some propose structure-preserving mechanisms (Schreiber et al. 1999). In practice, this leads to design and implementations that are out of sync (Jeffries, Anderson & Hendrickson 2000).

On the other hand, knowledge systems already have a tradition of tools that allow the consultation of a system at any time during their development. By increasing the conceptual level of modelling in a knowledge system, and by allowing such a design to be executed immediately, model and implementation become the same.

This negates any problems relating design and implementation, relegating the discussion on design and design-less approaches to the background. Both live models (discussed in the section below) and this form of transformation-less development can be supported using the program as a dynamic medium that can accept and run any query.

*Executable specifications* are modelling approaches, where the model is operational, or in other words directly usable or executable. No transformation to another medium such as a programming language is required. This can make reversing one's steps a lot easier than when such a transformation has to be reversed (see reverse engineering). In addition, these transformations are often quite technical in nature and could prevent the expert to be able to create a model and publicise it independently. To make it possible for a modelling approach to be used to remain close to the conceptual model of the knowledge, but eliminate the transformation from model to implementation, executable specifications can be used.

The critique that has befallen executable specifications can be most succinctly seen in the article by Hayes & Jones (1989). The main criticism by Hayes and Jones is that the executability conflicts with the expressiveness of a modelling language. Another problem they perceive is the overspecification in executable specifications. In order to make the model executable additional detail must be given, which may unnecessarily constrain the choice of implementation. For example, where a specification might have specified that a non-specific sorting procedure be used, an executable specification would force the choice to be one of the available techniques such as QuickSort. The final argument made by Hayes and Jones is that there exist formal specifications that are not executable. They describe a number of examples that in their opinion show that abstract specifications are not necessarily executable (Hayes & Jones 1989).

Fuchs formulated the counter case, and since has refuted all the examples provided to support this by relying on the properties of declarative languages such as Prolog (Fuchs 1992). This means that all these examples can be stated in predicate logic and can more or less directly transformed to executable form. Any solution made by Fuchs can therefore be transformed into a solution on a procedural level. This means that such statements must be expressible in a form that accommodates declarative statements but they are solved by procedural solutions.

On the subject of expressiveness, Fuchs sees a role for the developers of these languages to ensure that they remain expressive. It is the question whether overspecification is necessarily a bad thing. For one, a working solution may be preferred to a possibly optimal solution. The optimal nature of a solution is always very context dependent. Or, as stated by Fuchs himself (1992), "...executable

specifications can serve as prototypes, which allow to experiment with different requirements, or to use an evolutionary approach for software development. This is especially important since in many projects the requirements cannot be initially stated completely and precisely". The latter is exactly the situation when discussing knowledge system development.

In a transformational approach, i.e. one where specifications are transformed into the implementation, the specification will form the only relevant document for all phases of software development. In the proposed approach, this only document is also the application. The proposed "design = implementation" approach implies an extension of UML like models and formalised specification to an executable form of the Object Constraint Language (OCL) (see (Warmer & Kleppe 1999)). This creates an object oriented equivalent of an executable specification.

### 3.5.8 Live Model

Most environments for software development of knowledge engineering are either compiler based or interpretation based. This means that one codes (writes) first, and then transforms the whole into a program, or runs the system interpreting the whole. This often involves making several changes before a compilation is made. This approach precludes making any changes at run-time.

Such facilities do exist, where the coder and the program have the capability to make changes to the program, while the program is running. Reflective systems for one are a technique that allows for such enhancements, although its focus is on self-change. A *reflective system* is one that reasons about and acts upon itself. Reflection has been a technique and programming capability for some time and has been introduced in object-oriented systems (Maes 1989). Introspection, i.e. examination of a program structure, and dynamic loading of classes is also a facility present in other mainstream programming languages, such as Java. Smalltalk knows similar mechanisms such as dynamic compilation. No compilation of code is necessary to reach this form of modification or feedback. All modifications are performed at run-time, while the system is operational.

A live model is one where modifications to a system are made directly on the system, rather than to code to be transformed into a software system at a later time. The system immediately reflects the changes, or refuses the changes when they violate certain constraints. The system can report any errors or failures that a single change brings with it. Live models are related to reflective approaches, but the support goes beyond making constructive changes, or developing towers of reflection to allow program changing themselves (the essence of reflection).

Changes to the system for example deleting a certain attribute may make those parts of the system that reference that attribute invalid. Disabling these parts and reporting it to the modeller is then a possibility, keeping with the principle of having an operational model for all time. The effective cycle from design to development is very short. Every single change is merged into the model, with immediate feedback on its consequences and side effects. Simple syntactic and semantic errors are easily located. Therefore, it fits very well with a cyclic process. The approach is related to the use of transformation-less development.

### 3.5.9 Conclusions

	Vivid Knowledge Representation	Visual Knowledge Representation	Component Based Development	Model View Controller	Abstract Communication	Centralised Deployment	Executable Specification	Live Model
<b>Participation</b>								
Understandability	✓	✓			✓		✓	✓
Locatability	✓	✓			✓			✓
Expressibility	✓	✓						✓
Changeability	✓							
Extendability	✓							
<b>Stewardship</b>								
Usage Information				✓	✓	✓		
Metrics					✓	✓	✓	✓
<b>Medium</b>								
Operational			✓				✓	✓
Gradual UI			✓	✓	✓			
Diff. Visualisation				✓	✓			
Dedicated Solution				✓	✓			
Integration			✓	✓	✓			
<b>Cyclic</b>								
Start Minimal						✓		
Small Increments						✓		
Revocable Changes						✓		
Scalability			✓					
Gradual Integration			✓					
Easy to Change								
Easy to Extend								

**Figure 3-5 Requirements and Features**

The above table shows the support that these features can offer at an architectural or technical level. These different architectural and technical issues are by no means a complete list and not all of these will feature as strongly in the realisations of the tools. The list is also an inventory of the influences that have played a role during the development of the two tools. What the table does show is that the attention from the different features overlap in their support for the requirements stemming in the continuous knowledge engineering approach. This will make it very difficult to attribute certain benefits to any one particular technology that was employed.

## 3.6 Evaluation Method

The goal that was formulated in the first chapter was to solve the four scientific problems of knowledge engineering. The global solution that was chosen was to arrange the development of knowledge systems as a learning process, or a process of continuous development. This chapter has shown the details of that approach.



In the previous, a path was described towards a principled program leading to changes in many different aspects of knowledge engineering. This includes requirements for the manner in which these knowledge systems should be developed and for the tools that aim to support this manner of operation. Finally, a variety of techniques that could be employed to create these tools.

The next step in this thesis is the implementation of the solution and its demonstration in a number of case studies. After which the evaluation of the proposed solution can take place. The implementation of the solution in part is the development of tools for the support of the method. The plan is to create such tools, to realize the solution, and test these in a number of practical applications.

Now the problem is known, and the proposed solution has been presented, a look ahead is made to use this additional insight to detail a method for determining the quality of this solution. This section formulates the methodology to evaluate the veracity, quality, and effect of the continuous knowledge engineering approach and, vicariously, the scientific metaphor as a basis for knowledge engineering. Furthermore, the tools that are developed are analysed to see whether they provide the required support and the effects of this support. By formulating these criteria, the task of validating the support through experimentation and application is made structured and transparent.

### 3.6.1 Objective

Different difficulties are commonly identified in evaluating both process and tools. Few application areas can be used to perform repetitive tests, complicated by the fact that both domains and domain experts cannot be assumed homogeneous. Furthermore, it is hard to compare different models, while the quality and completeness of the knowledge acquired is hard to confirm. Thereby, the influence of the different tasks in the domain in relation to the kind of tool or process used still is unknown, but some relationship can readily be imagined. In addition, it may be hard to distinguish what the effects are, and to what cause they must be attributed; if a negative result is seen, is this due to the problem, the domain, the expert, the knowledge engineer, the tools or the methodology used. Finally, in collecting different data from possible metrics, none of these metrics is easily used or interpreted.

This does not mean that such attempts at evaluation are not made, for example, the Sisyphus experiments, e.g. (Linster 1994). In (Hayes-Roth, Waterman & Lenat 1983), a similar experiment using a mystery expert is mentioned. Menzies (1998) also discusses this and proposes that testability should arguably be part of any approach to situated knowledge engineering.

There are several reasons for conducting evaluations of KA tools. Evaluations that compare tools with each other can indicate the strengths and weaknesses of tools that support the same knowledge engineering methodology. This evaluation can also be used to identify of the features of the tool that require further development, for instance increase the ease of use, enrich the user interface, and generally improve the tools.

In addition, knowledge system development tools are often constructed around different methods or techniques, and some methods and techniques are only possible using a particular tool or tool-set. In these cases, comparing tools involves comparing the underlying methods and techniques that they support. Just as the areas of a tool

that need improvement can be identified through its evaluation, so can it identify the areas of improvement for the underlying method or technique on which it is based.

An additional consideration is that experiments can reveal the way in which users perform the tasks the tool supports. There is still a lot that is unknown about knowledge acquisition and modelling, as well as the actual influence the methodologies have in the quality of the process. An essential factor in the evaluation of both tools and methodologies is the use of metrics. There are few metrics for evaluating knowledge system development tools, techniques, methods and products (cf. Lethbridge 1998, 1994). To evaluate these tools effectively accepted metrics to evaluate them against and allow identification of good and bad tools, techniques, methods and products (Menziez 1998).

The purpose of a knowledge system development system is to support the construction of knowledge models, and the essential purpose of all evaluations is to improve the product. This is true both in helping to identify which tool or technique to use, or by improving the tools and techniques that are available. This will be the subject for the following chapters.

The research in this thesis will attempt to create answers at a few different levels, to strengthen the image that they create as a whole and to cross-validate the conclusions that are drawn from them. Starting with the conjectures that project from the discussion on the engineering and scientific metaphor, this continues with an examination of the people, project, product and process aspects of knowledge system development. Secondly, a number of questions exist on what kind of tool support can realise the changes required by the continuous knowledge engineering approach. Finally, what are the questions that exist to directly evaluate the quality of the solutions? The evaluation first is in terms of the scientific problems of knowledge acquisition and maintenance and the ability to create industrial strength knowledge systems. Secondly, it examines the practical aspects of cost, risk and benefits and thereby the bottom-line problem.

### 3.6.2 Metaphoric Evaluation

Concerning the two metaphors that were described in this chapter, there remain to be some questions whether the engineering metaphor may provide a better basis for knowledge engineering, than the scientific metaphor proposed here. Alternatively, is there perhaps a need to combine the two metaphors in a common approach?

When looking at the conjectures of the scientific metaphor it is clear that they make a number of interesting predictions about the nature of knowledge system development. The same goes for the engineering metaphor.

The arguments for the engineering metaphor would collapse if it could be shown that the following assumptions are true:

- People: Experts are capable of independent modelling.
- Project: Knowledge systems without design can be created and knowledge systems see more change after deployment.
- Product: Knowledge systems see considerable change within the knowledge to keep the system in line with the specification, while the environment
- Process: Knowledge systems can be constructed through a cyclic incremental process, rather than a phased incremental model.

The arguments for the insight-based, scientific metaphor in this thesis would collapse if:

- **People:** Experts are not able to either participate or be independently involved in knowledge modelling.
- **Project:** Knowledge models do not require significant change after deployment.
- **Product:** The motivation for changes originates outside the modelled knowledge and the knowledge system, by break down of the specification.
- **Process:** Knowledge system not be constructed without a design, or cannot be constructed in a discovery process.

Clearly, on these points both metaphors are contradictory, which makes them the ultimate vehicle for the evaluation of the implementation and demonstrations further on. This evaluation will take place at the level of the method and the tools, where these issues are elaborated.

### 3.6.3 Continuous Knowledge Engineering Evaluation

At the level of the development method, as derived from the scientific metaphor, certain lower level questions exist. The questions, in order of specificity, will be considered as goals for the method:

*Realisation* – is it possible,

*Qualification* – what advantages and disadvantages does it attain,

*Quantification* – what quantitative results does it yield in what circumstances?

These questions are divided over the aspects that they originate in. The questions on these aspects are split into questions that can be directly answered, without an analytic process.

#### People

In the explanation created from the engineering metaphor, knowledge is hard to articulate by experts. They require a skilled knowledge engineer to create a knowledge model, which constitutes a design by formulation and formalization supported by a modelling activity. The insight-based scientific metaphor is convinced that an expert is best person able to conduct the modelling, because they share the same context as the knowledge system, and they are by virtue of their training and experience best able to appreciate possible new lessons to be learnt from the modelling and the application of the knowledge through the knowledge system. The engineering metaphor then predicts that experts cannot model, where the scientific metaphor requires such a mode of operation as essential. Without it, continuous development becomes an impractical approach.

*People Test:* Ascertain whether experts are capable to participate in knowledge modelling or can independently model their own knowledge.

The test allows the determination of the ability to employ experts as a direct source of knowledge over the lifetime of the knowledge system, either directly or by involvement on the knowledge modelling. This will give insight in the possibility and the degree to which this is possible.

### Project

The engineering metaphor presupposes that knowledge system development is a finite process, followed by maintenance. The scientific metaphor counters this by assuming this development is necessarily continuous in nature. When most changes are necessary in a system before initial deployment, or conversely, afterwards, this would indicate support for one assumption while falsifying the other.

*Project Test:* Ascertain whether a project knows more change than initial development activity.

The test will allow the examination of the degree of change and where such change occurs. This gives details on the correctness of the engineering and scientific assumptions.

### Product

The product of knowledge engineering is a knowledge system, and the knowledge model is merely a part of the design to specify that system. This engineering view of knowledge equates a knowledge system with any other artefact: designed, implemented, tested and deployed. The discrepancies between the system's actual behaviour and the specification, caused by errors in the implementation or by changes in the specifications, are reasons to perform maintenance. This contrasts with the view that a knowledge system is a medium for communicating knowledge. In this view, the product of knowledge engineering is an ongoing knowledge model forming a theory of the domain, and a knowledge system is only a medium to communicate that knowledge. The theory will be replaced in time by a more predictive or more elegant theory. In this sense, a knowledge system is constantly changing as its knowledge model changes.

*Product Test:* Determine whether changes to a knowledge system originate in its environment or from the invalidation of the knowledge contained within.

The examination of the origin of the changes to a knowledge model and a knowledge system will provide the information on whether it would be possible to operate in a design centric fashion or to operate more or less based on an exploratory basis. This would discourage or encourage the view of the knowledge system as the medium for knowledge models.

### Process

While the incrementality is a feature of both engineering and scientific approach, they differ on the reasons for using it, leading to variations in the importance of phases, and in the ordering of the phases. The engineering metaphor with an emphasis on risk and cost minimization seeks to divide and conquer, professing no preference for either cyclic or phased incrementality, but most cases in literature show an incremental design and incremental development (see also chapter 2). The successive adjustments and critical feedback proposed by the scientific-based metaphor sees a greater role for cyclic incrementality. This leads to the question whether the preferred way is to perform complete cycles or be incremental within each of the phases or only specific ones.

*Process Test 1:* Establish whether some phases are more important than others are or whether they are equally important.

Posed alongside these assumptions are the notions stemming from the scientific metaphor that the complete cycles matter more, than others are. Therefore, the discovery of knowledge depends on having operational models at every moment in time.

*Process Test 2:* Establish whether cyclic development is more important than incremental division.

The two tests in conjunction provide a possibility to survey the effects of a cyclic process, versus incremental and iterative processes as well as determine the relative importance of the different phases in gaining the required insight.

### **Synergy**

*Synergy Test 1:* Determine to what extent the measures on the different dimensions are necessary.

*Synergy Test 2:* Determine to what extent the measures are influencing one another.

The engineering metaphor would have that different approaches to solving the problem are most likely unrelated as they attack unrelated problems. Within the perspective granted by the scientific metaphor, these are all related to the discovery of insight, and therefore synergy is expected. It is not expected that these measure have to be taken in unison.

#### **3.6.4 Tool Evaluation**

The investigation of the tools, by their different philosophies, is not meant to directly analyse the two metaphors. Both tools are inspired by the scientific metaphor. They aim to elucidate the form of support that should be given to make adjustments in people, project, product and process possible, and thereby facilitate the investigation of the metaphors. The two forms of philosophy can be summarised as simplicity vs. vividness. The implied distinction is that it is not possible or at least hard to allow both simplicity and vividness.

*Tool Test 1.* A tool that can allow knowledge modelling by experts must be simple.

*Tool Test 2.* A tool that can allow knowledge modelling by experts must use a vivid modelling language, to aid the discovery of new insight and support the changeability of knowledge models by experts.

The dichotomy described by the tests above is that between simplicity and completeness of representation required by a vivid modelling language. Increased completeness was thought to further complicate the task of non-specialist users such as the experts whose knowledge was to be modelled. The results show that tools based on either test are usable. Furthermore, use of the first generation tool show definite signs of self-imposed structure, which is constructed out of available materials. This is not unlike the analysis of Clancey (1985, 1983), which showed different types of rules to exist in MYCIN and other knowledge systems.

The approach used to determine an answer to all these questions is seen in the development of two tools that represent extremes in a spectrum in their approach to the problem. These tools provide a realisation of the continuous knowledge engineering method, to provide an implementation of the notions it embodies, as it were. By examination of these tools in practical settings, it is hoped they will show different results, giving insight into the variety of features that are supported. While

this answers, first of all, whether the features are possible to implement, i.e. Realisable, it also allows an examination of the benefits as to their Qualitative and Quantitative aspects.

The rationale behind the study of two tools are used that realise a philosophy rather than examine a specific feature has more than one reason. First and foremost, it is not easy and practically impossible to extricate a single feature and study it independent of the others. The features are interdependent, like schemata in genetic algorithms. It is combination of features in unison provides support for a certain aspect. Secondly, the type of experimental setting to examine a difference in a system is hard to set-up correctly. It is very hard to fix the 'other parameters', for instance the background knowledge and skill of those involved. A third reason is that the opportunity is here to examine two related tools that share common features, but nonetheless possess a different philosophy to knowledge engineering.

### 3.6.5 Bottom-line Evaluation

The question addressed here is whether the global solution, proposed as the scientific-based scientific metaphor and its realisation in the continuous knowledge engineering approach shown in the case studies, solves the economic, practical bottom-line problem. It was determined that an incremental approach inspired by the engineering metaphor was in principle a sound approach to overcome the difficulties of knowledge system development that lead to the uncertain bottom-line. The benefits of the approach depend on the granularity of the increments, the magnitude and cost of each increment and the risk of each increment. It does not address the problems that cause the cost and risk.

The scientific metaphor partially extends the incremental approaches proposed by the engineering approaches. The questions that then remain are whether an insight-based scientific approach is an actual improvement.

*Benefits* – does it increase benefits,

*Cost* – does it lower the cost,

*Risk* – does it lower the risks of development,

*Clarity* - make the bottom-line clearer.

The first three questions are aimed to improve the bottom-line of knowledge system development itself. The last two aim to improve the perception of the bottom-line and provide tools to have checks and bounds on the development. These aim to increase the sense of security of the development itself. The answers given to these questions provide an additional check on the results of this research, as these answers are to be given independent of the previous, which is aimed at making clear the effects of the proposed solution on the scientific problems.

### 3.6.6 Approach

The requirements support the principles set forth in the continuous knowledge engineering approach. The intent is to use the requirements in this chapter to develop two tools, which both implement the continuous knowledge engineering program in different ways, i.e. by implementing different requirements or by implementing them in different ways. These tools should offer certain benefits based on their realisation of the program, and in different ways. The differences between the systems allow the determination of the relative importance of the requirements and their relevance to the

establishment of the principles. This will establish that different realisations can implement the requirements, going towards the Realisation question and provide additional detail into the questions of Quantification and Qualification.

### **A Tale of Two Tools**

The first step then is to ascertain that the tools do indeed implement a number of requirements. In the next step, these tools are applied in practical applications, where data must be collected to measure the effect of the tools. By implementing two tools that are different in philosophy, some information can be gathered about their relative support for continuous knowledge engineering, and by the bias of their implementation gain some insight into the sensitivities of either tool. This allows us to compare the requirements they implement and perhaps say something about their relative importance. By this, the approach can be calibrated somewhat. For the tools, the best way to check is by examining whether some of the requirements are fulfilled and whether the features that implement the requirements lead to the benefits that were supposed to exist.

Therefore, each tool must make clear in what way it implements the requirements. This enables us to see clearly how the tool supports the approach. Then through experiments and applications the benefits of the features can be examined. This lends itself to the validation of the requirements. This also goes to justify the continuous knowledge engineering approach from which these requirements are derived. If the benefits are realised then this lends support to the approach itself.

The first dichotomy between tools that is examined is whether the tools should be very simple or whether the completeness of a tool aids the ease with which modelling can be done, by allowing knowledge models that are vivid. A way of examining this is by looking at two tools that represent these extremes.

In a preview of what is to come, the two philosophies that underlie the tools are discussed in this section. The first tool realises a program of simplicity, the system with many similarities to first generation systems is realised that aims to support experts in modelling their own knowledge. The system provides a primitive domain model and a visual way of formulating reasoning knowledge. Furthermore, extensive facilities are added that allow the development of customised and dedicated applications based on the modelled knowledge.

The second system aims to generate support for knowledge systems in a second-generation fashion. The user support aims to assist the forming of models that are vivid, close in conceptual structure to the experts' own ideas and conceptions. Furthermore, the additional structure facilitates maintenance. Because of the more complex nature, it does pose more of a challenge to non-computer specialists.

This research approach is also given by a healthy dose of pragmatism, as one tool is predecessor of the other. The implicit assumption is that the successor should be better than its predecessor, but considering that one is not merely a continuation of the other, but implements a distinct philosophy should be sufficient caution to give that assumption too much heed. The next two chapters will be dedicated to the description of both tools. These chapters will include descriptions of their support for the continuous knowledge engineering approach.

It should not be forgotten that any knowledge engineering tool is usable in a continuous knowledge engineering approach as formulated in this chapter. Other tools

may be just as apt as the tools in this thesis. Nevertheless, these tools contain combinations of features that are considered complete and consistent in their support for the approach in the sense that they make it ‘reasonably easy, safe and efficient’ to use the approach.

### Case Studies

The case studies are analysed for their results to answer the question whether the approach formulated in this chapter truly constitutes a solution. The case studies will be executed to validate both the tools and the continuous knowledge engineering method. Caused by the difficulty to perform experiments in knowledge engineering, and by the constraints of the practical environment in which the research is taking place a number of practical projects have been selected where these tools are employed. In some cases, these do constitute prototyping projects to validate the systems functions and its abilities for certain tasks.

The projects employ the tools and in some cases implement the continuous knowledge engineering program from a methodological standpoint. As these project represent actual real-life problem situations the evidence they supply is realistic, however at the same time this has restricted the possibilities for measuring, or even the implementation of a project directly along the lines of the proposed approach. In many cases, the project have therefore been analyses *post facto* to their effects and evidence for benefits.

From the battery of projects, examples are shown of expert modelling and expert participation in modelling, continuous development, direct application and customisation, and insight gained through modelling and experiences with the system. The evaluation of the proposed solution will be done by defining a number of experiments that will be used to evaluate the success of results of the implementation of the two tools and their demonstration in a number of practical situations. By extension, this allows a verdict on the two perspectives that are proposed by the two metaphors.

## 3.7 Conclusions

This chapter has defined an alternative view of knowledge system development in continuous knowledge engineering. The approach is based on the scientific metaphor as inspiration for knowledge engineering and provides the appreciation of the fluid nature of knowledge and its origin in learning. It shows focus on building and changing knowledge systems, as proposed by the *weak situated cognition* position.

Three principles were proposed to enable that learning: stakeholders, project stewardship, knowledge system as a medium for knowledge models and cyclic development. The unity of these principles moves towards a situation where knowledge systems are developed continuously over an extended period, in majority by the experts, with typically frequent deployment of iterative changes to the knowledge model. The changes needed to create this situation, require that support is given by tools to implement these changes.

Primarily, this chapter has described several conceptual steps were taken to move towards realisation of the continuous knowledge engineering approach. First, the principles were used to come to a number of requirements for tools to make the approach viable and practical. Furthermore, a number of candidate techniques to



realise the tools were discussed. Finally, a number of evaluation criteria were presented, allowing the testing whether the tools conform to the requirements and support the principles, and thereby support the continuous knowledge engineering approach.

Following the identification of the problem and design of a method of solving it, the next chapters will show implementations of tools based on the program described in this chapter, and demonstration of the use of these tools in practical and experimental settings. These describe the concrete steps taken on the path towards continuous knowledge engineering.



# Chapter 4

## A Simple Tool

---

*...simplicity is the ultimate form of sophistication...*

*– Leonardo Da Vinci*

The tool described in this chapter is a simple, limited system for knowledge modelling and knowledge system development. This system realises requirements stated in the previous chapter. It employs a visual knowledge representation, provides facilities for centralised deployment and allows component-based development techniques. KBE has been applied in a series of practical projects and thereby forms the basis of a number of real world applications.

The first section of this chapter offers an overview of the KBE, providing an elaboration of the philosophy behind the system, the architecture of the KBE and the system's key features. In the following section examines the internal knowledge representation constructs as well as the inference mechanism that the system uses to apply the modelled knowledge. In the third section, the basic methods to model knowledge using the KBE is deliberated, with special attention for aspects to be used in evolutionary development strategies. The fourth section the KBE supports use of a knowledge modelling a knowledge system or as part of a larger system. This shows the customisation of the presentation of the system to the user to create both default and domain-specific systems. The fifth and final section recapitulates these subjects, and highlights the primary characteristics that will come under consideration when the system is applied.

### 4.1 Simplicity versus Completeness

The overview in this section presents the KBE. Before delving into the system itself, it is important to understand the basic philosophy of the system, which values simplicity over completeness. Using this philosophy as a perspective, the functionality is discussed in general terms, followed by an examination of the architecture. Finally, a description of some of its key features is given.

#### 4.1.1 Philosophy

Many researchers and developers have over the years researched different approaches to the development of knowledge models and knowledge systems. The challenge often was to represent some grave and complex problem in a certain representation. Alternatively, the challenge was in the difficulty to elicit the knowledge itself, the sheer scale of the knowledge, or the need to constantly verify and validate the completeness of the knowledge with mathematical precision.

The KBE takes a different view on the problems in knowledge engineering. Most knowledge problems in existence today do not concern space shuttle repair or brain surgery. They concern knowledge that is not accessible to people or that they cannot apply in the form that it is in. This knowledge is implicitly available in a single person or distributed over a group of people. These problems require a consolidation effort to make this knowledge explicit, bring the knowledge together in a central repository

and transform it into a usable form (Steels 1992, 1986, Stefik 1995, 1986). Note that these issues do not require that the knowledge itself must be elaborate, complex or necessarily involves great amounts of modelled knowledge.

A tiny knowledge base that many users can have access to can be an enormous economic benefit to an organisation and to people in general. An Internet advisory knowledge system that can tell people what to do if their child swallowed an object and which answers 19 out of 20 that they should take the child to see a doctor will probably save as many or more lives than MYCIN ever did. The problem therefore is to bring the possibility to develop and access knowledge systems to a great many people.

The aim should therefore be to make it easy to develop and use knowledge systems for simple knowledge. In this context, simplicity is seen as a virtue, rather than a lack of sophistication. The only consideration that such simple, but effective knowledge systems must submit to is to the usability of the system. The knowledge system must offer the same quality and ease of use that people have grown accustomed to in standard software. This means that the post-production facilities for knowledge system development must offer the possibility to create a high-level quality product.

The remainder of this section will provide additional insight into how this philosophy has been realised in functionality that the KBE system offers and how this was made possible by the underlying architecture.

### 4.1.2 System

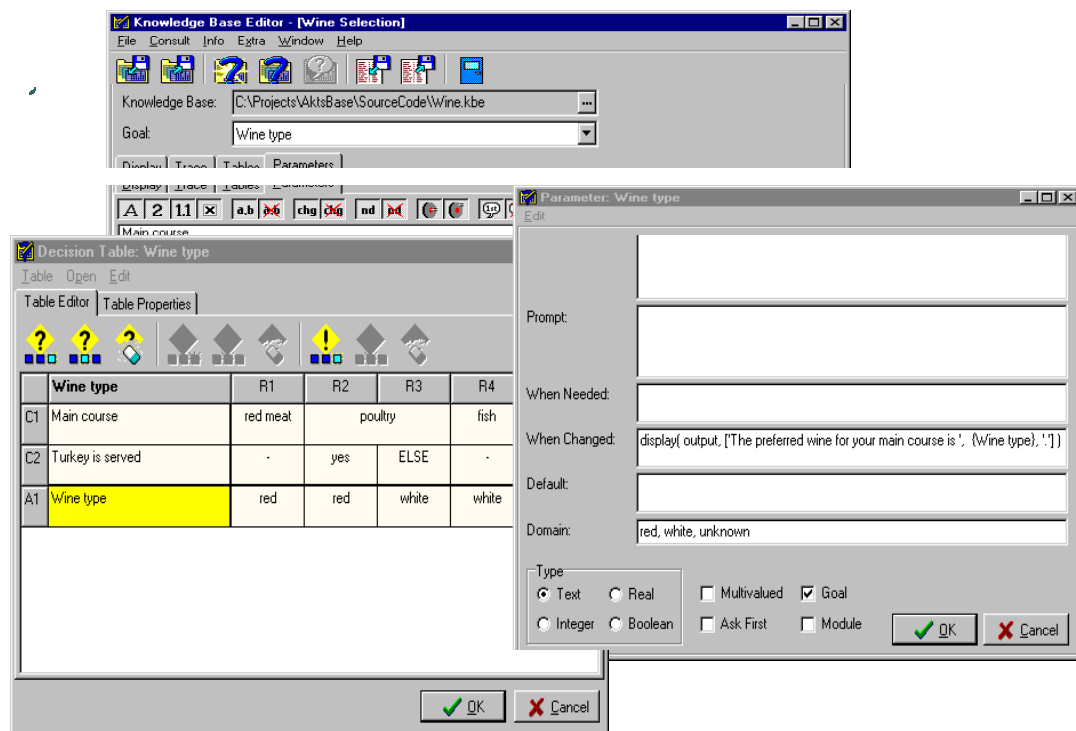


Figure 4-1 KBE User Interface

The KBE is an extension of an earlier system called the Advanced Knowledge Transfer System (AKTS). This system is a front-end to a Prolog engine that allows entering decision-tables and translates these to Prolog facts and clauses. Initially, AKTS was restricted to a knowledge acquisition tool, but the system eventually

allowed the consultation of the knowledge models (Lucardie 1994, 1992). The KBE is its successor, independent of a Prolog engine, and enhances the systems capabilities for knowledge modelling and knowledge system development. The remainder will make no distinction between AKTS and KBE, as the KBE subsumes the facilities of AKTS.

The KBE is a knowledge modelling tool based on the philosophy of simplicity. It does not have an elaborated domain-model and its inference mechanism is based on simple backward chaining. The basic technology contained in the system is therefore not an elaboration in a technical sense, rather it can be perceived as a simplification of the technical possibilities that can be discerned in many other tools. This simple tool is coupled with visual representation of the knowledge in the form of decision tables and extensive facilities to deploy the knowledge models, as standalone applications and through web-enabled user interfaces. Furthermore, these deployment facilities are highly configurable and can be used to easily develop dedicated applications.

The KBE is a system for the capture, maintenance and consultation of knowledge. The tool provides a user-friendly point and click environment that is intuitive and easy to use. Using the system requires no programming skills whatsoever. The system represents the domain as attribute-value pairs using only primitive data types and uses decision tables as the representation of the knowledge itself. The visual nature of decision tables is easy to understand and assists in the validation and verification of knowledge.

- Visual Knowledge Representation
- Component Based Development
- Model View Controller
- Abstract Communication
- Centralised Deployment
- Executable Specification
- Simple Knowledge Model
- Simple Inference Mechanism
- Adaptable Visualisation

**Figure 4-2 Key Features**

The KBE allows non-specialists and knowledge engineers alike to create knowledge models, verify and validate them, and maintain the represented knowledge. Furthermore, the use of decision-tables enables strategies that allow a knowledge system to grow continuously and evolve over time. Decision-tables allow for easy extension, both top-down and bottom-up. This means that it is easy to start a small but useful system and develop it incrementally.

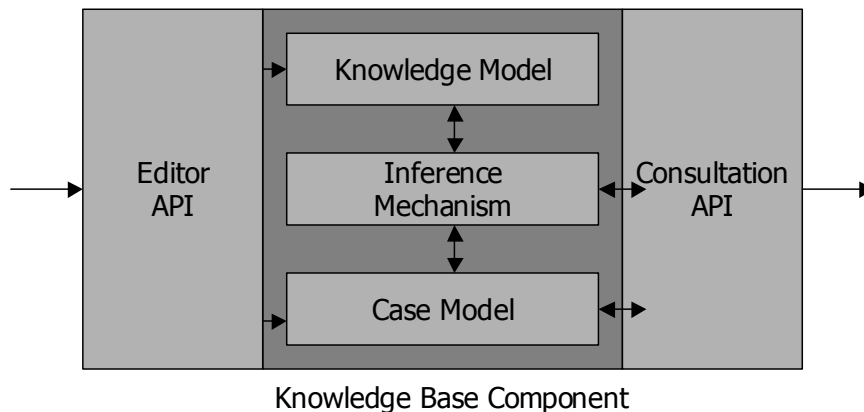
Beyond the modelling functionality, the KBE has a number of sophisticated tools to publish the knowledge models and make them available to the end-users. These tools make it possible to use the knowledge model in a default user-interface, but also using simple customisable interfaces based on HTML, or by engineering dedicated domain-specific systems.

While having a commitment to simplicity of use, the requirements that were stated in the previous chapter stand as functionality that must be realised in order to support the continuous knowledge engineering approach. The Figure 4-2 Key Features an overview of the features that the KBE implements. Some of these will be described at greater length in later sections of this chapter.

### 4.1.3 Architecture

The KBE family's architecture is highly component-based, which allows the development of both domain-specific knowledge-modelling tools as well as knowledge-consultation systems. A central component handles all the actions on a single knowledge model, and is equipped with additional facilities and extensions for other functionality. Based on these components several applications have been devised as default design and consultation environments. First, the knowledge base component is described, after which the remainder of the architecture is elucidated. Then a number of applications based on the component are discussed.

#### Knowledge Base Component



**Figure 4-3 KBC Basic Architecture**

The architecture is based on that found in many different kinds of knowledge systems (compare Figure 2-3 Knowledge System Architecture). The knowledge consisting of a factual knowledge, a reasoning knowledge and inference mechanism is contained by the knowledge base component. These will be described in the sections below. The knowledge base component does not contain any visualisation aspects, much like a database. The visualisation of either modelling or consultation is considered a separate issue. This allows integration of the component in any kind of software environment, examples of which will be discussed later. One effect of this is that it makes the development of dedicated systems possible.

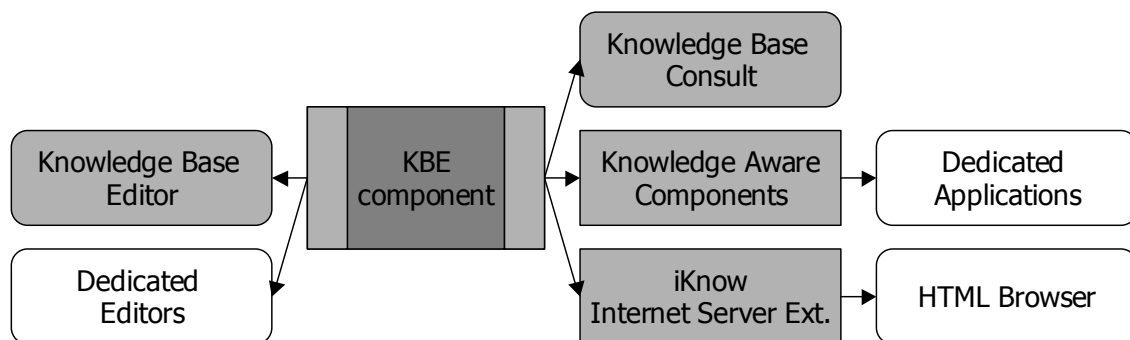
The advantages of CBD, described in the previous chapter, is the great modularity of different aspects of a system, and their exchangeability with other components that realise the same functions, in another way. The architecture requires a knowledge editor and a consultation of a knowledge model, and may include other components to be integrated, such as databases, etc.

The API of the knowledge base component is simple. It allows modification actions on the knowledge model elements as well as consultations actions, supplemented by a general introspective API, giving access to the different sub-components of a knowledge model. The component-based approach yields the possibility to employ

virtually any kind of visualisation, medium or interface. The different modalities are represented in the architecture. Whether it concerns a standalone application developed around interactive forms or by using a sophisticated web-server extension, the possibilities abound.

The component has a number of methods that arrange for an abstract model of communication. This has as an advantage that several different facilities can handle the visualisation. The knowledge model itself can be developed using this abstract communication rather than worry about the specifics of any of the representation forms.

### Component-based Architecture



**Figure 4-4 Knowledge Component Architecture**

The architecture of the different systems that comprise the KBE family centres on the knowledge base component. The additional facilities and extensions are mainly aimed at visualising the knowledge model and develop a knowledge system based on a knowledge model. A number of other components were added to support the development of knowledge modelling and consultation systems, as well as a default visualisation of the modelling environment and consultation system. Additionally a series of components was developed to allow knowledge systems to be deployed through an Internet web-server.

The abstract component does not contain all functions that are essential for developing knowledge systems. In its bare form, there is not way to pose any questions, and provide a user with a final result, perhaps as a report. To the component, these are auxiliary issues, which allow anyone to add these visualisation functions to the component, as well as others that have yet to be contemplated. The system's architecture includes a number of auxiliary facilities for developing knowledge systems based on a knowledge model.

Other components can be added in time to the framework, as some have already been done. A default knowledge modelling environment has been developed, as has a default consultation environment. Additional components include so-called knowledge aware components, allowing very fast development of dedicated user-interfaces. Other components allow visualisation of a knowledge system through an Internet server as HTML pages.




### Knowledge Aware Components

This first category of additional support was implemented by providing for a set of visualisation components. The integration of the knowledge base component and the knowledge aware components enables developers to use rapid application

development environments such as Borland's Delphi to quickly and easily create every possible type of application. These simple components make it possible to develop a graphical user interface for a knowledge system very quickly. They obviate the need for programming this integration.

The knowledge aware components are visualisation aids that can be used in conjunction with the knowledge base. Such a component like an edit field can be linked to a parameter in the model, by virtue of which link the value represented by the component is the value of the parameter. Changing the value in the edit component, changes the value of the parameter.

Each of these components is integrated into the Delphi programming environment. By using a true point-and-click interface, these components can be linked to one of the parameters in the model, allowing them to display and in some cases change the value for a parameter.

Name	Visual	Description
KBEdit		Display and modify Values through open questions.
KBCombobox		Display and modify values through closed questions
KBLabel		Display messages, prompts, explanation and values of parameters.

**Table 4-1 Knowledge Aware Controls**

Any one of these components needs only to be told which of the parameters in a knowledge model it belongs to. It then is ready to visualise and modify the parameter during a consultation. Changes may propagate to other parameter values, under the inference mechanism. These changes are reflected immediately in these components.

### Internet-based Deployment Components

To support centralised deployment and allow for a scalable and customisable solution, Internet-based deployment components were developed as an additional framework. This framework consists of visualisation and reporting facilities. The former visualises the questions posed to the user of the knowledge system, while the reporting facilities generate a more insightful end-result based on a knowledge model.

The Internet capabilities of the system enable centralised deployment of knowledge systems to distributed users. The cost per additional user is negligible compared to other schemes and makes it very easy to release systems based on small changes. In fact, such changes are transparent to the end-user.

The iKnow Server Extensions supports the use of a knowledge model in an Internet server environment. These components serve out HTML pages and receive answers through the HTTP protocol. This allows many different users around the world to make use of the knowledge system at their convenience. These components give many possibilities to customise a knowledge system. The system employs a set of templates that are used to generate different screens. Changing these templates or adding new ones can change the look-and-feel in any way.

The first option is to change the knowledge system's surface visualisation, such as the colour scheme. This can include company logos, etc. The next stage includes a



number of specific templates to be added to a knowledge system, to visualise certain specific questions. For example, a question on colour can be visualise as a colour map, rather than a closed question giving a textual description of different possible colours. This type of customisation can be added incrementally.

To visualise the results of a consultation it is standard to deliver a concluding report. The iKnow Report Extension allows the definition of an XML (Extended Markup Language) file containing a template report that gives a specific report for the situation described in the knowledge model during a consultation. In some cases this report is created during the consultation, a so-called running report, in other cases this is limited to an end-report. Such reports are also in many cases what an expert would have to fill in or deliver as the conclusion of the involvement in a problem case. This report generates HTML to be integrated into the consultation pages generated by the Internet Extensions.

### **Applications**

The KBE itself is based on the design and consultation API provided by the component. It provides a possible visualisation of the knowledge model and allows modification of its content. It is default editing interface placed on top of this component. Dedicated environments can also be developed in much the same way. The component-based approach further allows easy integration of this knowledge technology as a small part of a larger system. As such, knowledge systems have been integrated into document information systems and geographical information systems (GIS). A consultation only version Knowledge Base Consult System (KBCS) also exists making it possible to deploy a default standalone version of any knowledge model. A special file format has been developed for this to encrypt the knowledge itself, as in many cases the knowledge itself can represent considerable value. Furthermore, a default Internet Interface has been developed to allow the consultation of any knowledge model over the Internet with no other effort than the development of an initial knowledge model.

These three default systems allow the development and deployment of a knowledge system to take place with minimum effort. At the same time, different paths are provided to move from a default system to a customised system, through to a full-fledged dedicated visualisation of any knowledge model.

## **4.2 Modelling Primitives**

In this section, the different aspects of the knowledge representation employed in the KBE are highlighted. This is a more detailed description of some of the aspects introduced in the section above. The subjects that will be treated are the domain model, the inference model and the inference procedures. This serves to clarify the constructs as they are used in a knowledge model.

### **4.2.1 Simple Knowledge Model**

The model of knowledge that the KBE employs is intentionally simple. It only has three concepts that require understanding: the decision table, the parameter and their dynamic relationships bound up in the backward chaining inference process. The simplicity means two things for the user: there are not a lot of choices to be made and the choice for representations of knowledge are rather limited. Understandably, this has both its benefits and its disadvantages.

The decision-table is an example of a visual knowledge representation, as was described in the previous chapter. The use of decision-tables in this system assists different kinds of users, in understanding, locating and changing the modelled knowledge at a lower level. Decision-tables group a number of related rules, and thereby provide an element or concept at a higher level of abstraction than unstructured rules.

This system knows no mechanisms for dividing a knowledge model into separate modules, and no object oriented modelling concepts are supported. Therefore, some of the advantages mentioned for aspects to do with participation in knowledge modelling are not realised.

### 4.2.2 Parameters

The screenshot shows a dialog box titled "Parameter: Wine type". The dialog is divided into several sections:

- Title:** Wine type
- Explanation:** (Empty text area)
- Prompt:** The knowledge base has no preference for any type of wine. What would you prefer yourself?
- When Needed:** (Empty text area)
- When Changed:** display( output, ['The preferred wine for your main course is ', {Wine type}, ''] )
- Default:** (Empty text area)
- Domain:** red, white
- Knowledge Base:** (Empty dropdown menu)

At the bottom of the dialog, there are several options and controls:

- Type:** Radio buttons for Text (selected), Real, Integer, and Boolean.
- Options:** Checkboxes for Multivalued, Ask First, Goal (checked), and Module.
- Precision:** Input box with value 0.
- Decimals:** Input box with value 0.
- Group:** Input box with value 0.
- Buttons:** OK (with a green checkmark) and Cancel (with a red X).

**Figure 4-5 Parameter Specification**

The domain model part of the knowledge representation is formed by a set of parameters. A parameter is in essence an attribute-value pair. A parameter also has additional information describing it and determining some of the aspects of its usage within the system.

Name	Description
Title	The name to be used when referring to the parameter. This name may contain spaces
Prompt	The prompt is used to formulate the question to the user.
Explanation	The explanation is used as an additional description to augment the prompt.
Default	The default value is the one assigned to the parameter when it is required. This can serve many purposes, but is especially useful when debugging
Domain	The domain limits the values of the parameter
Type	The type of the parameter, where the value can be one of four base-types: String, Integer, Real, and Boolean (see Table 4-3)
Goal	Determines whether this parameter may be used as a goal for the knowledge model.
Ask First	This determines whether the parameter should first be presented to the user, before attempting to infer a value. In this particular case, the user may answer 'don't know'.
Multi-valued	Multi-valued parameters can attain more than one value. For instance, if the domain is 'red, green, blue, yellow', and the parameter is multi-valued, it may contain the value 'green, yellow', or 'red', or 'green, blue, red', or any other combination of the domain values.
Precision	Determines the precision of calculation and storage of a real valued parameter.
Decimals	Determines the number of decimals for calculation and storage of a real valued parameter.

**Table 4-2 Parameter Settings**

The most important attributes of a parameter are shown in Table 4-2 above. The remainder of the attributes shown in Figure 4-5 are either advanced concepts or have been depreciated.

Type	Description	Example Domain	Example Values
String	Open text	none	name = •John Smith•
	Any value from an enumerated domain	(red meat, poultry, fish)	course = •red meat•
Integer	Any whole numbered value	[0, inf) [1,10]	number = 6
Real	Any real numbered value	[-273.15, inf)	temperature = 25.5
Boolean	Any truth value	(true, false)	vegetarian = true

**Table 4-3 Domain Model Parameters**

String parameters can be given a domain, which enables it to operate as an enumerated value. Boolean parameters can be configured as to what term is used to denote true and false. Each parameter further has a prompt, an explanation, a domain and a default value. The prompt and explanation are used to query the user for a value when the inference cannot yield a value for the parameter. The domain can restrict the possible values that a parameter can accept or state the possible enumerated alternatives that can be chosen for the parameter.

## String

String parameters are parameters that contain a text value. Usually these parameters have a predefined domain, for instance, a parameter which contains a colour, can have a domain "red, green, blue, yellow". If a text parameter has no domain, it can contain any value. The string type can hold any value that is an array of characters. The text type plays a double role in the KBE. If it is not used in a decision table as a condition then it has no restrictions on the values it can assume. It can then be used to hold auxiliary information like the users name, the name of the object under diagnosis, etc. If it has an explicit domain given by definition, or if it is part of a decision-table's condition-alternatives, it can assume an implicit domain defined by local restrictions on its values during consultation. In those cases, it functions as an enumeration. The parameter can then only take values from the domain. In these cases, it is used to enumerate for example the wines of choice: Bordeaux, Merlot, etc.

## Boolean

The Boolean type is the well-known logical type that is used to specify values that can be either true or false. The domain of a Boolean parameter is determined by that fact. Boolean types are used to specify yes-no situations such as *'Is the guest coming to dinner a vegetarian?'* Boolean parameters are similar to text parameters, but they have a predefined domain. The KBE by default makes this the domain "Yes, No", but you can override those values using the knowledge base properties. Besides that, "Yes" is equal to TRUE and "No" is equal to FALSE.

## Integer

A parameter of the integer type can accept any whole-number value. It can be supplied with a domain limits the values it can assume. The domain for an integer is not a list of values, as with text parameters, but a range of values. The domain can be specified as a language expression, or as a range. If the domain should be all integer values greater than 3, the domain is  $X > 3$  (the X is used to refer to the parameter value itself, so  $3 < X$  is the same as  $X > 3$ ). If it is all integer values less than 9, the domain should be  $X < 9$ . If it is all integers between 3 and 9, the domain is  $X > 3, X < 9$ . You can use  $\geq$  and  $\leq$  to express "greater than or equal to" and "less than or equal to" respectively. You can only express a single interval, so, a domain combining of the values between 3 and 9 *and* the values between 22 and 37 cannot be defined.

## Real

The real type can accept any broken-number value. It can be supplied as the integer can with a domain that functions as a limitation on the values it can assume. The real can be used to declare parameters that hold the temperature in degrees Celsius, with a domain of  $[-273.15, \text{inf})$ . Real parameters have real values. These are handled in the same way as integer values. For a decimal point, a dot must be used (e.g. 3.14156). As was discussed before a real-valued parameter may be equipped with a precision and number of decimals.

### 4.2.3 Decision Tables

The knowledge model further consists of a set of decision-tables. The decision-table has been around for some time and has been used for knowledge representation in other systems as well (Mors 1993, Montelbano 1973).

	Wine type	R1	R2	R3	R4	R5
C1	Main course	red meat	poultry		fish	ELSE
C2	Turkey is served	-	yes	ELSE	-	-
A1	Wine type	red	red	white	white	ASK

**Figure 4-6 An Example Decision-table**

The left top part of the table contains the conditions, below which the actions are located. These contain references to parameters in the domain. The right side contains a tree of condition-alternatives and action-alternatives. The condition alternatives and action alternatives can contain expressions yielding a value. This can be simply a value in the domain of the parameter or a reference to another parameter. Special values include the DON'T-CARE (-), which means that the condition is not relevant and can be ignored, and the ELSE alternative, meaning the remainder of the domain. Another special keyword here is ASK which can be added as an action alternative. When this action-alternative is executed, the question to ask which wine type is required is forced, as in that situation the knowledge system has no preference.

The condition parameter is compared to the active alternatives; for example, Main course is compare the values red meat, poultry, and fish. The chosen value is used to determine the next set of alternatives, below the chosen alternative, as visible in the figure. If there is a don't care, the condition is skipped and the condition alternatives in the row below are chosen to be executed. When all conditions have been evaluated a column of the table's action alternatives is selected, and these actions are performed, in this case assigning a value red to Wine type. By virtue of this visual representation, it is easy to examine the contents as to their completeness and exhaustiveness. Therefore, it is easy to whether one of the required alternatives is missing or that there are no actions associated with a particular configuration. Furthermore, it is easy to add additional knowledge at any point in the decision table, as an additional condition, condition alternative, or the consequences of the table's execution can be extended with additional actions.

A decision table can be perceived as an implementation of *heuristic classification* (Clancey 1985). It uses a number of features in the domain, associated with a number of conclusions, not necessarily through causal links. A knowledge model in the KBE is a network of heuristic classifications.

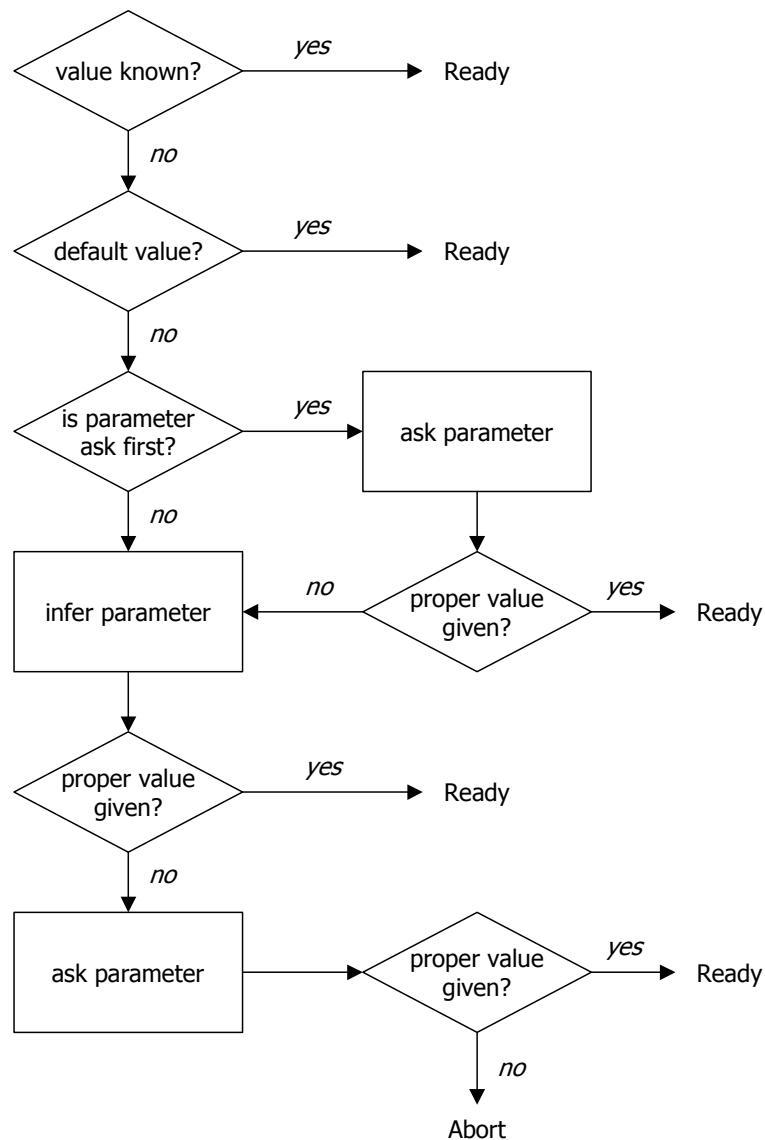
#### 4.2.4 Inference Process

The inference process used by the KBE is simple backward-chaining reasoning. There is no way to influence or change the inference process directly. The only control by the user is implicit in the model. To exert this control requires modelling approaches that explicitly incorporate control structures. These approaches will be discussed later on in this chapter.

##### Evaluate Parameter

The inference process commences by attempting to determine the value of a goal parameter (see Figure 4-7). Any one of the parameters in the knowledge model can be marked as goal parameter. The consultation process can be halted at any time, by the user or by any other outside agent. This gives the possibility to examine the facts of

the knowledge model, save the consultation for later, retract one of the earlier answers or stop the consultation.



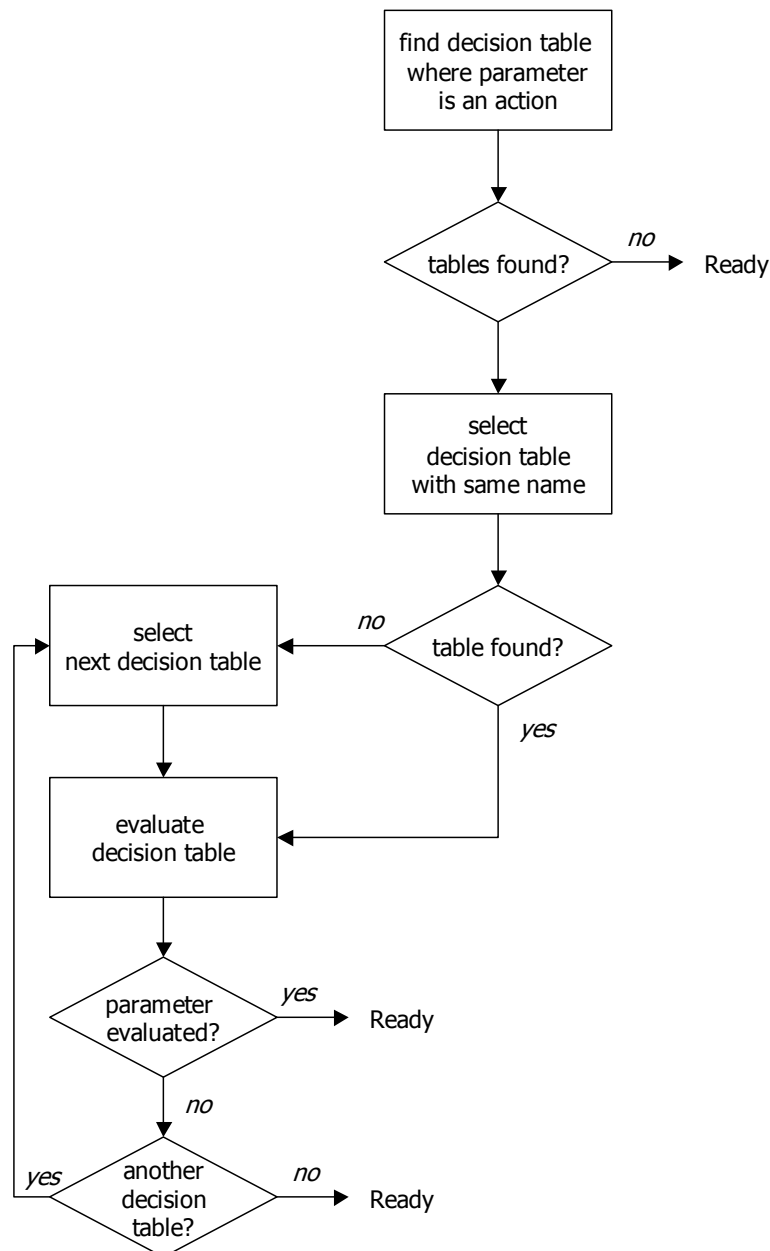
**Figure 4-7 Evaluate Parameter**

The system works by deepening queries, attempting to derive the value of a parameter with the available knowledge, which may require inferring other parameters. The process is the same for the goal as for any sub-goal parameter. For example, if the value for a parameter X has not been set yet, the system attempts to determine it. First option in the inference process is to assign the parameter X a value is the default that has been set. If there is a defined default then that value is returned. In some cases, this may give rise to further inference by virtue of the replacement expressions that may be present in the default (See replacement expressions). If there is no default, the inference proceeds. In the next step, the system sees whether the parameter is defined as ask-first, and if from this user interaction a value is returned, then the system is ready and returns the value. If this user interaction does not yield a value, because the parameter could not be asked first or the user's answer was *don't-know*, the inference continues.

The system then attempts to infer the parameter, using those decision tables that have the parameter contained in one of the actions. There may not exist such decision

tables or when used they may lead to a situation where the decision table does not assign a value to parameter X. As there may not have been a decision-table, or the decision-table did not produce an answer, then there is another possibility to ask the user for the answer. This may be the second time that this question arises. This time however, the user is not allowed to enter *don't-know* as an answer. After this, the value is either known or the consultation aborts.

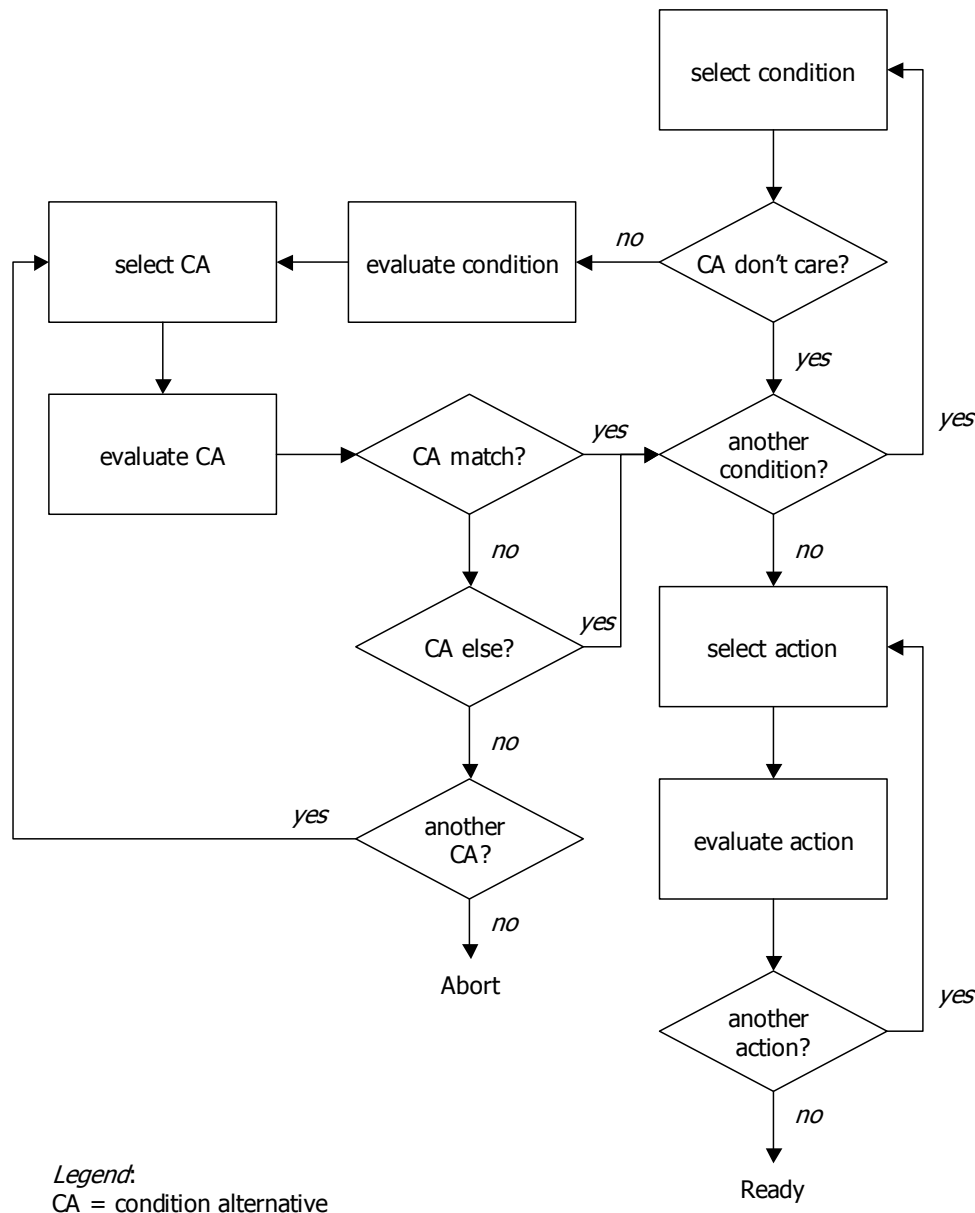
### Inferring a Parameter



**Figure 4-8 Inferring a Parameter**

The inference of a parameter is performed during the evaluation of a parameter. When the decision-tables are consulted, it is important to remember that there may be more than one decision-table that can assert a parameter. In that case, the decision-table with a name equal to that of the parameter is used first. If this does not lead to a value for the parameter because none is specified in that specific case, the remaining decision-tables are consulted in order of the list of decision-tables.

### Consulting a Decision-table



**Figure 4-9 Evaluating a Decision-table**

A condition is evaluated by first checking the expression of the left-top condition alternative to see whether it contains a *don't-care* symbol. If it does then the consultation continues to the following condition; otherwise, the current condition 's expression is evaluated.

Reaching the evaluation of a condition, the resulting value is then successively compared to the value yielded by evaluation of the current condition alternative expression. If the condition alternative has specified variable replacements (e.g.  $X > 10$  or  $X > \{\text{temperature}\}$ ) in its expression, as references to the result value of the condition, then the value of the condition is substituted into the expression. Otherwise a normal comparison of the condition and alternative values is performed. This comparison yields either true or false. If the values match, then the tree of condition alternatives under the current condition alternative is selected, and the system proceeds to the next condition. This continues until all the conditions have



been evaluated. The column of action-alternatives under the last selected condition alternative contains the selected actions.

These actions are executed in order, combining the parameter in the action expression and the value is the action alternative. The action and action alternative may contain a valid assignment statement or they can contain a COMMAND expression. The first leads to a value to be assigned to the parameter mentioned in the action. The COMMAND expressions leads to some action being taken on the model or on the outside world.

### ‘What if...’ Analysis

One of the aspects of the inference process is the possibility of performing a ‘what if...’ analysis. The system allows users to change the value of a single parameter and explore an alternative interpretation of the problems situation. This revokes all inferred parameter values and all answers given to questions after the revoked parameters timestamp. This leaves only the answers that were provided by the user until the point at which the original answer was given.

After this operation, the consultation is restarted, using the remaining answers for questions that may arise. The user therefore does not answer these questions again. The system automatically ends up at the place where the revoked parameter was asked, giving the user a chance to change the answer previously given.

### 4.2.5 Communication

As is clear from the inference process, communication with external systems is sometimes required. Although the system has a bias towards a human user using a prompt and explanation, this does not negate the possibility to use other sources of information, such as databases, active sensors, etc. The knowledge base component fulfils all possible modes of communication discussed in the previous chapter.

#### Obtain

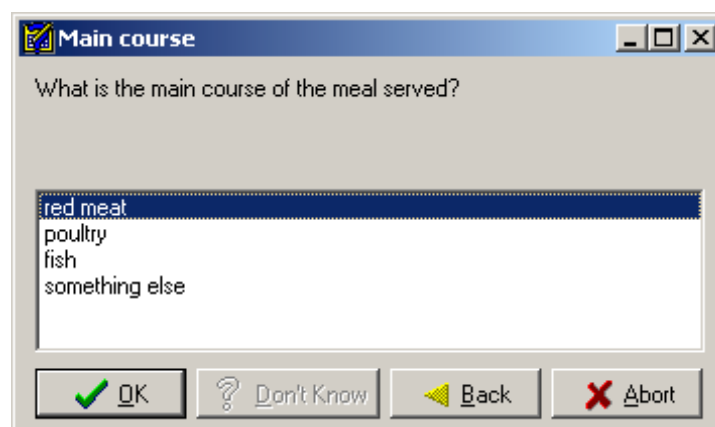


Figure 4-10 Q & A Dialog

At different points in the inference process, questions to the user are directed to obtaining a value for a parameter. This employs the prompt, explanation, and the domain to create a basic question format. When a parameter has no alternatives and the values used in the decision table from which the question originated are used. This negates the need to enter a domain for all parameters.

It is possible to direct the system to order the questions posed to the user. First of all, a parameter can be made *ask-first*. This means that the first time that the parameter is evaluated, instead of using any available decision-tables, the parameter is first presented as a question to the user. If the user answers the answer is unknown to him (the 'don't know' button in the dialog), the system continues with the inference process as before. The purpose of this is that experienced or knowledgeable users can provide high-level answers, which negate the need for some inference, and the many questions that can be related to it. For inexperienced or novice users this can create the possibility to have difficult questions inferred or approximated by the system. An example would be a question to the temperature in the room. If the user does not know, it can be approximated by knowing the season. Using this as an estimate, the system can continue.

	<b>Wine type</b>	R1	R2	R3	R4	R5
C1	Main course	red meat	poultry		fish	ELSE
C2	Turkey is served	-	yes	ELSE	-	-
A1	Wine type	red	red	white	white	ASK

**Figure 4-11 Explicit ASK Alternative**

Another capability is to explicitly ask a specific parameter from a condition-alternative in a decision table. This allows a certain question to be asked at the time the decision-table uses a certain condition-alternative or action. When explicit control is needed to change the order of questions, this mechanism can be employed. The inference mechanism sometimes asks questions in an order that is unintelligible or seemingly illogical from the user's perspective.

This is related to the practice of users who try to guess at the inference process, and according to their own reasoning necessitates a certain order in the questions asked. The system may have determined a more efficient order, or has determined that a question is irrelevant in the current context. In certain cases, these questions are explicitly incorporated in the knowledge model to appease the users. While this may seem wrong, the choice to do so may be very important in the acceptance of the system.

It is an important mechanism to incorporate a question for a parameter explicitly, because the knowledge does not provide for certain situations (yet), as in the table above. The ELSE condition means that some main course unknown to the modeller is selected, and determines that the system might as well ask the user for the type of wine to be used during the meal.

### **Present**

Certain parts of the reporting are produced during the consultation of the system, by explicit commands within the knowledge model.

	dinner report	R1	R2
C1	dinner	INCLUDE fowl	ELSE
A1	COMMAND	display( output, "fowl is served." )	display( output, "no fowl is served." )

**Figure 4-12 Display Command**

The COMMAND display as shown in Figure 4-12 is the KBE's equivalent of present mode of communication. The text sent in a display command uses HTML syntax, so a great variety of ways exist to present elements from the model.

### Query

Any parameter in the knowledge model can be accessed and queried for its value. The parameter allows access to the current value or its evaluated value. The latter may start an inference process when the value for the parameter is not yet evaluated. In this way it functions as a goal.

### Receive

Likewise, it is possible to access any parameter in the knowledge model and set a value for it. These values are checked for conformance to the domain that was set for it. This mechanism may be used to set a host of parameters before the consultation is started. This is often the case when dedicated knowledge system with an interactive user-interface are integrated with the knowledge model.

## 4.2.6 Language

The KBE knows a limited language for creating expressions that can be used in different parts of the system. The explanation here will be concise as the previous showed many examples. It also explain replacement expressions.

### Expressions

Type	Name	Examples
<i>Literals</i>	String	"A string", "foobar"
	Integer	10
	Real	1.2
	Boolean	TRUE, FALSE
<i>Operators</i>	Numeric	*, /, +, -, ^
	Logical	OR, AND
	Comparison	=, <, <=, <>, >, >=
<i>Symbols</i>	Evaluation	UNKNOWN, KNOWN
	Decision table	ELSE, ASK
	Replacement	{<name-of-parameter>}, X
<i>Functions</i>	Call command	COMMAND name(par <sub>1</sub> , ..., par <sub>n</sub> )
	Mathematical	sin, cos, min, max
	Date& Time	date(dd, mm, yyyy), time(hh, mm)

**Table 4-4 KBE Language**

The expressions in this language contains commands, symbols, operators, functions, equations, assignments, text display, picture display, parameter logging. Some examples are shown in the Table 4-4 above.

These expressions are found in many different places. Decision tables are an obvious location, but the default can also be such an expression. Another issue in this respect is the capability to parse replacement texts from a parameter into a string, for example into a prompt. Even though the language is extensive enough to support basic math functions, the set of functions that is considered the minimal set is quite small and self-contained.

### Replacements

As was seen earlier in domains,  $X$  can be used to refer to the value of the parameter. This also operates in the decision table. On execution the value in the parameter is substituted for  $X$ , after which the actual comparison is made. In fact, the normal form of the decision-table is a short notation for this principle. In actuality it should say  $X = \text{"red"}$ , instead of just  $\text{"red"}$ .

In condition- and action-alternatives, it is also possible to use replacement expressions, of the form  $\{\text{<name-of-parameter>}\}$ . This is then replaced by the value of that parameter. This can also be used in prompts, replacing the reference to a parameter, changing "Is the owner  $\{\text{'name of owner'}\}$  living in the dwelling at  $\{\text{'address of building'}\}$ ?" into "Is the owner Jack Smith living in the dwelling at 4 Main Street, Hicksville?". This allows the system to tailor its questions and responses based on the content of the knowledge model. To be able to parse the original text, it is necessary to evaluate those parameters. Therefore, these kinds of replacements can change the natural order of inference. Another replacement expression could be  $X < \{\text{room temperature}\} / 10 * \{\text{alpha\_factor}\}$ , found in a decision table.

### 4.2.7 Conclusions

The full spectrum of possibilities that makes up a knowledge model in the KBE remains to be a set of parameters, with related decision tables and a backward chaining inference engine. The language that is used to formulate the different expressions is quite simple as well. The inherent complexity of navigating and understanding such a knowledge model is relatively low. There are only a few concepts that are required material and the learning curve has been shown in practice to be very limited indeed.

## 4.3 Knowledge Modelling

This section describes the manner in which the KBE can be used to develop a knowledge model. This takes the form of a scenario. From the start to a developing system, different aspects of modelling are shown, including later changes and enhancement.

### 4.3.1 Starting off

In many cases, developing a knowledge model is an exploration, rather than a development along a predefined path. This means knowledge modelling can take many forms, but is most commonly an iterative process, proceeding either top-down or bottom-up, or by using mixed approaches. The most important feature is that the minimal system that can be consulted is very small and that subsequent changes are small as well. A single decision table and a couple of parameters are sufficient for a first implementation and some trivial systems require no more than this.

Integrity control makes sure that the knowledge model can be consulted at every moment. This allows testing of the knowledge model to its ability to deal with a specific case. As each parameter can be used a goal, testing parts of the knowledge system, up to an individual decision-table is easy. Additionally, a decision table can therefore be developed incrementally as a separate functional unit. By virtue of this knowledge model representation, the knowledge can start small, and accrete in small increments and be extended to include additional functionality.

### A Single Table

Using the KBE, developing a knowledge model begins relatively easy. The basic system can be as simple as a single decision-table and a couple of parameters. Such small systems can elucidate certain reasoning, and allow people to make a certain decision explicit. For example, the calculations necessary to determine eligibility for rent-subsidy are relatively straight-forward but difficult to explain. Rent-subsidy is a process which many different people with different backgrounds go through, but often only once or at least very few times. As an alternative to a leaflet, a knowledge model consisting of a single table can solve the problem. Iterative or cyclic knowledge modelling makes use of this as a starting point, although in most cases a higher-order result is required. Nevertheless, this effect can also be seen on larger scale, and is still present when examining the development of individual decision-tables in the system.

The system guards the integrity of each table. This eases the development of a table considerably. In every configuration, a decision-table is always in a correct and consultable state. Errors that break the knowledge model are simply impossible, because the system keeps each decision table consistent and complete. For example, a new condition adds new condition alternatives for each of the columns, filled with a `don't care` expression. Whenever a `don't care` alternative is provided with a definition, another alternative is automatically added filled with the `else` expression. Conversely, when all the alternatives are deleted, the final alternative is filled with a `don't care`. This last one cannot be deleted.

Furthermore, when the table uses a parameter name, but the name of the parameter is not present in the list of parameters, the user is asked whether he would like to add it to the list of parameters. These initially default to String parameters.

The first attempt at a table is often not the correct one nor the optimal one. Sometimes this becomes obvious when the expert views or comments on the table, but many times this becomes obvious from building the table in the first place. Conversely, the consultation of the system can reveal inconsistencies in its notation mainly from the external behaviour of the system.

The common mode is that as the system grows, tables are added, and the list of parameters grows. The major activity however, lies in sharpening up the decision-tables. Different types of feedback make it clear that certain aspects are not present, leading to questions to the knowledge provider. This can be an incomplete covering of the domain of a parameter, when only red and white wine is in the table and the domain also contains rose and champagne. Alternatively, when the action alternatives are blank in certain situations, this is an indication that some situations just are not covered. Each of these questions leads to improve upon the table being in a piecemeal fashion. Adding new alternatives, changing existing ones or even removing them can all be done with relative ease.

The development of a decision table can be likened to that of ripple down rules, where there is an increasing refinement of the knowledge present. An exceptional case is quickly added to the table as can be seen in the case of champagne in the example above. At every point in the table can be room for additional detail, and particularities can be inserted. When a new parameter becomes relevant in a particular case, it can be added as a condition, and this exception is handled, leaving the remainder as don't cares.

### 4.3.2 Adding tables

If a single table does not suffice, the next step is to add new tables. Adding a table is a simple button press. In principle, the system allows new tables to be added singularly, based on the inference of a single parameter. The common approach is to add sub-tables, which is basically a top-down approach. Another possible way is to add a decision-table that uses the existing table, in a bottom-up fashion.

#### Sub-tables

When one of the conditions in a decision-table needs further deepening, an additional table can be added for that condition's parameter, as a 'sub-table' of the original table.

	<b>Wine type</b>	R1	R2	R3	R4	R5
C1	<u>Main course</u>	red meat	poultry		fish	ELSE
C2	Turkey is served	-	yes	ELSE	-	-
A1	Wine type	red	red	white	white	ASK

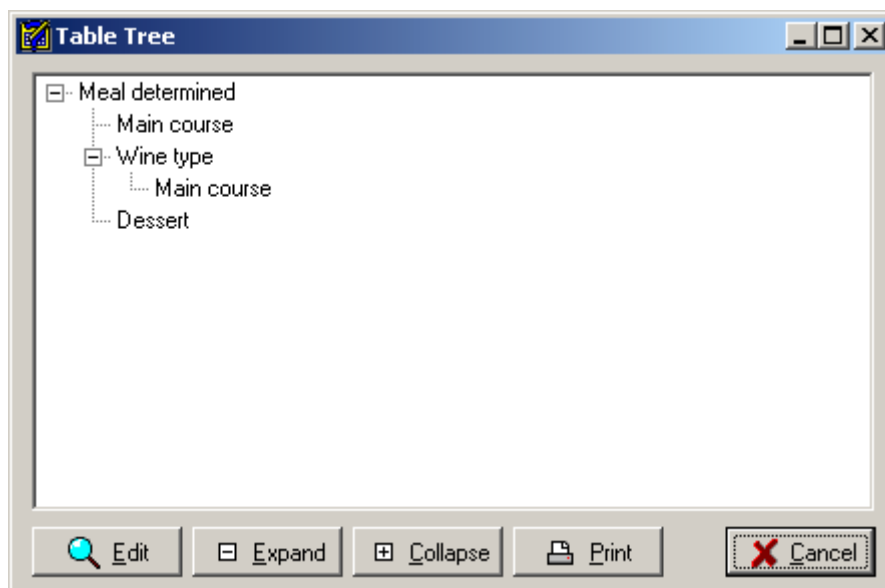
**Figure 4-13 Super Table**

The fact that the *Main course* has a sub-table is indicated with the underline in the top-left condition.

	<b>Main course</b>	R1	R2	R3	R4	R5
C1	Any vegetarians	Yes				ELSE
C2	Country Style	American	English	Japanese	None	-
A1	<u>Main course</u>	red meat	poultry	fish	ASK	something else

**Figure 4-14 Sub Table**

In the figure above, the sub-table elaborates on the *Main course* condition of the previous table. When the value for that condition parameter is required, for example when the super-table is evaluated, the sub-table is evaluated in the process. For conceptual purposes, it can be supportive to think of the interconnections as a tree.



**Figure 4-15 Tree Structure**

In fact, the connection between the tables goes through the parameter in the condition. This said, a table may have more than one super-table, as the parameter may feature in more than one place (see for example the wine type which features twice in the tree). In addition, a parameter can have more than one decision-table that assigns a value to it. In this case, the conflict resolution procedure will choose which decision-table to evaluate (see *Inferring a Parameter* on page 117).

### Structural Knowledge and Tasks

It is also possible, in a reverse form of what was discussed above to introduce new decision tables, ‘above’ another table, subsuming its reasoning. This allows for bottom-up approaches that proceed from the earlier description. Therefore, only a special case will be treated here.

	Meal determined	R1
C1		.
A1	DETERMINE	{Main course}
A2	DETERMINE	{Wine type}
A3	DETERMINE	{Dessert}
A4	Meal determined	Yes

**Figure 4-16 Structure Table**

A common approach is to introduce so-called structure tables, which can organise and steer the reasoning process (cf. the structural rules in Clancey (1983) that contain *strategic* and *meta* knowledge). In larger systems, these structure tables can constitute the majority of the tables contained in the system. Experts sometimes have an explicit view of the order in which certain tasks are executed in the domain. They also feel unsupported by, in their eyes, weak control derived from the inference mechanism.

Using structure tables such as the one in Figure 4-16 is similar to the introduction of a control structure in a program, introducing a linear, and sometimes, conditional ordering of steps. The DETERMINE entry in the table is not a keyword, but a dummy parameter to which the value of the replacement expression to the right is assigned. Different forms of these structural tables exist, depending on the preference of the modeller.

In the eyes of an expert, the operation of the system can sometimes appear random. Not so much because they do not understand the modelled knowledge, or the way this knowledge is employed by the system's inference mechanism. The experts and knowledgeable users attempt to second-guess the system using their own line of reasoning, expecting certain questions to be asked. The system may have left these out because they are not relevant.

### 4.3.3 Change and Enhancements

During the lifetime of a model, its knowledge changes and the model is expanded in functionality. Make a change or an enhancement is very much influenced by the subject matter that has already been added to the system, the bias of the knowledge model. Such a bias always grows into a knowledge model, in implicit and explicit assumptions and particular idiosyncrasies of the modellers that were involved.

Because of these influences understanding the structure and content of a knowledge model is important when making changes or contemplating the best way for introducing enhancements. The task of understanding a knowledge model can be supported by giving support for distinguish the relationships that the element that is to be changed has with other elements. Making changes to a knowledge model can be difficult because of the many implicit assumptions and hidden structure. It is important to understand the relationships between the different parts of the model, to locate elements.

KBE allows the user to locate all uses of a particular parameter and pure text search. It also includes facilities to establish which elements in the knowledge model are not connected to other parts in the model (orphaned). These, together with other means of navigating the system support the ability to understand the model it contains. The user can navigate from a parameter to all the locations in the model where it is used, and from those to the parameters that it influences and so on.

Enhancing a knowledge model often means adding additional sections as distinct new functionality into the system. After developing a model for problem diagnosis naturally comes a model for problem solving, which means that a new structure table is placed on top of the earlier table, to first ascertain the diagnosis and then start the solution to the problem based on that diagnosis. It joins the two sections in the knowledge model.

Another more explicit approach makes use of this feature directly. Developing a structure table, with trivial implementations for parts of the knowledge model yet to come, prepares it for enhancement and helps structure that development. These trivial implementations create 'space' within the model with the idea that the space will be filled at a later point in time. The purpose of this approach is to flesh out and test the structure tables in the system, which can then become relatively stable when compared to the remainder of the system.



### 4.3.4 Problems and Limitations

The representation of knowledge chosen in the KBE aim to be simple and understandable. The simplicity of the chosen representation has number of problems associated with it. One of the main problems is the lack of scalability of the knowledge base. The system knows no modular structure. Without measures being taken, the size of the knowledge base can quickly become overwhelming. As a decision table's meaning within the knowledge base is determined largely by its use, knowledge of the whole system is required to understand the implications of a change. Running test cases can then become the only means to ascertain the working of the knowledge system.

As was shown when the number of decision-tables grows, decision tables can be added to impose a structure on the tables in a top down fashion. In fact a whole section of knowledge can be captured in this way, that could be denoted task-knowledge. The decision table format is not the most appropriate for this, leading to knowledge being 'hidden' in the decision tables. Only that which can be caught in a decision-table is seen by the system. Implicit assumption consistently used remain however, and are sometimes explicitly used by the expert and developer.

The structure-less parameter representation is also not able to deal with problems that concern instances of a common type, such as reasoning over a set of rooms. This a fundamental flaw in the representation format. To be able to reason about more than one room, each room must be modelled explicitly, in its own set of parameters. This is also called an 'is-a' or instance problem.

## 4.4 Knowledge System Development

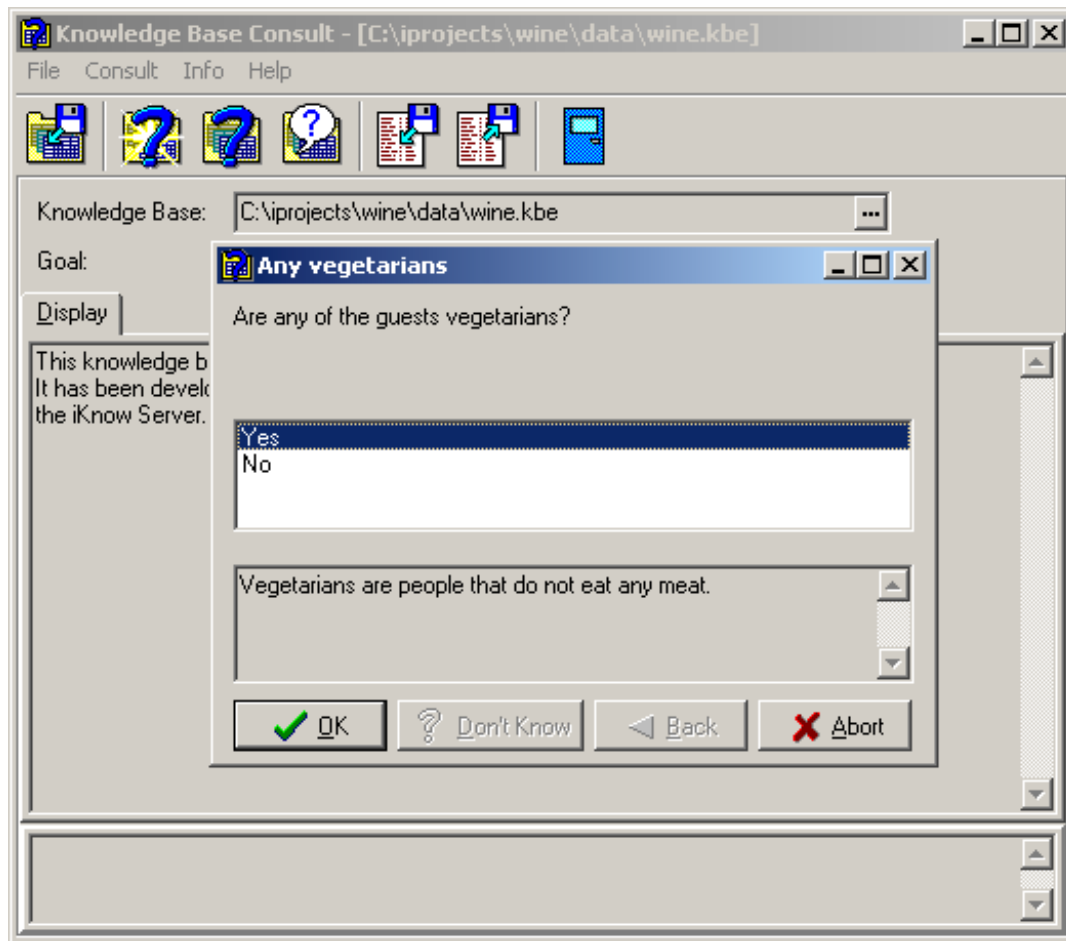
There are several ways that a knowledge model can be deployed. There is a need to provide for off-the-shelf default solutions, but systems must also be fitted with dedicated visualisations to support the user. Furthermore, incremental development of such visualisations must be supported. This section discusses all the possibilities that exist to deploy knowledge systems using the KBE.

### 4.4.1 Default Deployment

The intent of these systems is to provide a suitable environment for the knowledge bases to be used with zero effort, within the capacities of a non-specialist users of the system. Based on the knowledge base component a number of default- applications is available created. The architecture of the KBE allows consultations to run as standalone or as web-application.

#### Knowledge Base Consult

The KBCS is the standalone version of a default consultation tool. It consults knowledge bases designed with the Knowledge Base Editor. It is not a very advanced tool and it provides only rudimentary consultation facilities. In fact, these facilities are the same as the functions present in the KBE without the editing facilities.



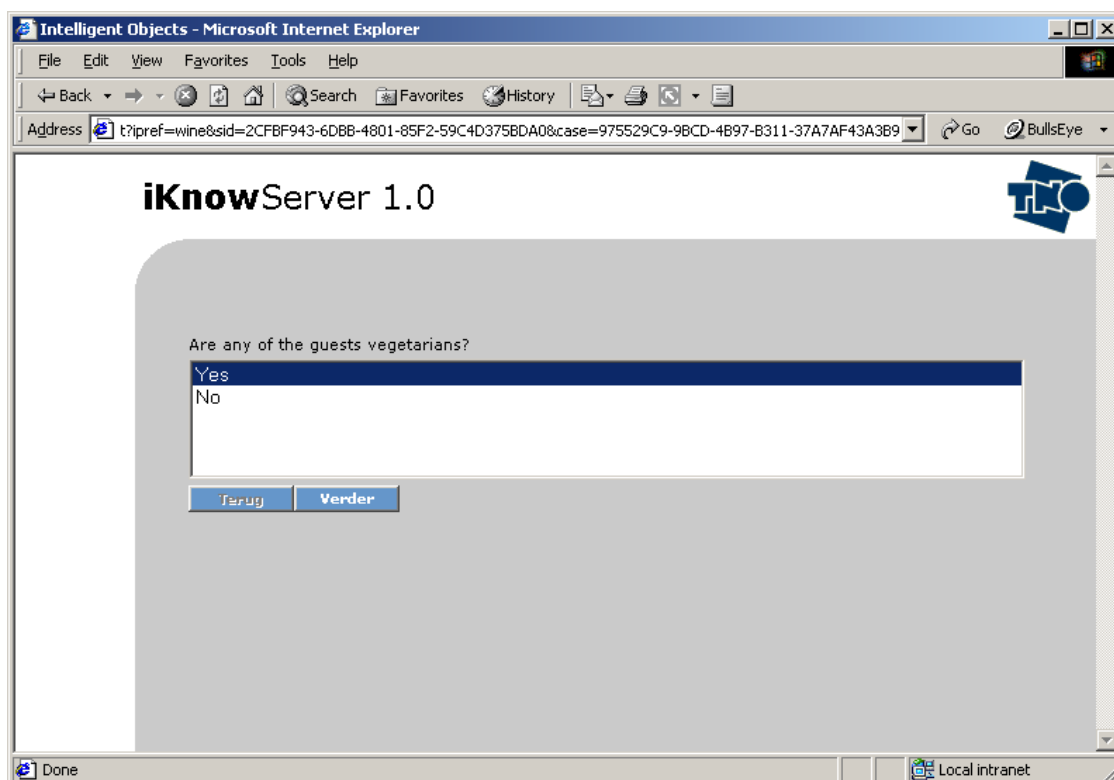
**Figure 4-17 Knowledge Base Consult System**

As shown in Figure 4-17, the KBCS system can be used as a knowledge base viewer. A dedicated format has been created that allows knowledge bases to be shared. The files are encrypted which makes them impossible to inspect or open for inspection in the KBE. The knowledge bases can therefore be shared and deployed without fear that critical knowledge is copied or changed. The latter is also important as the knowledge may have strict requirements on the validity of the knowledge that may not be compromised.

### **Default iKnow Visualisation**

The iKnow architecture offers the possibility to deploy a knowledge system, through either as a stand-alone system or through an Internet server extension, using the same visualisation of HTML templates. A series of default templates present in the system are used when no preferences are provided by the author of the system. These preferences are normally located in the preference file for the knowledge model (iPref file).

When the server is up and running the deployment of the knowledge system is relatively easy. The installation of this infrastructure constitutes no more than the installation of a web-server extension (a standard solution for these kinds of tools). For each knowledge system or update of a knowledge system, uploading the knowledge base files to the web-server in its own dedicated directory as well a simple preference file is all that is required.



**Figure 4-18 Default Web Interface**

To start the consultation of the system a normal URL can be used, entered into the browser of choice or through a link on an HTML page. A stand-alone version of the iKnow system uses the same default templates and does not require a preference file, but allows distribution in the same fashion as the KBCS stand-alone application.

#### 4.4.2 Customisation

The important difference in using the iKnow facility is its customisability. Both the Internet and stand-alone iKnow system share the same set of templates. Therefore, any changes made for one version are quickly integrated into the other system as well. Furthermore, a change in preference file and templates can change the complete look and feel of a knowledge system, allowing for differential visualisation. The need for customisation is to make a system conform to the expectations of a user, and make it easier to use the system. The approaches discussed here show an incremental strategy that can be used by virtue of the iKnow architecture.

##### Preference File

The iKnow knowledge server is accessed using a URL of the form "http://<sitename.com>/iknow.dll?<name-of-system>". The name of the system points to a specific preference file name <name-of-system>.xml. A preference file contains a number of fixed partitions dealing with different parts of a knowledge system. Some concern simple messages that are presented to a user, while others determine the templates to be used and various other settings.

Title	Description
kbase	Determines which knowledge model this preference file is for. There may be more than one preference file describing a knowledge system configuration per Internet server.
templates	Lists all the different fixed and custom templates that are defined for the knowledge system.
goals	Contains a list of all the goals that the knowledge model can be consulted for. One of these goals can be tagged as the default goal.
messages	Holds the text for certain fixed messages, such as errors, warnings, etc.
reports	Lists the different reports that can be generated for the knowledge system. More on this later.
casebase	Describes in which folder to store the consultation cases that are kept for each user.
database	The different cases produced by the different users have to be coordinated. This is accomplished using a database in some cases.
settings	Contains various other settings. An example is the user authentication levels that determine whether the user must be known and has to log in or anybody can consult the system anonymously.

**Table 4-5 Preference file elements**

Using the preference file, a parameter can be given a specific dedicated template. This means that the parameter when asked uses that template to build the question and send the answer. This allows for example that images are used to present the choices or that an image-map allows people to click on that part of a map that they wish to explore.

```
<template id="error" type="system:error"
  file="error.htm"/>

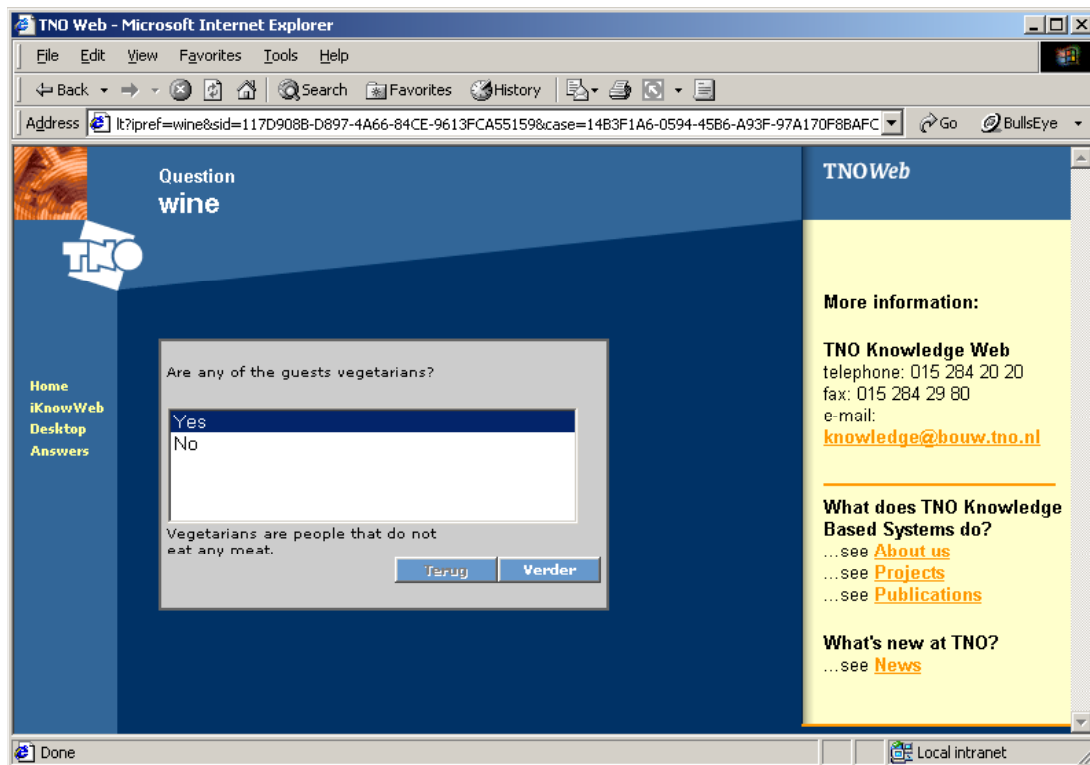
<template id="graph" file="graph.htm">
  <parameter name="Characteristic graph"/>
</template>
```

**Figure 4-19 Template examples**

The figure shows two examples of a template definition. The first show the definition for a fixed template, the one that will be used for any of the system's error messages. The type `system:error` identifies its role, and the file identifies the name of the file containing the template. The second template definition is slightly different, it contains no type and within it, a single parameter is identified. For that specific parameter in the model, the `graph.htm` template file will be used. The parameter tags are contained within the template tag, because more than one parameter may be serviced by the visualisation contained in the template.

### Custom Defaults

The customisation starts with replacement of the default HTML templates by other system templates made for the system in question. This changes little to the functionality of system. It merely changes its presentation, an HTML page to perhaps the style of the company or a colour scheme that fits the company colours.



**Figure 4-20 Customised Dialog**

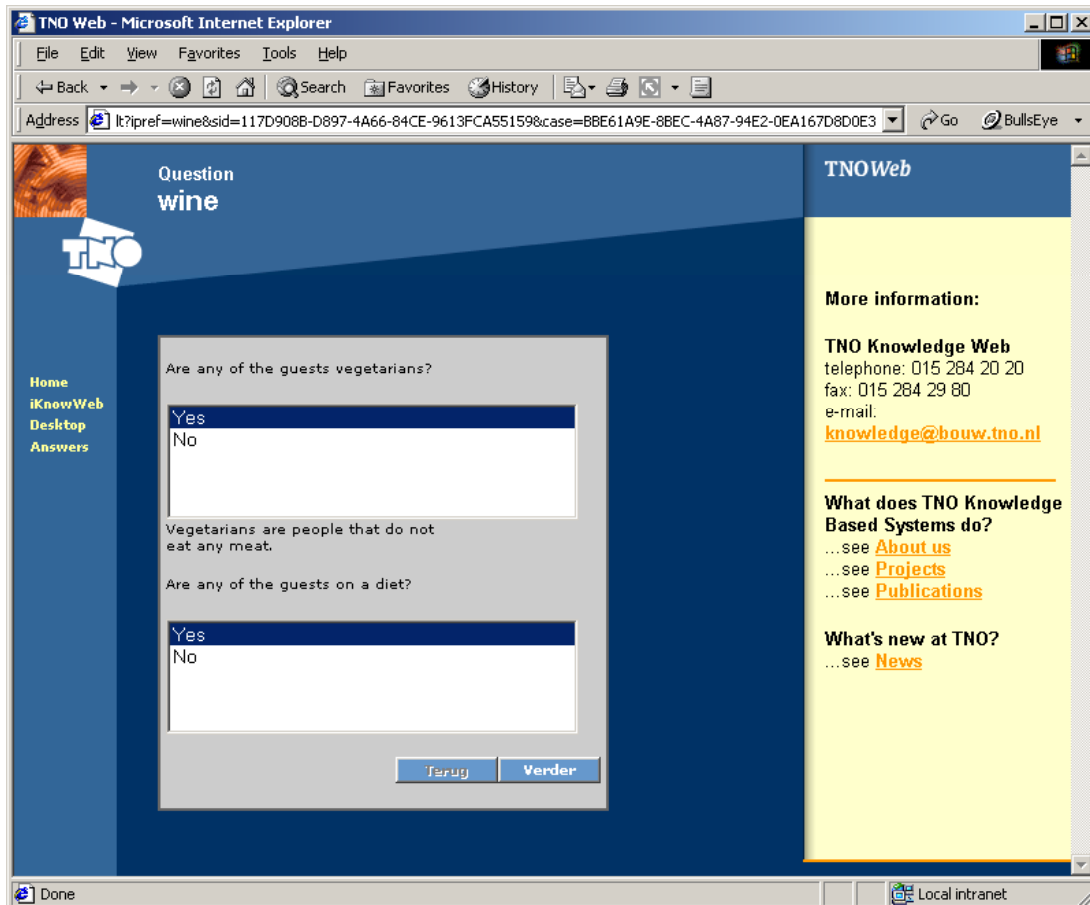
An example is given above, showing the knowledge model as displayed in the TNO web site look and feel.

### Custom Parameter Dialogs

Further customisation would entail producing different templates for specific parameters within the model. This would allow certain aspects of the usage of the system to be drastically changed. The capabilities of HTML offer a variety of visualisations to be used, like image maps, Flash animations and many others, while not requiring highly trained people to develop these visualisations.

The dialog can also present more than one question on a HTML page as is shown in Figure 4-21. This allows some related questions to be asked at the same time, rather than one after the other. For the user this makes an easier user interface as the questions can be grouped into meaningful configurations. For example, name, address, and town can be asked together.

A problem may occur when questions are dependent upon each other. For example when asking whether the room is a bathroom or a bedroom, the subsequent question how many people sleep in the bedroom is not relevant for the bedroom. This question should be disabled when the bathroom is chosen. Such interdependencies can be handled in a knowledge system by resending the HTML page to the user, whenever a change is made, but a more practical solution in many cases is to employ Java-script.



**Figure 4-21 Multi Question Dialog**

This does mean that some knowledge is then present in the visualisation of the questions, which may lead to maintenance problems when this is a common approach. Changes that occur in the knowledge itself must be reflected in the user interface. Therefore the latter approach is only used as a last resort.

### 4.4.3 Dedicated Solutions

Dedicated solutions are a step beyond customised solutions. This comprises a knowledge system that is given a specialised user-interface completely oriented to make the use of the system as easy as possible, or a knowledge system as an integrated non-visual component within a larger system.

#### Dedicated User Interface

Knowledge systems are often positioned as standalone systems, that provide for certain functionality as a whole. A dedicated solution can be developed in any phase of development of a knowledge system. This goes beyond what is mentioned earlier for customising a knowledge system.

By accessing its API, the connection is easily made between the functions of an application and the knowledge base. It is possible to build a customised user-interface around the application in advanced professional programming environments. In this way, a knowledge system can be developed much as any other software development, but with a knowledge model as its core.

The end result of such a development is software that is indistinguishable from other professional, industrial strength software. In fact, such software will most likely not be perceived as a knowledge system, as knowledge systems have an aura of difficult and complex surrounding them.

### Integrated Solutions

A knowledge system can also be a component of a larger system, much like a database. It is also possible to use the knowledge base as a part of a greater system, as a calculation core, answering all the questions that are posed itself, without ever posing one to the actual user. The component-based architecture makes it easy to use the Knowledge Base Component in any software. Communication between the application and the component is realised through its API. To an outside observer the presence of a knowledge system within the larger system may not be suspected. Its presence may not appear to the users of such a system, but this can allow for some additional flexibility in a system.

#### 4.4.4 Developing Reporting facilities

The KBE knows an additional service that allows the creation of templates for document generation, as post-production after the consultation. This can be used in the iKnow architecture, but also features in dedicated and integrated solutions.

`case`           Conditionally places text in the report, when the case matches the parameter.

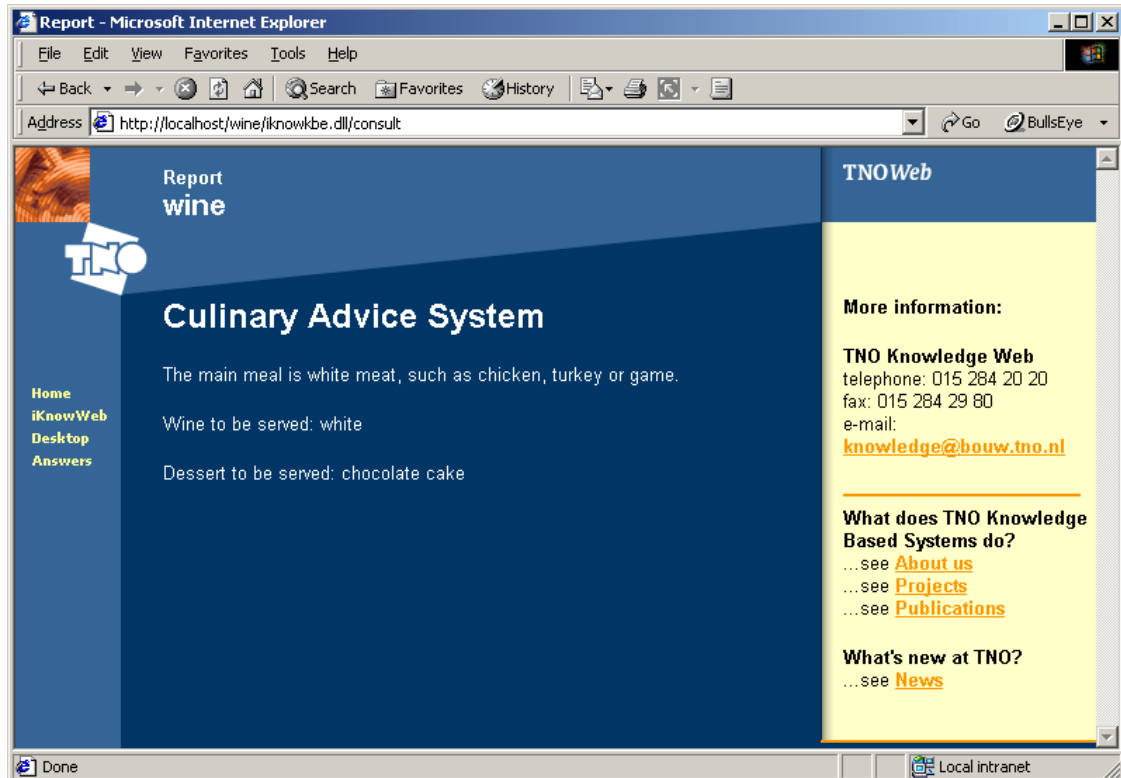
`parameter`   Includes the value of a parameter in the report.

These two simple commands can be used are shown. These determine what specific text to include in the report. Because it is possible to include the value of a parameter, it is possible to perform some of the more complex reporting tasks from within the knowledge model. While this may be seen to dilute the distinction between model and view, some facilities exist within the KBE to keep those parts within a knowledge model that concern reporting separate.

```
<report>
  ...
  <case name="Main course" value="poultry"
    operator="equal">
    The main meal is white meat, such as
    chicken,
    turkey or game.
  </case>
  ...
  Wine to be served: <parameter type="kbe"
    name="Wine type"/>
</report>
```

Figure 4-22 Report Template

The XML format of the report is processed to create a final report. This requires that the values of parameters are placed in the text and that the text from the case is only included when the condition is true.



**Figure 4-23 Example Report**

As can be seen from the example report, it can be quite specific to the situation. The danger here lies in that part of the knowledge is transferred to the report creator, whereas the knowledge should be in the knowledge base. This would create many difficulties. To keep the XML report-file as simple as possible, and remove the knowledge content of the report format, it was kept intentionally simple.

#### 4.4.5 Conclusions

The facilities that were described allow for high degrees of customisation to be added incrementally. The iKnow architecture provides for both distributed and centralised access to knowledge systems, which allows for most distribution models that currently exist. This manner of development fits well with the necessities of continuous knowledge engineering. In addition, by allowing such a gradual path with the possibility of changing to a dedicated solution, any amount of effort can be placed in the development of a specific system, at any time during the development.

### 4.5 Conclusions

The description of the KBE in this chapter shows a simple knowledge system development environment. To this simplicity it adds a visual form of knowledge representation and the ability to support a full development process towards professional industrial strength knowledge systems. The KBE realises almost all of the requirements for tools to support continuous knowledge engineering, as formulated in the previous chapter.

The different features and architecture of the KBE described in this chapter each support one or more of the requirements. The table below describes a summary of these requirements as well as the aspects of the KBE that support them.



	Visual Knowledge Representation	Component Based Development	Model View Controller	Abstract Communication	Centralised Deployment	Executable Specification	Live Model	Simple Knowledge Model	Simple Inference Mechanism	Adaptable Visualisation
<b>Participation</b>										
Understandability	✓			✓		✓	✓	✓	✓	
Locatability	✓			✓			✓	✓		
Expressibility	✓						✓	✓		
Changeability									✓	
Extendability									✓	
<b>Stewardship</b>										
Usage Information			✓	✓	✓					
Metrics				✓	✓	✓	✓			
<b>Medium</b>										
Operational		✓				✓	✓			✓
Gradual UI		✓	✓	✓						✓
Diff. Visualisation			✓	✓						✓
Dedicated Solution			✓	✓						✓
Integration		✓	✓	✓						✓
<b>Cyclic</b>										
Start Minimal					✓			✓	✓	✓
Small Increments					✓			✓		✓
Revocable Changes					✓					
Scalability		✓							✓	✓
Gradual Integration		✓								✓
Easy to Change								✓		
Easy to Extend								✓		

**Table 4-6 Features and Requirements Implemented by KBE**

1. As the system is always operational, the smallest change is also the smallest possible increment.
2. Internet-based consultation is scalable with ease, as increasing numbers of users come at no extra cost.

The table shows that the only requirement not supported is revocable changes, from within the system. However, the files can be placed in a version-control system, to make backups of different versions of a model. Therefore, this is a requirement that is not essential in this approach.

The features that implement the requirements have overall created a system that is easy to understand and use. A knowledge model can be comprehended to its dynamics with little knowledge of information technology or software. The KBE is a simple tool that can be used by non-specialists and has sufficient features and capabilities to develop small and medium sized systems. The decision-table format is a useful knowledge representation because it inherently supports direct understanding, modifiability, verification and validation. The expert in question can perform this validation and verification, and non-specialists and novices could arguably be seen to perform those tasks as well by virtue of its simplicity and visual knowledge

representation. The KBE's ability to consult the knowledge base at any moment in time supports verification and validation.

In addition, deployment can be reasonably simple using the technologies described earlier and facilities for building dedicated systems are integral to the system's architecture. There are several ways that a knowledge model can be deployed. Standardised solutions include a stand-alone system, and a web server extension that can dish out HTML pages. Both systems operate on a series of default templates, but it is possible to define a custom page for a parameter and to create elaborate pages that answer several queries at once. Dedicated systems can also be made using knowledge-aware components that are used to create sophisticated forms.

The KBE allows for different deployment scenarios by virtue of its many formats for publications, where many different types of visualisation are possible. This includes centralised deployment as well as stand-alone standard modes. The knowledge base engine by its component-based architecture can be easily incorporated into larger systems. This allows it to be employed in many different settings and in different roles. In these ways, the system supports continuous knowledge engineering.

This simple representation of knowledge in the KBE in a unstructured domain model, and singular representation of reasoning knowledge as decision tables also creates a number of problems.

The necessity or wish to enforce control over the reasoning can lead to many structure tables being used, which is can be seen as a misapplication of the decision-table representation form and may lead to maintenance problems. But it also brings to the fore an important need to structure the domain, in terms of the control structure that is apparently part of the knowledge as the solution strategy.

Fundamental representation problems also arise through the simplicity of the system. Most notably this includes the inability to represent problems about variable numbers of instances and collections. The KBE cannot, or at least with considerable difficulty, represent problems dealing with different instances of an element. It can reason over a room, but it has no facilities to reason over a set of rooms.

The knowledge model format further has no facilities to provide structure in the domain, to pool different related parameters together. This make it harder to manage and maintain the factual knowledge represented by the parameters, when the number of parameters grows. Approaches commonly found in knowledge model devised using the KBE are to introduce textual hints in the names of parameters, clustering them in that way. For instance, all parameters dealing with a type room are prefixed with the roomtype 'bedroom temperature', 'bedroom occupied'.

The system has features worth keeping in its simplicity but causes problems by its representational poverty and its lack of decompositional features. The problem is that the simplicity is also a result of the absence of such advanced features. Solving these problems without sacrificing the simplicity may prove to be quite difficult.

# Chapter 5

## An Advanced Tool

---

*Things should be made as simple as possible, but not any simpler.*

*Albert Einstein*

This chapter describes an advanced system for knowledge modelling and knowledge system development called Intelligent Objects (IO). This system is based on a philosophy of vividness, in other words creating ‘true-to-life’ models of the world. IO incorporates features that are also used in the KBE, but employs them in a different way. In addition, IO introduces new capabilities. To avoid reiterating the descriptions in earlier chapters, this chapter will therefore focus on these differences and additions.

In the first section of this chapter the general overview of IO provides an examination of the philosophy behind the system, the architecture, and key features. The second section presents the set of modelling primitives that IO offers. The modelling vocabulary of IO based on object-oriented models, providing various representations of reasoning knowledge and an explicit reflective architecture. Continuing in the third section, an assessment is made of the manner in which IO can be used to model knowledge. This examines the basic modelling approach and aspects of that approach that enable the system to be used in evolutionary development strategies. This also goes into some practical problems that arise as the result of the more involved approach taken in this system. The fourth section discusses what the differences are between the possibilities offered by IO and KBE in developing a knowledge system. As both use the same iKnow architecture, in this respect they offer largely the same potential to visualise and deploy a knowledge system the KBE has. The fifth and final section recapitulates the subjects that were presented, and highlights the primary characteristics that will come under consideration when the system is applied and evaluated.

### 5.1 Completeness versus Simplicity

The simple tool described in the previous chapter provides for support to create and develop systems based on the continuous knowledge engineering principles and requirements. The limitations of its simplicity create problems of its own, in terms of scalability and a number of fundamental representation problems. The tool discussed in this chapter pursues a different philosophy. It attempts to aid the user by providing a richer form of expression that is based on object-oriented principles. Retaining some of the accomplishments of the KBE in many other respects, it aims to create a kind of simplicity in the models through the principle of vividness. In some ways it is a different software implementation of the features found KBE, but the difference in philosophy makes the modifications and extensions all the more important.

#### 5.1.1 Philosophy

The philosophy maintained by IO is to offer freedom of expression and multitude of possible forms of knowledge representation to create a vivid model of knowledge.

This model should be the least constricting possible to enable the development of models that conform to an expert's own opinion on the domain of expertise.

The ease of use aimed for derives from creating models that are familiar to the expert, much like a map of one's own neighbourhood. Because the expert makes the map, this combines to empower them to know where to go, how to make a meaningful change and verify that the change indeed has its intended result. A concept that exists in their mind will be recognized as omitted in the knowledge model. Locating a place to make a change, and the character as well as the form of the change is informed by the bias of what was modelled already. This approach requires extensive and varied representation capabilities, while retaining simplicity of use of the representation forms. The aim is to make the knowledge model as simple as possible, rather than reducing the initial complexity of the tool used to build it. This philosophy may therefore seem diametrically opposed to the one followed by the KBE.

The idea here is to supply a complete set of modelling concepts that create simplicity by offering many options to model a domain. Without wishing to undermine the quest for simplicity and ease of use, IO sees the need to represent a great many things with a different character and purpose. These can be described and modelled in a variety of ways. The IO system offers a richer and more complete set of expressive capabilities for this purpose. This must allow the expert to build knowledge models that have a direct correspondence with the domain, making them easier to understand, modify and verify. IO discerns three basic categories of components in a knowledge model: structural, behavioural and reasoning knowledge. The KBE only knows one kind of reasoning knowledge.

A domain knows many entities and these entities are interrelated somehow, where one of these relationships is often one where an attribute described some part of the state of that entity. Such entities are defined by their state, their behaviour and the relationships implicitly and explicitly available. This is the structural knowledge, analogous to structural knowledge representations.

The entities in such a domain are therefore more than static descriptions, but active representations of elements in the domain that can exert some dynamic behaviour. These can be actions to change the state of an entity, but also can be ways to model certain task behaviour. The description of this behaviour must allow expression of actions and task structures that can be performed on and with the entities in the domain. This can be behaviour supported by elements in the domain, but also approaches to problem solving, encapsulated as behaviour supported by the domain as a whole (cf. PSMs).

The last category of knowledge is reasoning knowledge, to make inferences in the domain. Where decision-tables are the sole type of reasoning in the KBE it is easy to see that this can capture only part of all the modes of reasoning that may be required and in many cases will not be the preferred mode of expression. Many different ways of modelling knowledge are possible each with its own strength and weaknesses, as was seen in the second chapter.

The differences between these forms of representation must be understood, as when modifications are made, choices must be made between different change strategies. The large number of concepts needed to be understood by the user makes for an initial complexity that is greater as there are more concepts that have to be understood.

There is a non-trivial relationship between the learning curve introduced by these functions and representations, and the simplicity of operation of a system. The conclusion that the KBE's tools simplicity was reached in the previous chapter that the simplicity of the KBE and its ease of use could not be equated directly to the ability of a non-specialist to operate the KBE, even in participation with the knowledge engineer. At a certain size, the knowledge base becomes too complex and difficult to understand. The single representation of the decision table is used for purposes where other approaches could be more appropriate. The base simplicity gives a false view of the actual complexity, where IO aims to attain a scalable simplicity. In short, the path IO travels is characterised by attempting to speak the language of the expert, by allowing a knowledge model to remain close to the expert's perception of the domain. In the same way the implicit simplicity of the KBE may turn out to be more complex in the end, the initial complexity of IO may fade away in time.

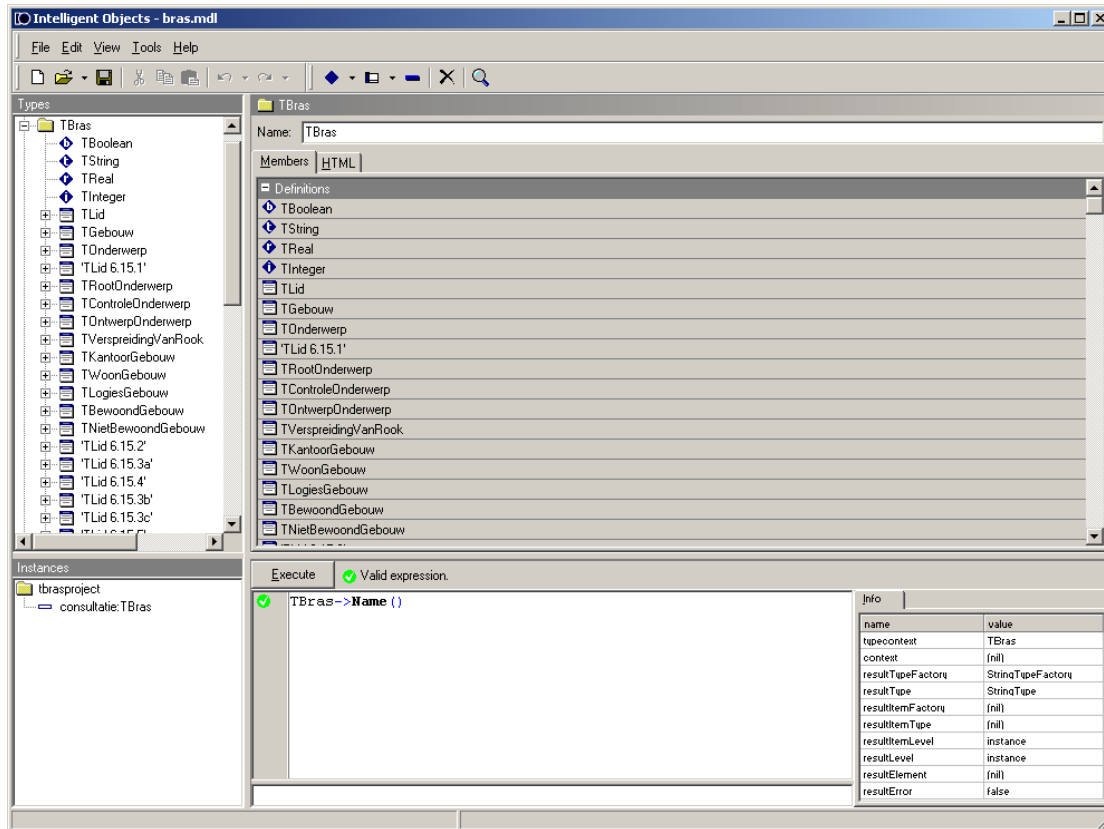
### 5.1.2 System

IO is a knowledge modelling tool based on the philosophy of vividness. It offers extensive facilities to model knowledge in a variety of ways. The system allows for development of second-generation knowledge systems. While it maintains a simple backward chaining inference mechanism, its operation can be directly influenced by explicit task structures. As an advanced system for the development and consultation of knowledge systems, IO is a continuation of efforts present in the KBE. For example, an IO knowledge model integrates seamlessly into the iKnow architecture, in the same role as the KBE knowledge base component.

IO supports the capture, maintenance and consultation of knowledge. Just like the KBE, it provides a point and click environment that is user-friendly, intuitive and easy to use. It is directed towards non-computer specialist users although its use is arguably more involved than the KBE. The main difference lies in the domain modelling capabilities, based on object oriented models, different inference knowledge representations, an extended language for articulating expressions and statements, and the use of a live model, providing feedback on any change made to the system.

IO's philosophy is to supply a vivid model, which is divided into structural, behavioural, and reasoning knowledge. The structural model is an object-oriented model. It contains descriptions of classes, associations, attributes, collections and other elements. The knowledge model, or project in the nomenclature of IO, is divided into a number of knowledge bases. Each knowledge base can contain a series of classes, and associations. IO, as shown above, places the importance with the domain model, which is used to organise the reasoning knowledge. This is counter to the reasoning focus approach of the KBE, where a view was forwarded of parameters in service of the decision-table. Using object-oriented model constructs allows them to function as a map of the territory.

In Figure 5-1 a screen shot of the IO knowledge modelling environment is shown. It displays an example knowledge model. The `TBrasProject` shown in the top-left tree-view contains a single knowledge base, called `TBras`. It in turn contains different types, both primitive (`TBoolean` a Boolean sub-type) and structured (`TLid`, which is a Class). The bottom-left tree-view shows an instance model. An instance model is an instantiation of the project.

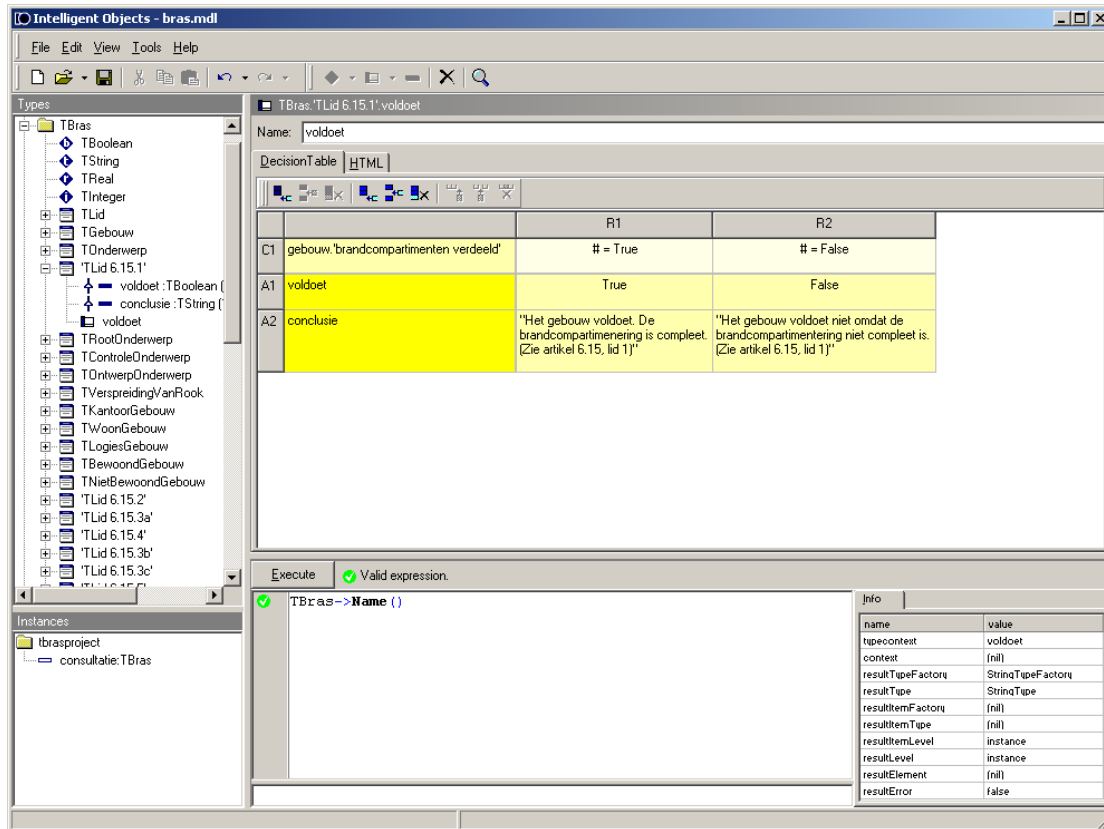


**Figure 5-1 IO Knowledge Modelling User Interface**

The bottom-middle corner contains an IO language editor, with additional detailed type information to its right. Entering commands there allows the user to operate on the model through its reflective language definition. The language editor is syntax highlighted and shows whether the expression or statement entered there is syntactically correct and whether it is valid in the model. It is used to answer queries on the model through expressions in the language. The figure above shows a query for the name of the type of an attribute. It can also be used to make changes in the knowledge model. For example, the command `TBras->Define("TDog", Class)` defines a class type named `TDog`. This part of the user-interface is provided as an advanced user feature. Most of these language operations are also accessible through the user-interface itself.

Using this interface, modelling starts by creating a model of the domain and other important concepts that may be necessary. This involves defining types, either primitive or complex, such as knowledge bases or classes. Relationships can be defined between classes, going beyond what is most often available in executable object oriented specifications. Classes, and other structured elements, can be provided with definitions of attributes and operations. The later defines an elements dynamic behaviour.

This structural model can be augmented with reasoning knowledge by adding reasoning method to an attribute. These count as determination methods when a value for a field is required, for an instance of a structured type. A reasoning method is added to the list of reasoning methods as soon as it contains a possible assignment to that field. The presence of `voidoet` together with a value `True` in the action alternative forms an assignment statement, which informs the model that the decision table should be counted as an inference method for `TLid.voidoet`. When the last assignment to that attribute is removed then the method is automatically removed from that list. This requires further no involvement from the user. An example of this is shown in Figure 5-2.



**Figure 5-2 Automated Reasoning Attachment**

All the changes that can be made by the user can be undone and subsequently redone. The system supports unlimited undo/redo, to give the user added security and allow them to operate with impunity. These operations allow exploration of functionality and make it possible to try different alternative solutions.

- Vivid Knowledge Representation
- Visual Knowledge Representation
- Component Based Development
- Model View Controller
- Abstract Communication
- Centralised Deployment
- Executable Specification
- Live Model
- Full Object Oriented Model
- Script Interface
- Unlimited Undo/Redo
- Code Insight
- Adaptable Visualisation

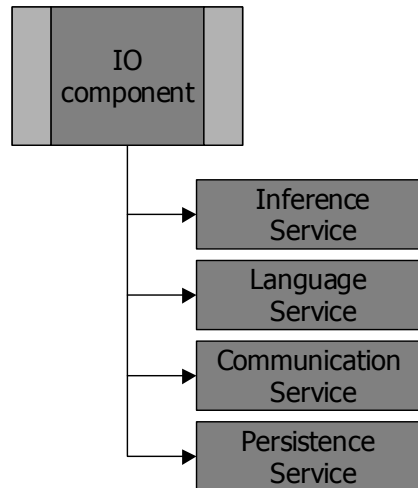
**Figure 5-3 Key Features of IO**

To support the requirements formulated for continuous knowledge engineering IO implements many different features and techniques.

Under the overarching philosophy to support a vivid model, the features shown in Figure 5-3 are considered to be most important in the support of the requirements and the demands of the philosophy. Some of these have already been described, while other will be described in later parts of this chapter.

### 5.1.3 Architecture

The architecture chosen for IO is designed to be extendible in the different knowledge representations that are supported. This extends to the structural, behavioural and inference knowledge. To this end, the design for the system has an extensive meta-model, to which new elements can be added in time. This will be discussed below in paragraph 5.2.2 Meta Model.



**Figure 5-4 IO Architecture**

Beyond the component-based architecture discussed in chapters two and three, it supports a number of services that aim to leverage the potential of the representations for use in design and consultation environments.

The architecture seen in IO shares many surface similarities with the KBE. A component encapsulates a domain model, a reasoning model and an inference element. The difference that is most markedly here is the provision of services. A service allows a service supplier to be created that enables certain functionality. Because the component and the services communicate through an abstract API, replacing the service with another is relatively simple. These include support for general capabilities such as undo-able and redo-able operations on the model and an abstracted communications model allowing for high degrees of configurability in the visualisation of knowledge systems.

#### IO Component

The component and its API play the same role as the KBC from the previous chapter (see page 108). It holds a knowledge model and enables communication with the model, sometimes leading to inferences being made, or other types of dynamic behaviour being supported. The component realises some of this functionality through reliance on one of the services.

#### Services

Rather than a completely different realisation, this architecture reorganises certain functional units within the system into self-contained services. Each service has an associated API, enabling its operation within the system. This allows some separation of concerns and is convenient for maintenance purposes. In many cases, the separation was for expressly for these purposes.

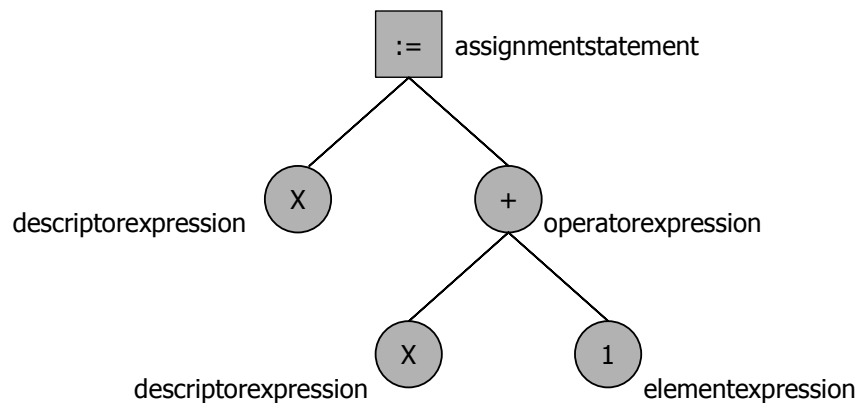


*Inference Service*

The inference engine is a service that arranges for the determination of a specific field's value based on the available inference methods in the model. It can be replaced to allow for implementations that are more efficient. This may be required to allow other kinds of inference, for example forward chaining or utility based inference mechanisms. Now, the inference engine contains a version that attempts to deliver an inference mechanism as close as possible to that present in the KBE.

*Language Service*

The language service is responsible for translation and execution of language expressions. IO currently uses an extension of the OCL language, which enables the formulation of expressions and statements performing queries and modifications on an IO model. As part of the component, this service allows a textual form to be transformed into reified elements. Any element that contains a language expression can use the language service to translate a textual form into reified language structures.



**Figure 5-5 An Assignment Statement Reified**

For example the expression  $x := x + 1$  is translated to the tree in the above Figure 5-5. On a change in the textual form, the language service updates the structure. The textual form of any language expression can be retrieved from the expression elements. This does change the text into a normal form. When a name of an attribute is changed, the text is adjusted to accommodate the new name automatically.

This approach also makes it possible to replace the current language service based on OCL syntax and semantics by another one, perhaps semi-formal, natural language or even a true natural language interface. As long as the element is able to translate a textual form into formal constructs, and back like the tree shown above.

The architecture of the system further allows for a high degree of reflectivity. In fact, the design interface uses the same API as the one available from within the model itself. In short, this means that an operation in the model can change the model itself, reflectively. This allows some additional features to be supported more easily in the future, like wizards for knowledge modelling support and self-modifying, learning models.

*Communication Service*

This service allows the model to communicate with the outside and vice versa. This service is based on a number of abstract communication primitives, both from and to the model. As with the persistence service communication it is supported by functions that allow these primitives to be translated into an XML format.

A replacement or extension of this first abstract communication service could provide for interfacing with other agents through a KQML/KIF type interface (Finin, McKay & Fritson 1992). This has not been researched as yet, but the possibility to allow this has been part of the requirements of IO.

### *Persistence Service*

The persistence service is responsible for reading and writing knowledge models form and to persistent storage. The current persistence service uses a proprietary XML format. An alternative format of any kind may be used, as long as it conforms to the interface.

### **MVC**

Each of the elements in IO is a descendant of a class `ModelElement`, where this class implements the Observable pattern. This interface is part of the Observer pattern (Gamma et al. 1994). It allows other objects to register themselves as Observers, in a one-to-many relationship, so that when the Observable changes state, it notifies all the Observers.

This allows a great freedom in developing view on a knowledge model, where besides editing actions, the model has great autonomy in making changes to itself. These changes need to be reflected in many different types of interfaces and views. All the Views can now simply react to a change by 'repainting' themselves.

This, among other things, allows elements from the model, to be presented in different perspectives on that model. It can just as easily feature in a tree based form, giving a quickly navigable form of the model, as a reference in a decision-table. Each is informed accordingly, when the user changes the name of the element, also negating the need to search-and-replace names in the model.

## **5.2 Modelling Primitives**

In this section, the knowledge representations employed in the IO are highlighted. The first part of the section will concern itself with the necessity to provide a fuller set of object oriented concepts in knowledge modelling, to allow a knowledge model to ascend above the normal implementation details. The second part takes a look at the meta-model that in underlies the total set of modelling primitives. After that, several specific groups of modelling primitives are examined. Finally, a look is taken at the current realisation of some of the services, specifically the inference, language and communication services.

### **5.2.1 Vivid Knowledge Model**

The need to provide knowledge-modelling representations that allow a vivid and direct model has been discussed in the section above. The approach taken to realise this has focused on providing an object oriented modelling language. Particularly, the extent of the object-oriented domain-modelling concepts and the role of an object-oriented model as the scaffold for other knowledge representations are seen as important aspects in this regard.

Object orientation is an important paradigm in the design and implementation of complex software based systems. Object models have proven themselves as a medium for describing the complex relationships commonly found in software and for making

such complexity intelligible. In this sense, even Brooks (1987) has credited object orientation with resolving some of the accidental problems in software engineering. This ability extends to enabling those not native to computer science to understand these models. The visual, conceptual language allows them to understand the models, discuss the content of those models and propose adaptations.

The elements offered allow most forms concepts and their relationships to be defined. The abstract level of modelling brings important concepts and relationships to the front, while at the same time relegating some aspects to the background. This gives a proper framework to discuss the responsibilities of different parts of such a system, from a particular perspective. The visual nature of most object-oriented models strengthens this level of abstraction by removing all kinds of cognitive noise. All superfluous information is removed, focusing on the important aspects.

UML is the foremost standard in object-oriented modelling (Rumbaugh, Jacobson & Booch 1998, Jacobson, Booch & Rumbaugh 1998). As the integrated form of different earlier object-oriented modelling languages, it provides an important standardisation to the object-oriented arena. This standardised nature of UML is also important when knowledge models are integrated into other software systems, as designers of these different systems need to communicate.

Both software engineering and knowledge engineering have now accepted object-oriented models into the mainstream. This is also mirrored by the increased acceptance of UML in the world of AI and knowledge engineering. As an example of the latter, UML has been adopted into CommonKADS (Breuker et al. 1999).

For the development of systems based on object-oriented models, many programming languages also follow the object-oriented paradigm. As the representation follows the principles of encapsulation, modularisation and information hiding, it has many attributes that allow for improved modifiability and enhancement of such implementations. However, comparing the range of concepts offered by object oriented modelling languages to those offered by programming languages, the latter seem to lack support for several modelling elements. Most notably this includes the different relationships such as associations and aggregations that connect the different classes to each other.

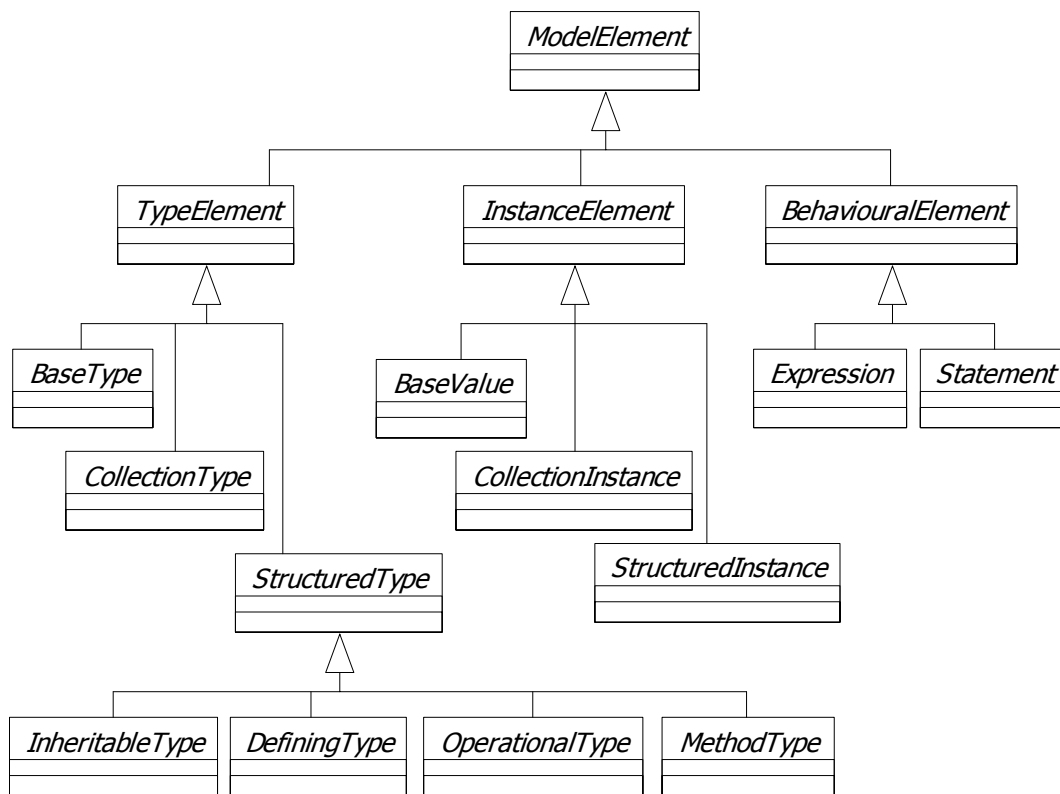
The choice was made to make the knowledge modelling elements used conform quite strictly to the UML definitions and to be quite complete in the different kinds of elements offered. In fact, conformance goes further than many other object-oriented tools. Providing an executable object-oriented specification implies providing the same level of object-oriented modelling concepts in an object-oriented language. Compare this for example to C++ or Java that may offer classes, but require significant implementation for association or aggregation relationships between classes. The difference between an object-oriented model and an object-oriented implementation is therefore significant. Offering a model that at the same time is an implementation removes this difference and allows the realisation of the model to remain at a high conceptual level. In supporting the use of an object oriented by non-computer scientists, this is thought to be an essential factor.

The structure of an object-oriented model allows it to function as the skeleton for attaching different kinds of knowledge representation. An attribute can be equipped with different inference methods, much as the same way a parameter can be the

subject of a number of decision tables in the KBE. The object-oriented model can then function as the universal glue binding these knowledge representations together.

## 5.2.2 Meta Model

A model is defined by the elements that may be used in it and the possible relationships between those elements. An explicit definition of this is called a meta-model. Where a model is a semantically complete description of a system, a meta-model's system is a model itself. UML even includes a meta-meta model, the Meta-Object Facility (MOF). It underlies a three-tier meta-meta model, meta-model and model organisation. This is not employed within IO, but because of the greater number of elements that are discerned with IO's meta-model and the increased complexity of the interrelationships, this meta-model will be given some explicit attention.



**Figure 5-6 IO Meta Model**

There are two ways of using a meta-model. The meta-model can be used through creation, i.e. the elements are made up from building blocks provided by the meta-level, or by inheritance, where the elements are derived from meta-model elements. Figure 1-1 above shows all the abstract classes in IO's inheritance tree, describing the divisions that are discerned within it. Each descent to a lower level adds some features or refines that which has been defined at higher levels of abstraction. The top element is `ModelElement`. This element is the root of all elements in a knowledge model with IO. It introduces a small number of functions, which 1) allow elements to be given a unique identifier (GUID), 2) allow each element to be translated into an XML/HTML representation, and 3) allows them to save their internal state in support of the Memento pattern. Furthermore, it contains support for the MVC architecture.

The first division could be clear from the figure: three main classes of element can be seen, topped by `ModelElement`. The main divide is the distinction between `TypeElement`, `InstanceElement` and `BehaviouralElement`. Types and instances are connected as all instances have a type that defines their composition, and constrains the functionality of the instance. The collection of types and instances are organised largely analogically, therefore in principle for every subclass of type there is an equivalent subclass in the instance hierarchy. A class at the same level is that of Behavioural elements. These elements are part of the IO language. All statements and expressions are transformed into reified structures that can be queried and manipulated like any other element in the model.

Besides these classes, there are a number of auxiliary classes that support the other classes described until now. These will mostly be left out of the description to simplify the discussion of the model elements.

### 5.2.3 Types and Instances, Attributes and Fields

Before starting the discussion of each of the individual categories of elements and their members, a more general examination will position these elements in a broader perspective. A view on the interconnections between the types, instances, attributes and fields is aimed to provide a context for the specific individual elements in a knowledge model. An important set of relationships in any knowledge model developed in IO is the correlation between types, instances, attributes and fields.

A type is a description of instances that can be constructed with it "... which allows the user to build composite type definitions from existing types and attributes, starting from some set of primitives" (Ringland & Duce 1988, pp. 228). A type within an IO model is known under a name. A type can be used to create instances and can be used in the declaration of attributes.

An instance of a type is an individual element that derives much of its behaviour from its type. Many different kinds of type exist as meta-types. Either these can be used to create a new type, a basic type such as `Integer` and `Real`, or types that are more complex, based on a `Class` for example. A user-defined `Integer` type for example can be given a domain, functioning as a constraint placed on instances of that type. A `Temperature` type based on `Real` for example can have a domain, from `-273.15` to `infinite`. This means that any instance of `Temperature` can only assume values in that domain.

Many types provide extensive mechanisms for their specialisation in the development of user types. The structured types are a category of types that allow for the use of attributes. A structured type is the only kind that can declare a set of attributes. An attribute is a combination of a name and a type, `year_of_manufacture` of type `Integer` for example. Any type, whether it is an existing type or a user-defined type, can be used in the declaration of attributes. A type `Car` based on the `Class` meta-type determines the manner in which a specific instance of car `myCar` will behave. The `Car` class may have declared a `brand`, a `year_of_manufacture`, a `colour`, etc. An instance such as `myCar` has values for each of these attributes, in structures called fields. This predisposes the system to also include different facts about the car as default, which are not put to the user as questions.

Thus, `myCar` may have values for each of the attributes of the type, in fields, so that it can be queried on each of these. This allows `myCar` to be represented as a `Fiat`,

built in 1990, with a black colour. In an instance of a structured type that has attributes, a number of fields are added that conform to the specification in the attribute declaration. The attributes of the `Car` class is only the specification, it does not have a value. The values are specific for each instance of `Car`.

The majority of an IO model consists of such structures. The user can define different kinds of type, and several of these fall in the category of structured types. The systems top-most element is a `Project`, which can contain `KnowledgeBase` types. Each of these is home to a number of user-defined types and attributes. These user-defined types form the brunt of what an expert might consider important concepts in the domain. The `KnowledgeBase` may have an attribute `Car`, which means that an instance of that `KnowledgeBase` has a field, containing a reference to a `Car`, possibly `myCar`.

Types do not have to be limited to mere descriptions of a set of conforming instances, but can also ‘have a life of their own’. They can provide useful functions by providing for behaviour that overarches the concerns at the level of instances. A type can for instance provide for access to all its instances. Every type can provide this functionality to a user of the model. It can also be more specific and user-specified in nature. For example, where a set of instances may have a temperature, the type can provide the average of all temperatures. Something that is a type merely plays the role of type, but it may also be an instance at the same time (as it is when perceived from the meta-level).

In fact, the types in IO provide an extensive API for use within a knowledge model, through the IO Language. This allows a user to define modelling actions within the knowledge model. Using the `Project->Define(CarKB, knowledgebase)` command adds a new knowledgebase to the project, for example. The knowledge model can therefore make changes to the type-level of its own model, through these reflective facilities.

#### 5.2.4 Base Types and Values

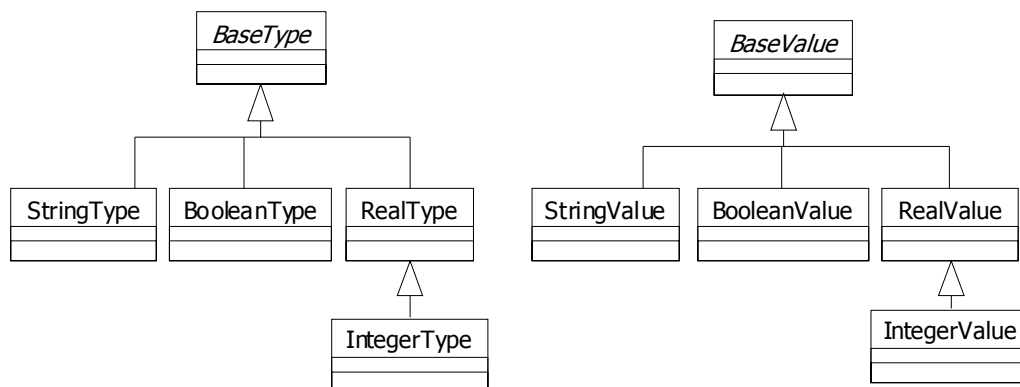


Figure 5-7 Base Types and Values

The collection of base types is almost a replicate of the types found in the KBE: `String`, `Boolean`, `Integer` and `Real`. The difference here is that the types themselves are present in the system, and not options in the parameter.

Each of the base types has a domain, which can be set to restrict the values that any attribute with that type can assume. It is possible to define subtypes of these base-

types, for example *Temperature*, as a *Real* with a domain defined as  $X > -273.15$ .

### String

A string can represent any textual value, and in its standard form it does not know any restrictions on what value can be assumed. A sub-type of *String*, *Colour* for example can declare a domain of {"red", "yellow", "green", "blue"}.

### Boolean

Booleans are used to represent basic truth-values. Sub-types of *Boolean* can be used to re-define the display representation of these values, for example, {"Yes", "No"} instead of {true, false}.

### Real

A *Real* can represent any real valued number, similar to the implicit *Real* type in the KBE. The domains can restrict the values to a specific range.

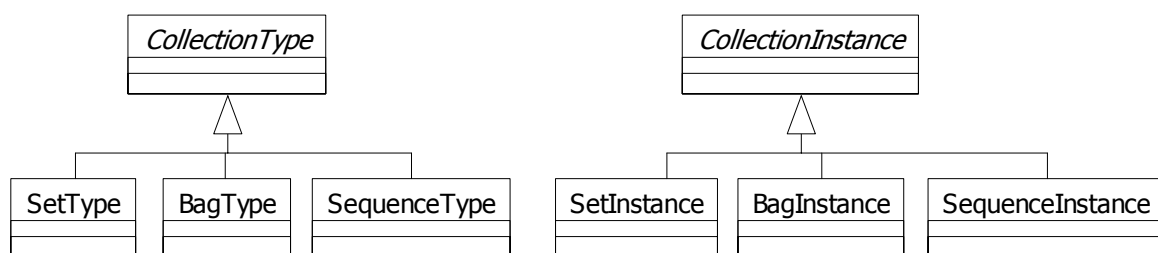
### Integer

An *Integer* can represent any whole-valued number, similar to the implicit type in the KBE. The domains can restrict the values to a specific range.

The *Integer* is derived from *Real* in this model, as it constitutes a restriction on the values that an integer can assume in relation to a real.

## 5.2.5 Collection Types and Collections

Collections denote containers for a number of elements. In their different forms, they are often part of the capabilities of object-oriented systems. Compare for example the *Container* packages in Java. IO knows three types of collection: *Set*, *Bag* and *Sequence*. Any of these containers has an item-type, the type of element contained with the collection.



**Figure 5-8 Collection Types and Instances**

Each collection supports a series of functions to query the collection, gain access to its contents, and create selections of the collection. As an example of the latter, `<collection>->select(<expression>)`. The `select` operator selects all the elements in a collection that when the expression is applied to it, returns true. For example `rooms->Select(temperature > 10)`, selects all rooms for which the temperature is greater than 10. Other such functions also exist.

**Set**

A set can contain only one single copy of any element, and may not contain any duplicates. Furthermore, it cannot be enumerated. The first two examples below are identical when compared.

*Examples*

```
Set of Integer : {0, 1, 2, 3}
Set of Integer : {3, 2, 1, 0}
Set of Room    : {room1, room2, room3}
```

**Bag**

A bag is similar to a set, but it releases the constraint on containing duplicates. The first two examples are identical to each other.

*Examples*

```
Bag of Integer : {0, 1, 1, 0, 1}
Bag of Integer : {1, 1, 0, 0, 1}
Bag of Integer : {1, 1, 0, 0}
Bag of Room    : {room1, room1, room2}
```

**Sequence**

A sequence is an enumerated array of elements, which may contain `nil` as a value. Access to individual items is possible by using an index: `<sequence>[0]` or `<sequence>->at(0)`.

*Examples*

```
Sequence of Integer : {0, 1, 1}
Sequence of Integer : {0, 1, 1, 2}
Sequence of Room    : {room1, nil, room2}
```

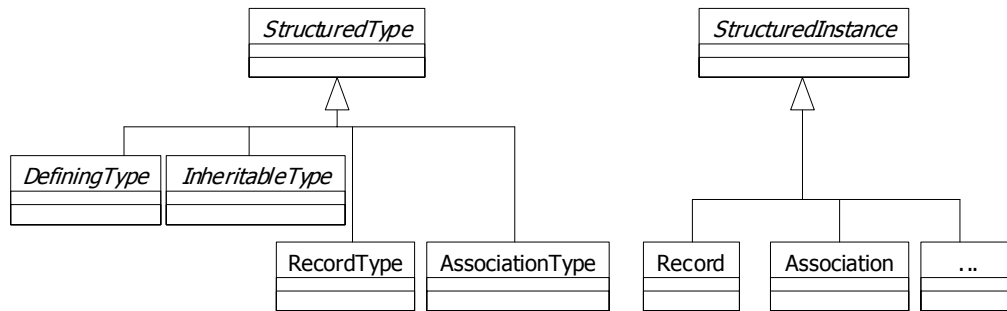
The different collection types have a number of functions that enable their transformation into another type of collection, for example from a set to a sequence: `<set>->asSequence()`. These operations create a copy of the original container, adjusting it for the target container. This may include removal of duplicates, or the introduction of order among the members. In the case of ordering the members, no guarantees are given that subsequence transformations will yield the same order.

**5.2.6 Structured Types and Instances**

Structured types allow the declaration of attributes. A structured instance contains fields for each of the attributes of its type. An attribute is the description of a named slot of a specified type; each instance of the type holds a named value of the type.

By virtue of the attributes, a concept described by a structured type can be given definition, by providing it with a related set of attributes, conceptually connected to a single instance of the type.





**Figure 5-9 Structured Types and Instances**

Within IO, many different structured types exist each with its own specialised take on the interpretation of concept. Some of the less involved descend directly from `StructuredType`. Others introduce more behaviour, and are discussed in later sections, such as the `DefiningType` and `InheritableType`.

### Record

A record is a basic form of a collection of attributes that are conceptually dependent. Some basic units such as complex number can be represented this way, or for example the date. From an object-oriented perspective, this is an obsolete structure, obfuscating the nature of an object-oriented design. However, in some cases a light escape vehicle such as a record can provide a more efficient basis for certain types of calculations. It is present in the system but is primarily intended for internal use.

#### Examples

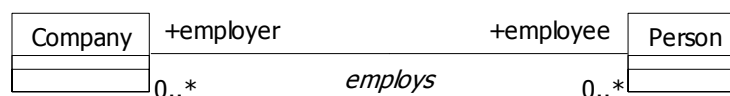
```

complex.real: Integer
complex.imaginary: Integer
time.day = 31
time.month = 12
time.year = 2001
  
```

### Association

An association is a relationship between a number of classes. By virtue of that relationship, links can be declared to exist between instances of those classes. In other words, these relationships may have attributes of their own. There are binary and  $n$ -ary association, which within IO are treated identically. Associations are especially useful for specifying navigation paths between objects.

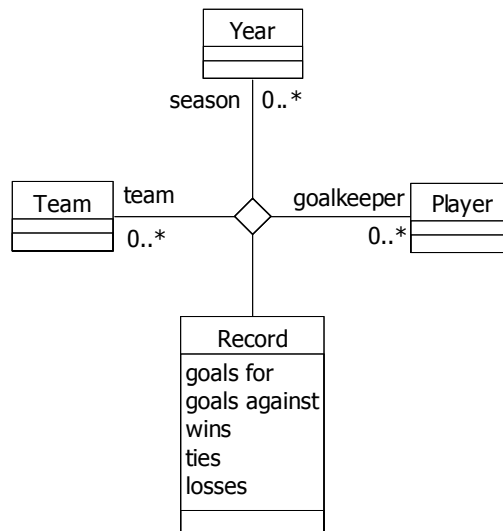
Inclusion of association is to prevent users requiring to implement different relational and ownership relationships, and instead enable them to operate with such relationships at a relatively high level of abstraction.



**Figure 5-10 A Binary Association**

Binary associations are by far more common than  $n$ -ary associations. As is visible in the figure just above, in an object-model an association is a solid path connecting two class symbols. The class type will be discussed below, as one of the `InheritableTypes`. The association 'employs' will introduce a number of pseudo attributes to the classes, namely `employee` with `company`, and `employer` with `person`. This allows the

formulation of the following types of expression: `aCompany.employee` and `aPerson.employer`. This provides a set containing the employees of one specific company, and the set of different employers that one specific person may have. It is also possible to access the objects taking part in the association through the name of the association itself: `employs.employee` and `employs.employer`. These yield all the Persons employed at any Company, and all the Companies that employ a Person.



**Figure 5-11 An N-ary Association**

In an  $n$ -ary association, a central diamond is shown to connect all vertices, as in the above graph. This also shows an association class, which normally in UML denoted a class like any other which is connected to each of the links present in the association, and each instance object of that class derives its identity from the existence of that link, i.e. if the link ceases to exist then so does the object. In IO, the association has a structured character and can therefore declare all kinds of attributes.

#### *Examples*

`company.employees` : Set of Person

All the persons employed by the company

`person.employer` : Set of Company

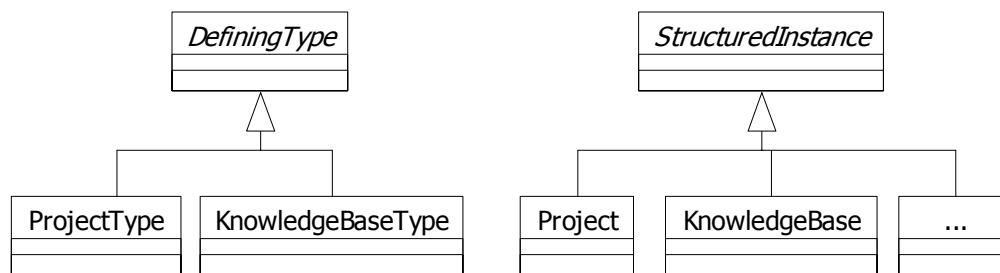
All the companies that employ the person

`Employs.employer`

All companies in the whole `employs` relationship.

### **5.2.7 Defining Type Elements**

These types of the model are those that have the capability of defining a new type. This means that they are the only ones enabled to introduce new types of elements and are the owners of these types.



**Figure 5-12 Defining Types and Instances**

With some restrictions, these types can define all of the types that are described in this chapter:

```
<definingtype>->Define( <string>, <meta-type> )
```

This defines an element of a certain meta-type (e.g. class, projecttype, knowledgebasetype, ...). This new type is then available to any element wishing to declare an attribute of this type, by reference to its name.

### Project Type and Instance

A project is the root element of the knowledge model. It can only define knowledge bases and declare variables of the knowledge-base type. It serves only to organise a number of knowledge base types.

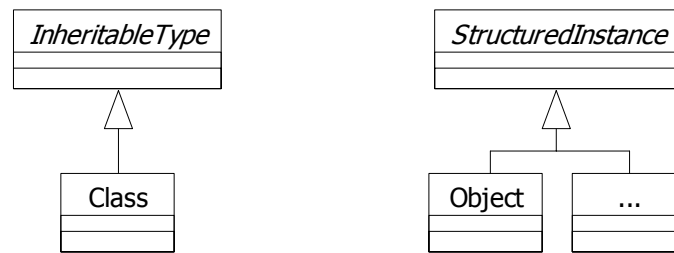
Its instance, Project is a normal instance. One of these instances is always present at any time, to enable the instance level design modelling. This involves the ability to provide an initialisation of the knowledge model, prior to consultation of the model.

### Knowledge Base Type and Instance

A knowledge base serves as a general purpose mechanism for organising elements in the model into meaningful groups. Knowledge bases may not be nested within other knowledge bases. Larger knowledge models offer special challenges in modelling. A natural upper boundary exists to the number of element that should be part of any model. This depends on the complexity of the elements, but at a certain point the classes, associations and other types become too complicated to understand, and thereby cannot be easily modified and maintained. Within UML, packages were introduced to solve this problem. A packaged delineates a number of related elements. The knowledge base that is presented here also delineates a series of elements, beyond that can also have state and behaviour of its own. This enables a knowledge base to operate as any other structured type. A knowledge base can serve to solve a sub-problem. This solution can be used by reference to its own interface rather than to elements that are contained within. This is thought to simplify the issues of encapsulation.

### 5.2.8 Inheritable Types

The elements in this category can inherit from another inheritable type of the same denomination (e.g. one class to another). To this end, they each have a parent, as IO knows only single inheritance. By virtue of inheritance, these elements form a generalisation hierarchy.



**Figure 5-13 Inheritable Types and Instances**

Each inheritable type is a partial specification of its instances, where it and all the parents of that type create the complete specification. A child element inherits the features of its parent and indirectly those from its ancestors. An *Inheritable* type can also partially re-define parts of the inherited declarations, by overriding them. This allows an *Inheritable* type to employ its parent's definition as a default, to be overruled when necessary. As can be seen in the figure above, the instances of *Inheritable* types descend from *StructInstance*. The approach to incremental definition of an *Inheritable* type has not effect on the instances. To an instance, it seems like its own type introduced its complete definition.

Currently, the only inheritable element in IO is the *Class*. This may change in the future as new types may be introduced, like *Interfaces*, or when other existing types are changed to *Inheritable* types such as the *Association*. *Inheritable* types provide a model with parsimony of expression, where knowledge needs to be expressed only once but used in many different places. From a maintenance point of view, this means that changes need to be made only in one place. The verification of those parts of the model affected is also simplified to concern only those that use that type, instead of those that use a number of types. *Inheritable* types are the class of structural elements that allow declarations and definitions to be shared within an inheritance hierarchy.

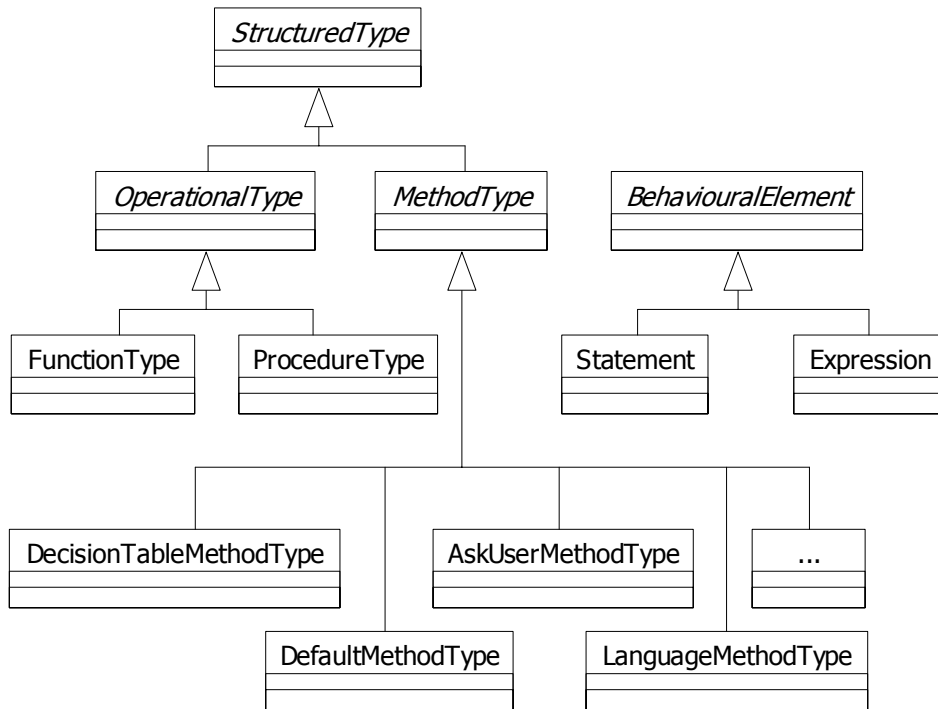
### **Class**

A *Class* is an element or concept that defines a set of instances, which share the same attributes, operations and inference methods. As such, a class defines a concept within the model. By virtue of its inheritable nature, it can have a parent class and several child classes, thereby defining an inheritance tree. Different families of classes can be created in that way.

A class adds the possibility to be incorporated into relationships to the inheritable type. This allows a class to play a variety of roles in a model. A person for example can feature as family, as an owner of some estate, or as employed by zero or more companies. With these relationships come attributes and behaviour. A person marries, has children, changes employer, buys and sells property. What is clear from this account is that relationships are defined statically as possibilities, whereas the behaviour of the model dynamically asserts and modifies the real relationships for the instances of the class.

### **5.2.9 Operational and Behavioural Elements**

This collection of elements is somewhat spread out over the hierarchy, but they belong to a common category nonetheless.



**Figure 5-14 Operational and Behavioural Elements**

Each of these elements supports the ability to be evaluated to perform some dynamic behaviour and perhaps change the model. This may be a value assignment at the instance level or a structural change at the type level. The different behavioural elements possess important interrelationships.

### Procedures and Functions

A procedure or function allows the definition of operations in structured type elements.

```

room.heat(degrees:Integer); // procedure
room.getHumidity():Integer; // function
  
```

**Figure 5-15 Procedure and Function Example**

They constitute a named reference to a method and have the capability to declare formal parameters. A procedure has no result and can therefore not be used in an expression, only in a statement. A function is like a procedure in all ways except for the fact that it has a fixed result parameter. This means that a function call can only be found in an expression.

### Method

From the outside, a method is a black box for determining the value of one or more attributes. The innards of a method are of no direct interest to the remainder of the model. An implementation of the innards of such a method can make use of the services that are available to the methods, like the language service. Thereby, statements and expressions can be incorporated into them. This is not a necessity,

however, and an implementation of say a Neural Network method can be devoid of any reference to elements internal to IO. This approach allows many more methods to be incorporated into IO. By adding to the list of methods, IO can be extended with different knowledge representation and reasoning components. A method may be incorporated into a function or procedure, if the method can set more than one attribute simultaneously.

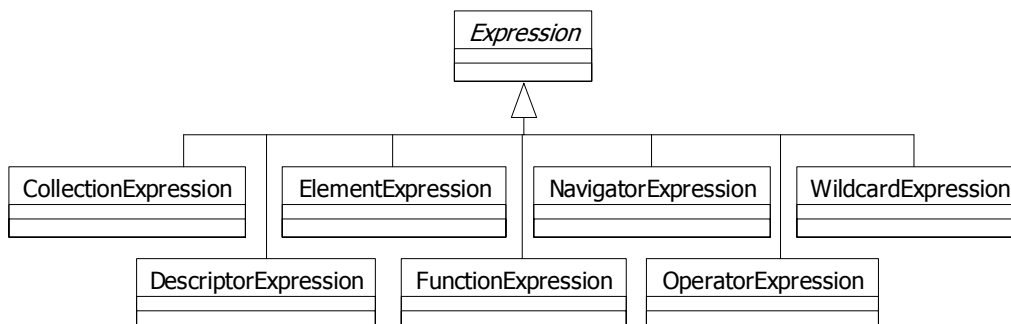
Name	Description	Number of attributes
DecisionTable	A decision-table implementations that employs IO Language Statement and Expressions in the conditions and actions.	Multiple
AskUser	Request for information to external system	Single
Default	Simple expression as value of element.	Single
Language	Number of statements expressing a procedural approach to deriving a value for the parameter.	Multiple
Database Method	Determine the value of a field by querying a database using an SQL query.	Multiple
External Method	Link to external function or procedure in a dynamic link library (dll).	Multiple

**Table 5-1 Method Types**

As is seen in the above table, some methods are used to determining a single attribute while others can be used to determine a number of attributes in unison.

### Expression

Before delving into expressions and statements, a more general remark is required. In the KBE, the expressions and statements are retained in a textual format, and interpreted when needed by a parser. By virtue of its live model, IO requires significant feedback from these expressions and statements. Therefore, it has reified them into objects forming many statement and expression trees. As the language is quite mature in capabilities, there are a great many of such statement and expression elements.



**Figure 5-16 Expression Items**

Whenever an element that contains an expression, for example a decision-table, is changed, it sends a request to the language service for an update. This parses the textual expression into the statement and expression elements. These elements have links to the remainder of the model. The elements also have the capability of rendering themselves into a textual format yet again. When an attribute that features in the expression changes its name, the expression will change accordingly. Through

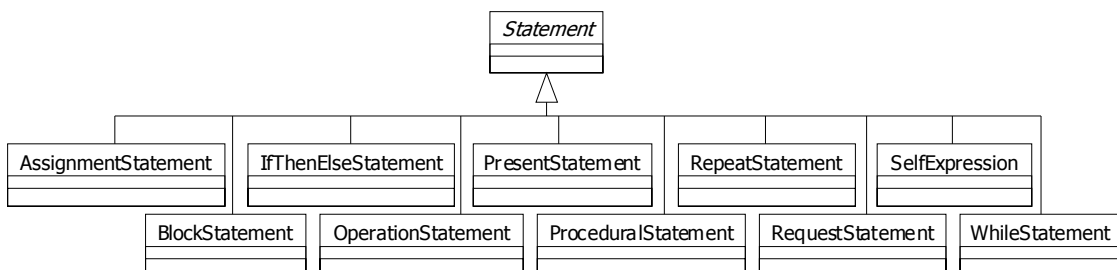
the MVC mechanism the user interface can be notified of this change. Furthermore, when that same attribute is deleted from the model, the expression will be aware of that change and can inform the model that the expression has become incorrect. This is the manner in which the live model and MVC cooperate, to create an advanced notion of feedback on each user action.

Name	Description	Example
CollectionExpression	Constant collection expression.	Bag of { x, true, false } of Boolean
DescriptorExpression	A reference to a name in the model.	temperature
ElementExpression	A constant expression.	6
FunctionExpression	A call to a user-defined function.	fac( 10 )
NavigatorExpression	Navigating over the names defined by the types using the <i>dot</i> notation.	x.temperature
OperatorExpression	Calling an operator, of which there are some 67 different ones.	x and y, x+y, x/y
SelfExpression	Calling the context of an expression or the context of the surrounding element.	self.temperature
WildCardExpression	A special reference to an expression that is present elsewhere.	# = true

**Table 5-2 Expression Elements**

An expression item is a special entity, as it has no state and is evaluated using the context in which it is called. This context can be any element in the knowledge model. An expression always returns some value on evaluation of one of the defined types, and is capable of providing type-information. This includes information on possible errors in the expression. The above table provides a list of the available expression items.

### Statement



**Figure 5-17 Statement Items**

A statement is a stateless element, evaluated using the context in which it is used. A statement executes some behaviour, through repetitive, conditional or linear action, that can change the model. A statement itself does not yield a value, although the behaviour of the different statement type does depend on values returned by contained expressions.

Name	Description	Example
AssignmentStatement	Assign a value to an element	<code>x := 1;</code>
BlockStatement	Organises a number of statements	<code>begin   x := 10;   ... end;</code>
IfThenElseStatement	Control a bifurcation by a Boolean expression	<code>if room.temperature   &gt; 50 then   room.heat( 10 )   else door.open();</code>
OperationStatement	Execute an operation by name.	<code>room-&gt;SetParent(   space)</code>
PresentStatement	Fires an event to show an element within the model to an outside party.	<code>Present( x )</code>
ProceduralStatement	Calls a named, user-defined procedure.	<code>room.heat( 10 );</code>
RepeatStatement	Repeats a number of statements contained until test expression is satisfied.	<code>repeat   x := x + 1;   y := y * x; until x &gt; 100;</code>
RequestStatement	Fires an event to the request a value for a field to an outside party.	<code>Request(   room.temperature );</code>
WhileStatement	Repeats a statement until the test expression is no longer satisfied.	<code>while x &lt; 100   x := x + 1;</code>

**Table 5-3 Statement Elements**

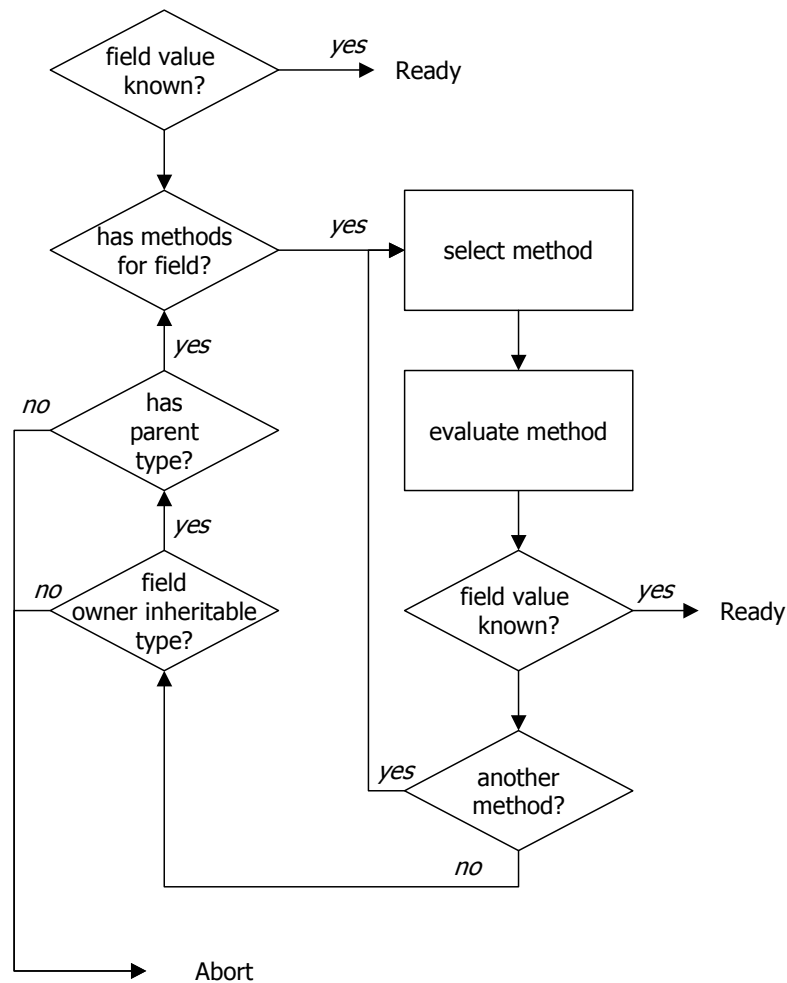
The Operation statement is special because it is a holdall for many different operations that are defined. Some 32 different operations are possible, although not all types allow each of these operations to be performed on them. For example, the `SetParent` operation is only valid for inheritable types.

Also special among the statements are the assignment and request statements. These set the value of an attribute explicitly and when they occur within a reasoning method, they are the reason to register the containing method as a valid inference for the attribute being set. The exception to this rule is when the method is contained by a function or procedure.

### 5.2.10 Inference Process

The inference process is quite similar to that of the KBE although some functions are perhaps present in a more explicit form. Actually, the inference process has been based on that of the KBE to make the transition easier for previous users of the KBE (compare the KBE process described on page 115). By provision of the inference service, it is possible to switch to another inference process, or to make it a configuration option for legacy knowledge system.





**Figure 5-18 IO Inference Process**

The inference process described in the figure above shows the system going through the list of available inference methods. In the case of an inheritable element, such as an object's class, the superclasses are also included in the search for an available inference method.

The actual operation of each of the methods is left to the particular methods. Each method is capable of accessing different aspects of the knowledge model, and can execute statements and expressions. A change made by a method in the end is assigning a value to the requested field, but a method can also set other fields value as well. A method does not have to assign a value, and in that event, the inference process continues with the next method until the set is depleted.

### 5.2.11 Communication

The communication is abstract in so far that the IO model component communicates with the outside world through a number of events, in a similar way as the KBE component does. However, where the KBE sometimes communicates through textual information, IO uses the actual elements. The present- and request-statement are the two communication elements of the language that the knowledge model uses to explicitly model communication acts. These can be used to initiate a presentation of information in the knowledge model or conversely to obtain information about the knowledge model from the external systems or users of the system. The other

communication modalities, query and receive, that are initiated by the external system or the user are supported by the consultation API.

### **Present**

The present statement is similar to the display option in the KBE. However, the intent of the display option was to show a textual string of characters. The present statement accepts any element from the IO model, whose actual representation as text or as an image or any kind of format that is used is determined by external facilities, i.e. determinable by the developers of the knowledge system. Support for some internal functions that create default visualisations of elements as XML or HTML are provided to make default visualisations easy. The choice of how to actually present an element in the model is up to the developers and can be to either a default textual form or a proprietary format.

### **Obtain**

The request statement implements the *obtain* modality as it is known within CommonKADS (see Figure 3-3 CommonKADS Communication Modalities) It is similar to the present statement but only operates on fields in the model. The field's attribute maintains certain information such as the prompt, the explanation and the domain of the attribute, as an expression for a set of alternative values. The information that can be used to support the visualisation of the request. However, the external system may decide to answer the question from a database, where such information is superfluous.

### **Query**

The functions in the API allow a user to formulate a query in any number of ways. The knowledge model can be queried using its goals, through any of the fields in the knowledge models, or by having the model execute a script consisting using the IO language. Either of these methods can start an inference process to determine one or more aspects through reasoning.

### **Receive**

The ability for the model to receive information is provided by allowing parties to set values for fields in the model. This can be done directly on a field or in the same way as the formulation of a query through an expression or statement in a script. A statement is acceptable in such a case because no value needs to be returned.

## **5.2.12 Language**

The language that underlies the formulation of statements and expressions within IO is based on the Object Constraint Language (OCL). It has been extended to include the constructive actions that are allowed on the different elements within the model, propelling it beyond its initial intent. The OCL is a text language originally intended for specifying constraints and queries in UML models, and was not intended for writing actions or executable code (Jacobson, Booch & Rumbaugh 1998). OCL allows navigation expressions, Boolean expressions and other queries. The full OCL is a thoroughly worked out construct that can be parsed like any other such language. It contains a large number of predefined operators on collections and primitive types.

A complete description of the OCL syntax and semantics is found in (Warmer & Kleppe 1999).

The OCL, although not intended for that purpose, gives a good control, specification and query language.

```
item.selector
```

```
item[<integer>]
```

```
item.selector(<argumentlist>)
```

*Examples:*

```
room.temperature
```

Get the temperature of the room

```
house.rooms[0]
```

Get the first room in the list of rooms in the house.

```
house.rooms->select(item.temperature > 15)
```

Select the rooms in the house that have a temperature above 15 degrees Celsius.

### 5.2.13 Conclusions

As is clear from the description of the many elements with which a knowledge model can be constructed in IO, the expressive capabilities far exceed that of the KBE, adding to the initial difficulty that may be attached to the use of IO. The main source of this is the sheer number of elements whose purpose and meaning have to be known to be able to work with IO. Nevertheless, a subset of the functions allows usage of IO in an almost similar way to the KBE itself, as IO subsumes many of its functionalities. Using this subset could reduce some of the significant learning curve on these subjects. However, using this set exclusively reduces the benefits that may drive from the other facilities.

## 5.3 Knowledge Modelling

This section describes the common knowledge modelling process when working with IO. In a way, this is a usage scenario, in an idealised description of the functions that IO should support. However, this description is also based on actual experiences such as those described the next chapter.

### 5.3.1 Starting Off

A fresh new project in IO contains no more than the library of system types such as Boolean, Integer, etc., basic functions and a new empty knowledge base. This is a valid knowledge model, but it cannot be consulted for any goal. As a project can only define and declare knowledge bases, a minimal operational knowledge model must contain at the very least a primary knowledge base and a variable of either the knowledge base type. Furthermore, this knowledge base must contain at least a single attribute of one of the base-types. Equipping this attribute with a reasoning method of any kind will create the smallest possible executable system.

The additional work is therefore not a great deal more than is required in the KBE, but the actions are perhaps conceptually more complex as the number of actions to perform is greater. A further difference is that where the KBE worked from a

decision-table as the main representation form, and added new parameters to the domain model when required in any of the decision-tables, the procedure here is quite the reverse. From the start, the main entry point is a domain model, structured in knowledge bases and other structural elements. The domain model can be developed to an elaborate representation of the domain, giving a plain perspective of the relevant concepts and relationships.

No reasoning method has to be added for this to be a meaningful model, and in this form, the system can already give some meaningful responses to queries in a form that compared to results from an object-oriented database. The possibility to create elements at the instance level and the reflective character supported by the type level elements, enable general queries. There are no ways to determine values for elements through inference and reasoning, but at the level of types and instances, such information can be returned.

### **Classes**

The likely start of any knowledge model is to locate some structure in the classes involved in the model of the domain. These important concepts determine to a great degree what can be discerned within the domain.

### **Inheritance Relationships**

After the core-set has been found, the relationships between the classes can be defined. The inheritance relationships are oft located first, in unison with the core set of classes.

### **Associations & Aggregations Relationships**

After this, associations and aggregations become part of the model's evolution, determining further the static relationships existing between the different classes. This occurs more or less at the same time as the attributes.

### **Attributes**

An attribute is considered to a relationship of a kind that is less pronounced, because the structural type element such as a class 'owns' the attribute. The attribute does not exist on its own, but is defined by its presence in class, for example.

### **Reasoning Methods**

Adding reasoning knowledge and attaching these to certain points in the structures contained in the knowledge model, creates its dynamic nature alongside the static nature of the structure model. It allows assumptions to be made based in the knowledge contained in the reasoning methods. These can be without structure, keeping with the backward chaining control that the system offers.

### **Procedures and Functions**

In many systems, some form of structured procedure is required when solving a problem. In the KBE, decision-tables served as a double role as structure-tables, which subverts the original intent of the decision-table for practical needs. It must be stressed, that this is considered practical and not incorrect in the KBE. It is necessary for resolving certain problems, and for remaining close at least to the task structure

that is perceived by the expert. In IO however, other mechanisms are offered that should remove the need to use such implicit structure in tables.

Task methods may be created any kind of method type, embedded in a function or procedure. For example, a decision-table can be used as the body of a procedure, and in this form determine the order of reasoning steps. Within the context provided by IO, the difference is that the method is explicitly embedded as a task procedure, and that the choice for a decision-table as the most appropriate form is made with other method types available as well. Other approaches may also be employed, but a choice is made for a decision-table explicitly. As soon as that choice is no longer the most appropriate one, a change can be made to another format. This does not affect those parts of the knowledge model that call upon the procedure or function to perform a specific task.

### 5.3.2 Developing a Knowledge Model

After an initial model has been created, from a basic model further development can be made in many different ways. The modelling style that is worded in the description of the elements above speaks from a presumed object oriented model in mind. However, this does not have to be the start of a knowledge model. Therefore, the two modes of operations discussed here will first show a design-less approach before discussing a design-focused approach.

#### Design-less

Much like the KBE, the knowledge base itself may be used as a container for a monolithical list of facts. While this does not derive benefits from the available structuring mechanisms, this may appeal to a sense of simplicity. This may moderate the otherwise considerable learning curve. As the mode of operation using this style is possible and creates valid knowledge models, it may well constitute a convenient starting point just before structure can be discovered to emerge from the long list of facts.

This can also be part of a migration path from the KBE to IO. This approach constitutes a design-less approach that can also be employed when experts start creating a system and don't want to be bothered initially by overt amounts of analysis and design specifications. As a knowledge acquisition tool, this can operate quite well. Eventually this can evolve into more structure, either by separating over several knowledge bases, which creates entities of great granularity, but expectations are that such an approach will freely give way to more structured approaches.

Although this minimal modelling scenario is a degenerate form of use of the IO system, it is important to stress that the modelling approach found in the KBE works well for certain types of projects. It does not do to force users to employ other methods. Structural knowledge needs to be discovered as well, and this structure does not exist beforehand in all situations. In many cases a structure can be proposed, but it may be preferable to leave structure for what it is, either temporarily or indefinitely.

By allowing the smallest common denominator, one can educate users, in small steps that can be applied in operations that are more complex later on. In addition, division, decomposition and abstraction can be quite difficult concepts to grasp. They are not pre-existing things in many domains, and do not necessarily represent the best way of examining a domain. This contrary runs to the high regard for analytical accuracy and generality that many computer scientists hold to be true.

### Design-focused

Another approach is the development of an incomplete but initial model, based on classes and associations, imposing structure. This form of model can also emerge from the previous approach, and could be seen as a continuation of the previous. The ability to extend the model, with parts of medium granularity, i.e. classes and associations and the conceptual nature of the models, is not very difficult and can be done on the basis of necessity rather than based on a complete design.

Adding a new class is as hard as adding a knowledge base. Its difficulty is determined more by choosing to compartmentalise a certain aspect of the knowledge model into a separate entity than anything else. The more structure is added to the model, the easier it will become to choose places for the remaining parts, and to determine whether a certain concept or association is missing. In fact, similar to the decision-table, the knowledge model's classes and relationships will expose weak spots and omissions when such an additional concept is necessary. Just as a blank decision-table column, or missing domain alternative. It is perhaps not as explicit, but the same basic principle applies. The scale is larger and accommodations can place the responsibility with for a certain aspect with an entity that is not truly suited, but for which it is at that time correct or merely practical. Because the knowledge model incorporates knowledge representations such as the decision-table, at lower scale this exposition of flaws and incompleteness is also included. The decision-table plays the role of an easy to understand visual representation, but is now one of many different tools at the disposal of the modeller, who now has gained considerable choice in the form and content of the model. This choice most likely is also one of the root causes of the systems difficulty.

This illustrates one of the main differences between the KBE and IO. Within an IO model there are many different levels at which some aspect of the domain can be defined, and support for modelling is given at each of these levels. A particular important concept can be a knowledge base, a class, a relationship, an attribute, etc., or even split out over different elements in the knowledge model. This freedom is gained at the cost of understanding these options and making an explicit choice between them.

The development of a knowledge model with a focus on design means adding new types using knowledge bases, classes, and relationships to create a structure to the model. These can be shown to exist in the domain or are discovered by working through the model. In many cases, the first choice may not be the correct one. Therefore, it is very important that the model remains fluid to allow changing one's mind.

Further development of a structured type is no more than adding content in the form of attributes, which is a task known from the KBE and from the earlier simple approach. This can be done in the same *need-to-know* basis as the KBE, deciding to add an attribute only when required in a reasoning method. It is possible to add a reasoning method to a class without referring to the attribute or attributes for which it should infer a value. As each of the structured types allows for the definition of procedure and functions, it is also possible to weave the structure of tasks through a model. Placing different aspects of the task with separate types in the model gives a division of labour and allows different types to perform a task in specialised ways.

Classes play a special role in defining their internal composition as they allow inheritance. This enables them to derive much of their operation from a parent class and allows a very concise formulation of knowledge. The child class needs only to express the difference between it and the parent class. The value for an inherited attribute can be inferred using the methods of a parent class. Reasoning methods in a class can even call the inherited reasoning methods explicitly by use of the `inherited` keyword. This invokes the methods of the parent classes to derive an initial value. The same works for procedures and functions that can call on inherited procedures and functions using the same keyword. Such features are known from other object-oriented languages as well.

### 5.3.3 Change and Enhancements

As was described in some detail, knowledge models go through considerable change and enhancement throughout their lifetime. As these activities are of a different character than development, some of the support for performing these activities is provided here.

#### Change

Change is an activity aimed at adjusting the current content of a knowledge model to perform as was intended when first formulating the knowledge. More than development itself, this then entails understanding the current operation and the desired operation. Furthermore, it requires formulating an approach that will effect a change that will adjust the behaviour. In many cases, the change originates in the behaviour on a specific situation or set of situations. The intended change needs to adjust the behaviour in these situations, while leaving the treatment of the other situations the same. Each of the types and method types has their own strategies for accepting change. The procedures for the overall model are described in this section.

One important component in making a change is that the procedure to modify certain behaviour makes it necessary to locate the place or places where the current behaviour originates. Making the model that is used transparent is an essential step, aided by the vividness of the model. Each type is a location in a map of the domain, which at smaller scale knows internal locations. The different interrelationships, parenthood, associations, usage of types, its own usage in places, and its general location as part of a greater knowledge base, constrain and assist the search.

The structure of the domain model therefore needs to be developed as that map of the domain. Its function goes above and beyond the need to *implement* the knowledge. The knowledge model needs to be elegant and recognisable as a theory of the domain. The facilities for doing so are present through the different types and alternative relationships between these types. This determines the localisation, navigability, transparency and understandability of the knowledge model. In turn, this determines the ability of a modeller to change the model. The more familiar the terrain is, the easier it is to understand and make effective changes. The more the experts experience the model as their own, the easier this task will therefore be.

In fact, certain relationships are as familiar paths, shortcuts to get from one place to the next in the model: from house to person, from person to employer. This will also affect the content of the model, as these paths will also be reiterated as part of the reasoning methods, when these are used in qualified descriptors: `house.inhabitants` and `person.employer`.

Exchanging one implementation of a type for an alternative implementation is part of the object-oriented possibilities. Design for maintenance as it was called earlier is certainly part of the options of any object-oriented model, supported by modularisation, encapsulation. The specifics of making changes to a knowledge base, or class occur on a different level of abstraction and are similar to the effort faced in their creation. The problems of an internal change made to one of these types, or to one of the reasoning methods, are comparable to those found in the KBE. Therefore, besides stressing that the same kinds of principles must be supported at different scales as much as possible, the description of such features is relegated to the previous chapter.

### **Enhancements**

In a similar way, the possibilities for enhancing or embellishing a knowledge model on a smaller scale are quite similar to that found in the KBE. Therefore, this section concentrates on object-model's support for enhancement.

Enhancing, changing the model to create new functionality, or extending the number of products known to the system is supported by the system.

Known as *extension points*, object oriented models can create specific places when new additions that are expected can be added with ease, without incurring a great cost to modify the existing knowledge model. Often these extension points are specific classes in the model that abstract a family of elements. When a new animal type is required, the model can be easily changed to incorporate it. Deriving a new type of animal is as simple as creating a class and setting its parent to that of animal or one of its abstract descendants. The differences for that animal are the only modelling requirements, and in many cases, no other change is required.

Functional enhancements are also supported. Adding capabilities to an existing structural model to assist in the inference of an additional goal, means weaving new inference methods through it. The structure of the model can remain the same or requires only minor changes and enhancements.

If the behaviour is task based, this is the same, but new procedures and functions are added to the user-defined types in the model, with inference methods to back the tasks that need to be performed. After building an initial system that performs diagnosis, other knowledge can be added that provides solutions based on that diagnosis. These two can be integrated into the same model. The structure of types defined in a knowledge model, can also be used to start another.

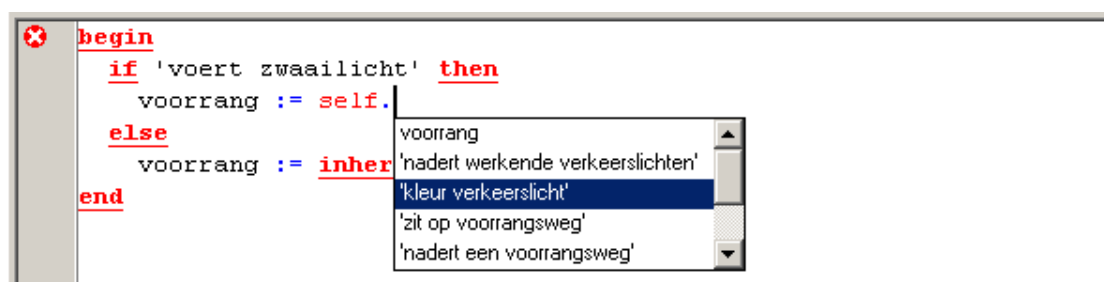
### **5.3.4 Special Facilities**

IO offers a number of special facilities that support the modelling process and aim to alleviate some of the effort in developing a knowledge model with confidence and with ease.

#### **Code Insight**

Code Insight is a term borrowed from Borland Delphi. In IO, it is used as a broad term for support in understanding and creating IO language expressions and statements. It includes syntax highlighting and a number of additional facilities to aid the knowledge modeller in producing quality knowledge models in little time.





**Figure 5-19 Syntax Highlighting & Code Completion**

Syntax highlighting changes the colour and font attributes of text in the IO language editor, making it easier to quickly understand and identify parts of the expressed knowledge. While being a very simple addition, it provides immediate feedback to the modeller on the meaning of the expression or statement that has been formulated. Beyond the styling of the text, the system also always gives an error status of the expression in the border of the editor, denoting a syntactic or semantic error. This is shown in the right-top corner of the figure.

Code Completion is the feature that allows a modeller to be provided with suggestions on how to continue the expression that is being constructed. In the image above, it is possible to see that behind the expression `self.` all possible continuations are given. This includes `voorrang`, and other attributes of `road user`. Code completion negates the need to look up these continuations and allow the user to enter these terms without making spelling errors.

The direct feedback coming from the syntax highlighting is in fact more advanced as it interprets not only the syntax, but detects when certain terms do not occur in the model, allowing missing attributes to be detected as soon as they are entered. When corrective actions are undertaken, the syntax highlighter makes it clear immediately if the actions produced the desired result. The turn around time on these types of changes is exceedingly short, compared to for example a compile cycle with error messages.

### Unlimited Undo/Redo

IO supports the Memento pattern. This in a very concise way means that it “allows the capture and externalisation of an object’s internal state so that the object can be restored to this state later” (Gamma et al. 1994, pg. 283). By virtue of this mechanism, it is possible to capture the current state of a knowledge model and return to this state on command. The memento is an object that stores the internal state of the knowledge model.

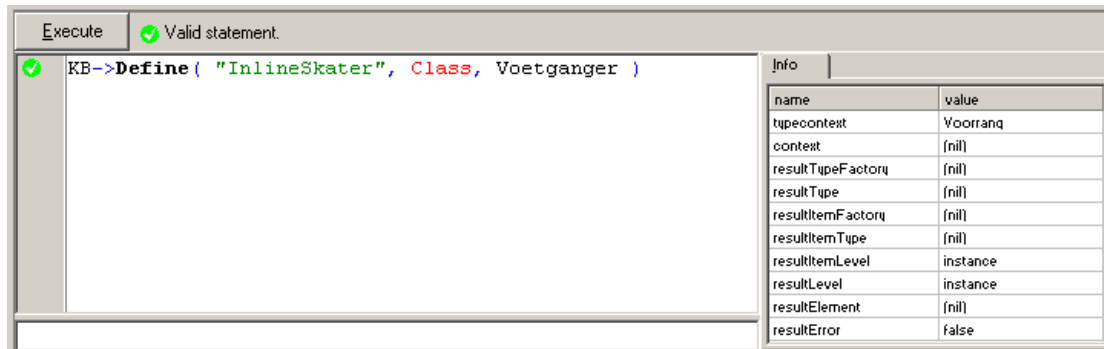
The Command pattern “encapsulates a command as an object” (Gamma et al. 1994, pg. 233). An action on the model is therefore reified as a command object. This object can be placed on an undo-stack when executed and the actions on this stack can then be reversed in order to return to previous states, effectively undoing some of the mutations on the model. Based on the Memento pattern it can easily reverse the model into an earlier state.

Using these two patterns as a basis the IO editor is capable of implementing an unlimited undo/redo mechanism. While this is a common enough feature in many professional systems, it is an essential feature when providing functionality to layman. It allows the user not only some measure of trust that errors made by him will only have limited effects, and can be safely reversed, and further makes it possible to use

this fact explicitly by exploring certain possibilities in freedom. It is essential functionality to allow learning and development to go hand in hand. It further allows users to experiment with the functionality offered by the system, which allows them to test whether a certain command ‘works for them’.

### Language Interface

One of the other advanced elements that is not directly related to knowledge representation is the language interface, which in effect is the door to the reflective live model contained in IO..



**Figure 5-20 Command Line Language Editor**

The language used in IO contains reflective elements that allow inspection and modification of all parts of the knowledge model. These facilities go so far that all operations possible through the API or by the user-interface are in principle also possible by executing statements and expressions in the language. This language interface is present in the user interface of the editor, providing part of the functions that can be used by expert users. At the right of the editor is the type information known about the result type of the statement in question.

### 5.3.5 Conclusions

The previous has shown different approaches that flow into each other with increasing levels of abstraction and complexity. It has also shown that the increased structure in the knowledge model aids and predetermines choices for representation, acting as frame of reference for each new addition as it were. Because these choices are explicit rather than implicit in numerous assumptions within the knowledge model, it is available to all that employ the model. This makes a model more open and understandable, for people outside of the circle of people that were involved in its initial development. All of the aspects of the knowledge are therefore open to be discussed and managed as to its quality.

This sections also aimed to show that although one might get an impression that IO is all about design, it is very suitable to be employed in a way that the design is discovered later, rather than imposed in the beginning. On the other hand, interpretation models such as those proposed within CommonKADS, can function similarly here, if and only of such model are seen as a kind of scaffolding that needs to develop with the model, rather than a rigid skeleton. The structure should create the frame around which changes are made, but that does not receive the brunt of the changes itself.

## 5.4 Knowledge System Development

Knowledge system development pretty much follows the same procedures as mentioned in the previous chapter. The iKnow architecture was developed to allow both KBE and IO knowledge models to be published through the same framework. For details on developing dedicated and customised applications, either standalone and using the Internet, the reader is reverted to the previous chapter. This section will discuss the support for the MVC architecture which determines the majority of features that are different or additional to the features of the KBE.

### 5.4.1 MVC Support

The MVC architecture that lies at the base of all the elements in the knowledge model generates improved support for the development of dedicated applications. The usage of the MVC architecture makes it much easier to develop dedicated applications on top of an abstract component.

All elements in the model deal out events when changes are made to them, like a new type that is defined, or when the model itself makes changes, such as assigning an inferred value. Different types of events are generated for different types of changes, allowing discriminative screening to be used before undertaking large-scale updates.

The only requirement for the application is to implement observers or listeners for these events. Support classes from the MVC part of the system allow viewers for classes to be developed that already realise the observer interfaces. This means that development of a proprietary viewer is limited to implementing a visualisation routine that is called whenever an event is generated by the model element in question.

For other uses, a separate observer non-visual component encapsulates the listener capabilities. This can be used in all kinds of visual and non-visual settings, where an application needs to be informed about changes or other actions performed on the knowledge model.

The MVC capabilities are necessary for different reasons. On the one hand, it is necessary as the changes made to the model are sometimes outside of the control of the client system, either by reflective parts of the knowledge model or by consulting the system. The visualising application will then have a hard time to keep the visualisation in tune with the actual state of the model. On the other hand, a complex knowledge model will also require alternative, co-existing visualisations of the same part of the knowledge model. Informing each of these using events is less complex than other possible approaches that require the initiative to lie with the application rather than the model.

The iKnow architecture provision for both distributed and centralised access to knowledge systems, as described in the previous chapter allows many distribution models. It allows a gradual path to develop a customised system, making it possible to place any chosen amount of effort in the development of a specific system, at any time during the development. The support given by the MVC architecture further empowers the development of dedicated consultation and design applications with ease. In addition, the support for abstract communication is also improved as was discussed earlier. As these were requirements deriving from the principles of continuous knowledge engineering, this support is perceived as both fitting and essential for knowledge system development.

## 5.5 Conclusions

The support for developing vivid and conceptually high-level models of a domain dominated the description of the facilities realised by IO. This chapter shows an advanced knowledge system development environment. Extending and reapplying the facilities first introduced by the KBE, the full set of functionality is all but basic in its support to evolve knowledge models and its abilities to support the development of professional industrial strength knowledge systems.

Many of the improvements to the changeability and extendability of a knowledge system, be it the model or the visualisation, derive from the object oriented modelling language. Coupled with the visual nature of the knowledge representation for the reasoning methods, this aims to provide a modelling language that can scale in size considerably, while retaining the fluidity and understandability required for continuous knowledge engineering.

	Vivid Knowledge Representation	Visual Knowledge Representation	Component Based Development	Model View Controller	Abstract Communication	Centralised Deployment	Executable Specification	Live Model	Full Object Oriented Model	Script Interface	Unlimited Undo/Redo	Code Insight	Adaptable Visualisation
<b>Participation</b>													
Understandability	✓	✓			✓		✓	✓	✓	✓	✓	✓	
Locatability	✓	✓			✓			✓	✓				
Expressibility	✓	✓						✓	✓	✓	✓	✓	
Changeability	✓								✓	✓	✓		
Extendability	✓								✓	✓	✓		
<b>Stewardship</b>													
Usage Information				✓	✓	✓							
Metrics					✓	✓	✓	✓					
<b>Medium</b>													
Operational			✓				✓	✓					✓
Gradual UI			✓	✓	✓								✓
Diff. Visualisation				✓	✓								✓
Dedicated Solution				✓	✓								✓
Integration			✓	✓	✓								✓
<b>Cyclic</b>													
Start Minimal						✓							✓
Small Increments						✓			✓	✓	✓		✓
Revocable Changes						✓					✓		
Scalability			✓						✓				✓
Gradual Integration			✓						✓				✓
Easy to Change									✓		✓		
Easy to Extend									✓		✓		

**Table 5-4 Features and Requirements Implemented by IO**

The advanced support facilities comprise the additional facilities such as syntax highlighting, code completion, direct error feedback, and unlimited undo redo/undo. While individually these facilities may merely provide an improvement to the

usability of IO, the set of features creates a very reactive model, where each change is meaningful and can be ascertained to its meaning. Coupled to the unlimited undo/redo the potential to experiment is very great. This is seen as a significant addition to the already short-cycle of development provided by the executable specification nature of the model.

The IO knowledge system development environment supports an object-oriented structure for a knowledge model. The realisation of the object-oriented knowledge modelling language supports a conceptually high level model by going beyond provision of classes to incorporate association and aggregation relationships. The static model of a domain created in this way can be extended not merely with attributes for static state behaviour and operations for dynamic state behaviour, but also extends such a structural model with reasoning knowledge. With the knowledge system development capabilities largely similar, IO provides more support for developing dedicated applications by increased support for an MVC architecture.

The accent on evolving design before starting with the reasoning knowledge is a definite break with the *modus operandi* proposed for the KBE. The larger number of elements available for modelling and the different conceptual level for actions that have to be undertaken for modelling activities may seem daunting, but first results have shown that the divide is not as overwhelming as it seems.

The exact opposite to the effect noted with the KBE is in play here, its initial simplicity leads to hidden complexities only noted during actual use, while here the actual difficulty in choice and conceptual diversity divides a model into meaningful parts, sub-parts and interrelationships in analogous way to the experience of experts in their own domain.

The further support for change, direct feedback on that change, and the ability to revoke changes goes far beyond the short cycle provided by the executable specification nature of the model. In IO, every change is susceptible to a test as each change is ascertained to the meaning of that change to the model. This strengthens the effects of having a live executable model, capitalising on its strengths.

The increased support for abstract communication further improves the model/view distinctions and means that less effort is required to create a visualisation for a knowledge model in the form of a knowledge system. The improved support for MVC strengthens the capabilities to create complex design and consultation environments for a user.

The encapsulation and inheritance of the object-oriented models are meant also to support the evolution of the system and aid the ability to keep changing different parts of the system. This is seen as a partial solution to the gridlock experienced in changing a KBE knowledge base when it has grown beyond a certain size like 200 tables. Two hundred tables do not pose a big challenge of IO, because they are structured and subdivided throughout the model. The difference between the KBE and IO lies mainly in the approach taken to knowledge modelling, which for the KBE resembles first generation systems, and for IO is more in line with second-generations concepts.

Compared to the KBE, less is known about the problems and limitations of the knowledge modelling capabilities and knowledge system development capabilities of IO. Where systems have been developed that stretch the capacity of the KBE, such

systems are yet to be built using IO. The problems that are seen therefore concern the initial stages of development.

Feature	KBE	IO
Vivid Knowledge Representation		✓
Visual Knowledge Representation	✓	✓
Component Based Development	✓	✓
Model View Controller		✓
Abstract Communication	✓	✓
Centralised Deployment	✓	✓
Executable Specification	✓	✓
Live Model		✓
Simple Knowledge Model	✓	
Simple Inference Mechanism	✓	✓
Full Object Oriented Model		✓
Script Interface		✓
Unlimited Undo/Redo		✓
Code Insight		✓
Adaptable Visualisation	✓	✓

**Table 5-5 Comparison of Main Features of the KBE and IO**

Comparing the facilities to that of the KBE, users have posed a returning question as to the ability to create an instance without a type. The KBE is able to reason on an instance, without ever thinking about a type for such an entity. IO requires a type before ever creating an instance. This limitation can reduce the ability to discover new knowledge on the level of instances, and transform this in time to knowledge at the type level. This problems is seen to be connected to an overarching problem, namely the initially more demanding learning curve for this system, which remains to be higher than that of the KBE.

# Chapter 6 Case Studies

---

*Make the easy things easy and the hard things possible*

*– Software Engineering Maxim*

This chapter depicts a diverse set of projects that put the tools and techniques described in the previous chapters into practice. The first section provides an overview of what this chapter seeks to convey. The following five sections describe each of the case studies used to evaluate the proposed solution, in its realisation in tools and guidelines for people, project, product, and process. In the final part of this chapter the results are gathered, going towards some general conclusions. The following chapter will complete the evaluation of the case-studies by comparison to criteria of the evaluation program formulated in Chapter 3.

## 6.1 Overview

This section discusses the relationship of these case studies to the research program, the structure of the description of the case studies and the intent behind the inclusion of each of the featured systems.

### 6.1.1 Research Program

The case studies aim to show a mix of systems built using one of the two tools, KBE and IO, and explore different of the aspects of the continuous knowledge engineering approach.

- **BOKS** – exemplifies a professional, industrial-strength knowledge system developed using the KBE
- **MDDS** – represents a knowledge discovery effort supported by the KBE
- **VDES** – shows a knowledge system for a smaller user community, and illustrate the more limited effort to develop a user-interface.
- **FRAS-KBE** – illustrates a separated knowledge design model and its implementation in a software system, i.e. use of a knowledge model as a specification.
- **FRAS-IO** – demonstrates the abilities of IO in knowledge modelling by an expert in participatory and independent scenarios.
- **Mebis-KBE** – establishes the support for usage of KBE knowledge models as component technology.
- **Mebis-IO** – establishes the support for usage of IO knowledge models as component technology.

**Figure 6-1 List of Case Studies**

The case studies provide a good cross-section of the work performed at TNO in the development of knowledge system. The knowledge systems described in this chapter

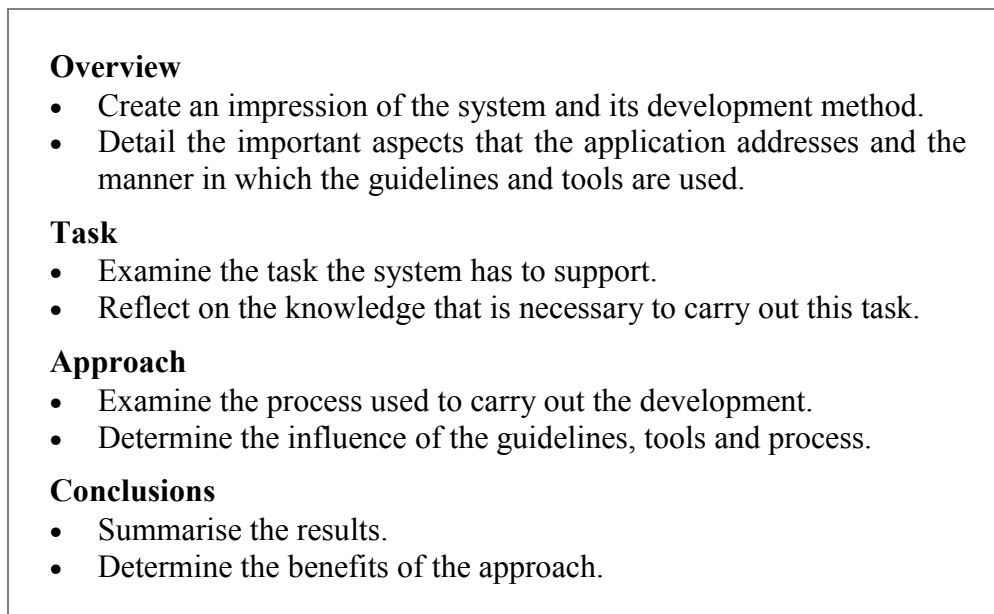
form the second part of the engineering phase of the research that demonstrates the strength of the implementation. These systems serve to enable the analysis of the implementation and the conceptual solution that was proposed. The case studies are evidence of the abilities of the proposed global solution by the implementation of this solution in both guidelines for development and the tools described in the previous two chapters. By virtue of this demonstration, it is possible to ascertain whether the solution is viable and practical, and subsequently examine possibilities to come to a qualification and quantification of the solution.

The applications are a selection from projects performed at TNO. However, these projects did not contain any special objectives to act as proof of concept for the approach proposed in this thesis. None of these systems described in this chapter was in effect a prototype or proof of concept purely dedicated to elucidate the questions originating in the research program. The cases therefore present a realistic view on the treatment of the different principles and idiosyncrasies of the tools used. They represent a good cover of the different possibilities through their similarities and differences. Because of this realistic aspect, this thesis may not assess some of the concepts exhaustively. In assessing these concepts and to compensate for this imperfection, the differences and similarities of the cases are used provide a more complete image.

### 6.1.2 Case Description

The case studies must answer the questions described in the evaluation program formulated in the first chapter. These aim to tease out the effects of the metaphoric distinction and descendant continuous knowledge engineering approach in adaptations to the people, project, product, process and tools.

The second part of the evaluation is more practical as it examines the effects on the bottom-line of knowledge engineering. These two lines of evidence are to enable the scientific and practical evaluation of the solution.



**Figure 6-2 Description Structure**

The five different systems in this chapter are quite diverse in their purpose, domain and approach. The standardised description of the important features of the application allows the comparison of their descriptions and properties in the final



section of this chapter. In addition the case-studies make explicit the manner in which they realise the guidelines and employ the development tools. The sections below describe each of these case studies according to the case description just shown. The different implementations of FRAS and Mebis will be discussed in one shared section.

## 6.2 Building Materials Regulation Knowledge System

The Building Materials Regulation Knowledge System (BOKS) incorporates the Dutch regulations on the use of building materials. It supports users in their adherence to the legislation when employing different kind of building materials. BOKS is an example of an industrial strength knowledge system with a dedicated user-interface and integration with other auxiliary software components. As a knowledge system developed with the KBE, it provides a good example of a system maintained over an extended period. Moreover, it shows experts in a participating role in the development of the knowledge model (Schilstra & Spronck 2000, Spronck & Schilstra 2000).

### 6.2.1 Overview

Environmental issues are now a common concern in all aspects of society, in many cases with the government as a coordinating body. An instance of this is the legislation concerning the use of building materials by the Dutch government in the 1995 publication of the 'bouwstoffenbesluit' (BSB).

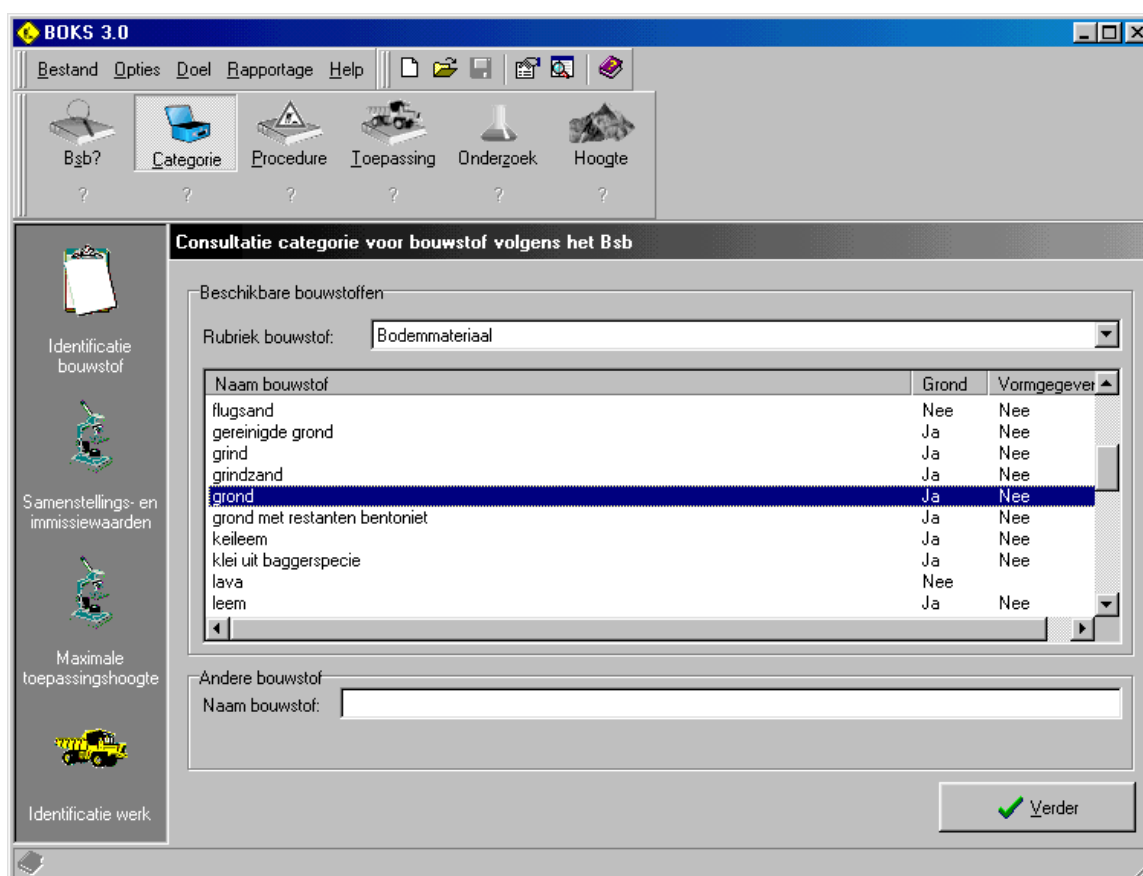


Figure 6-3 BOKS User Interface







The BSB determines whether building materials are permissible in a specific situation, what procedures to follow, and what conditions have to abide by. It actively promotes reuse of building materials, as long as this forms no threat to the health of

the environment. Since January 1, 1999, the BSB has become a set of official regulations, and every building contractor is legally bound to conform to them.

The government publication of the BSB consists of 300 pages of text that is difficult to apply not only because of its sheer size, but also by the numerous tables, references, schemas, footnotes and exceptions. All of these have to be taken into account, and the knowledge for one specific aspect can be found spread throughout the document. Practice has shown that even BSB experts find it difficult to use it correctly. One the other hand, once they have applied the BSB to a particular situation, the results are easy to understand.

A knowledge system emerged as an ideal way to allow people to check whether they are using building materials in conformance with the law without requiring training for this purpose. Approx. 3000 people in the Netherlands use it, constituting a large majority of those who need to work with the BSB.

BOKS is a stand-alone computer program that contains knowledge about the BSB, with a knowledge model developed using the KBE. BOKS can answer six fixed queries or goals, represented in the large buttons in the BOKS user-interface Figure 6-3 above:

Button	Name	Question
	BSB?	Is the BSB applicable to the current situation?
	Categorie	Which is the category the materials belong to?
	Procedure	What is the procedure to follow when the material is used?
	Toepassing	What are the requirements for application of the material?
	Onderzoek	How should the user sample and test the material?
	Hoogte	How high can the material be stacked

**Table 6-1 BOKS Goals**

The consultation of these goals offers enough information to allow a layman to know whether they conform to the BSB (see Table 6-1). Using the knowledge model the system will pose questions to the user, such as “What is the type of material”, “Where is it used” and “What are the emission values the laboratory reports”. The nature of all the questions asked by the system is such that a layman can answer them. After such a consultation it gives a report that is identical to the kind of conclusions an expert would reach. Other functions of the system include saving and loading a consultation, as well as generating different types of reports and providing access to the background documentation on the BSB itself.

The main benefit of the system is that it negates the need for people to be trained in the use of the law. This would be difficult because it concerns large numbers of people and because the law itself is still undergoing changes. The added advantage of a fast and consistent system has meant that even the experts involved in its development use BOKS in their daily practice.

The development was a mix of engineering and insight-based issues. While the project employed a standard version based deployment scheme, which has seen three different versions of the system released into the user community, the development

process was highly cyclic. The experts participated heavily in the knowledge modelling process. The development of the BOKS knowledge model has thereby seen more than a thousand iterations of different granularity. Minor tests and validation on test cases were performed with the default consultation environment.

The user-interface developed over several stages coinciding with the knowledge modelling. It was incorporated into the system for the deployment of the first version. The main product in this case was very clearly the knowledge system, as the problem to be solved here was availability of knowledge rather than the development of new knowledge. The intent behind the knowledge model was to provide a complete replica of the articles of legislation. It therefore does not constitute a true product of this project.

This case study shows the ability of the tools and approach to realise an actual knowledge system. BOKS is an industrial strength application, used by a great many people and with a certain amount of maintenance history.

## 6.2.2 Knowledge Modelling

### Task

The BSB regulations are applicable in situations where primary and secondary building materials are transported to a building site to be used there. If there is no official quality assurance of the building material, a laboratory test must analyse a specified number of samples of the material. The test determines quantities and emission values for possible environmentally harmful substances contained in the material. Based on these laboratory findings, in combination with the exact purpose the contractor has for the material, the BSB decides whether the material can be used, and if so, under which conditions.

The main part of the BSB is concerned with the determination of the category of the building material. The different categories are:

*Clean soil:* If the material falls in this category, it is usable without restrictions.

*Category 1:* Category 1 material is usable without restrictions in the situation the building contractor has specified. One of the consequences is, for instance, that if the contractor has specified that he will use the material in a layer with a specific height, he cannot add to that height without running the risk of the material falling in another category.

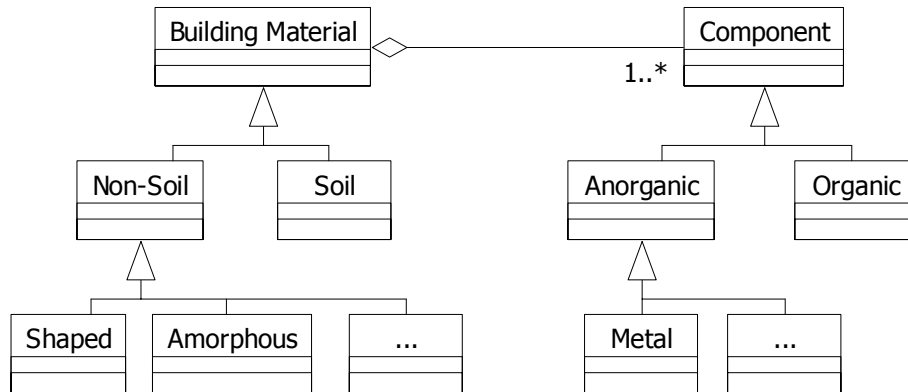
*Category 2:* Category 2 material is only usable if they are isolated from the environment, and even then, only in the particular situation the contractor specified.

*Unusable:* Most building materials are unusable at all, if they do not belong to one of the preceding categories. Only with a specific exemption can the material be used legally.

To determine the category of a material, the BSB expert just needs the laboratory findings and a description of the application situation. The answer needed from the expert is which of the six possible categories to which the material belongs. The user can understand the 'input' and output of the system quite easily. The input is the laboratory-supplied list and the situation description. The 'output', is the category of the material, and related information. The procedure to get from 'input' to 'output' is complex, however, even for experts, and even though the procedure is uniquely

determined in the regulations. After the material category has been determined, the procedures and conditions follow from this but are specific to the situation in which the material is used.

### Domain model



**Figure 6-4 Partial Material Model**

A part of the domain model used is the model of the materials, shown in the figure above. This model does not feature explicitly in the structure of the knowledge model featured as a paper-model alongside as an additional model. This model of the materials and its components is the basis of the conclusions reached on a single building material. The strongest category for any of the component materials determined the category of the building material.

### Task model

The task model for the system is relatively simple.

- Determine building material
- Determine environmental features
- Evaluate each of the component materials
- Determine procedure and conditions (opt)

Before being able to say anything about the procedures and codes, both the nature and origin of the building material and the location where it will be applied must be known. The evaluation of the each component materials is the next step, leading to the category of the building material. The next steps are optional, depending on which goal the user selected:

### Knowledge Modelling Process

The KBE was used exclusively to develop the knowledge model. Based on the functionality of the different goal questions mentioned earlier, the system development process incrementally added each of the six goals. Each goal functions as a sub-goal for other goals, therefore the development of the knowledge model in these subdivisions worked especially well.

The knowledge engineer took the role of the intermediary, who would interview and discuss the global and specific parts of the BSB articles of law. Initially this was fed back to the expert as text notes, but this practice was abandoned early on for an alternative propose-and-revise approach. This process was used almost exclusively for

the duration of the knowledge acquisition. The knowledge engineer entered knowledge into the system, and the expert reviewed the decision tables and parameters. This review allowed the knowledge engineer to explain the content of the decision tables and perform some test cases with the system. In many cases these discussions would concern a single table, but the knowledge engineer and expert performed two complete reviews of the knowledge as a whole as well.

The visualisation of the knowledge as decision-table and the simple domain model and inference mechanism allowed the expert to not only understand it enough to validate and verify the modelled knowledge, but also to comment directly on the table, giving instructions to add an alternative or change a value in one of the cells. Sometimes this could extend to the suggestion to add a new table at a certain place. The ability to consult the system at each point in time also enabled some experimentation and the use of test cases. In other cases, a single or a small set of decision-tables would come under scrutiny by consulting these few tables.

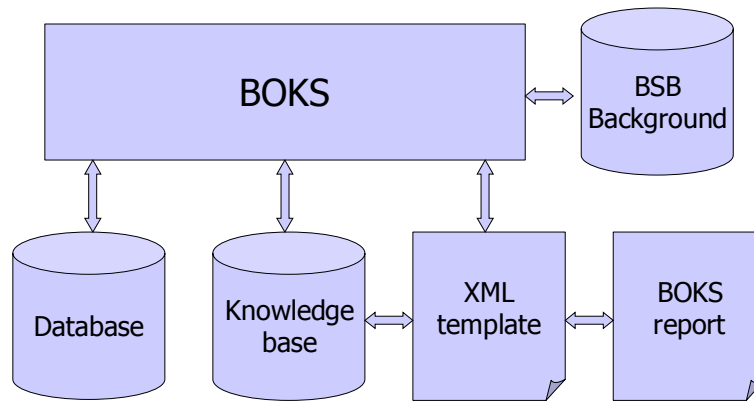
A further result was that by posing an initial formulation of a decision-table afforded an exploration of the knowledge representation. By subsequent reformulation, a decision-table could be located that would best fit the sometimes-difficult law articles. This has also led to a situation where the experts would many times faulted by their own knowledge. The consistent and complete representation of the decision table would contain a missing action alternative, indicating that the law did not cover a specific situation.

From consultation of the system in test- and real life cases, using both release candidates and deployed versions, different errors and misconceptions surfaced. The knowledge engineers have fed these back into the model. In some cases, it became obvious that these were a result of inconsistencies in the law itself. The experts have reported these to the legislative body. Furthermore, the law underwent changes in certain areas. In effect, the knowledge model grew considerably and the majority of the decision-tables in the model have seen one or more modifications during their lifetime. The development of the system will remain to be an ongoing effort as the law will undergo more change and the system's experience includes increasing numbers of real world problems.

### **6.2.3 Knowledge System Development**

The BOKS is a system that integrates different additional components. The knowledge component is only a part of the larger system. The application also contains a database of approx. 150 different building materials. This is used to answer many of the questions that would otherwise be directed to the user. In addition, it includes an integrated reporting facility. The domain experts have maintained both the report templates and the content of the database.

BOKS had to be an industrial strength application with a user-friendly interface and additional facilities. The system should be easy to maintain, to both reduce cost and reduce the time needed to update the knowledge in the system. The user should have access to the latest version of the regulations as quickly as possible. This extended to both the knowledge base and the reporting facilities. It was determined beforehand that the knowledge engineer would develop the knowledge base in participation with expert, using a propose-and-revise approach, described above. The developers went through many iterations although the number of versions released were quite limited in number.



**Figure 6-5 BOKS Architecture**

The architecture of BOKS is more involved than the more basic systems show later. BOKS integrates a number of elements not found in other systems. First, the database that contains pre-defined answers to questions on building materials and components that may arise from the knowledge base, built with Microsoft Access. The documents that are integrated include a HTML based version of the BSB, built with Macromedia Dreamweaver. The report format of the KBE was the basis of the reporting facilities. Adept ArborText was the tool for editing the XML file. Borland Delphi, a standard software development environment, was the integrator of these components, and was the basis of its visual user-interface.

It took approx. 1000 man-hours to develop the first version of BOKS. This divides into 40% knowledge acquisition, 40% application development, and 20% residual issues.

### User-interface

The user interface incorporated into BOKS is quite user-friendly (see Figure 6-3). Because it asks simple questions and provides an intuitive visualisation, people often suppose BOKS is no more than a friendly kind of spreadsheet. While this is in a way the intended effect, this obscures the fact that it contains an array of knowledge, which it applies diligently to come to its conclusion.

The system's user base is quite sizable and diverse in background, therefore a common denominator had to be found. The capabilities for the development of a dedicated visualisation was therefore essential. The importance of the user interface in the acceptance of the system can not be overestimated. A further important issue in this respect is the completeness of functionality supported by background information etc., as well as having a simple workflow. The number of questions asked of a user during a consultation is one particular part of this. The repetition in the questions asked at the beginning can be detrimental in the long run.

### Database

BOKS also incorporates a link to a database that contains information on most building materials, such as their composition into different chemical compounds, such as cadmium, etc. This database contains 150 different pre-defined building materials. The user can also enter an unknown material, but then will have to answer many different questions on that new material. This will add the new materials to the database. This is especially aimed at reducing the knowledge required to operate the system, and aims to lessen the total number of questions asked. The relevant

properties of a material as fixed per material type, but can not be considered common knowledge. Therefore its inclusion in the form of a database enables a much wider user base, as well support the ease of use.

## Report

The reporting facilities are also quite extensive, the user has the option of choosing different types of report: concise, normal and extended. This allows for quick review and full documentation for reference purposes.



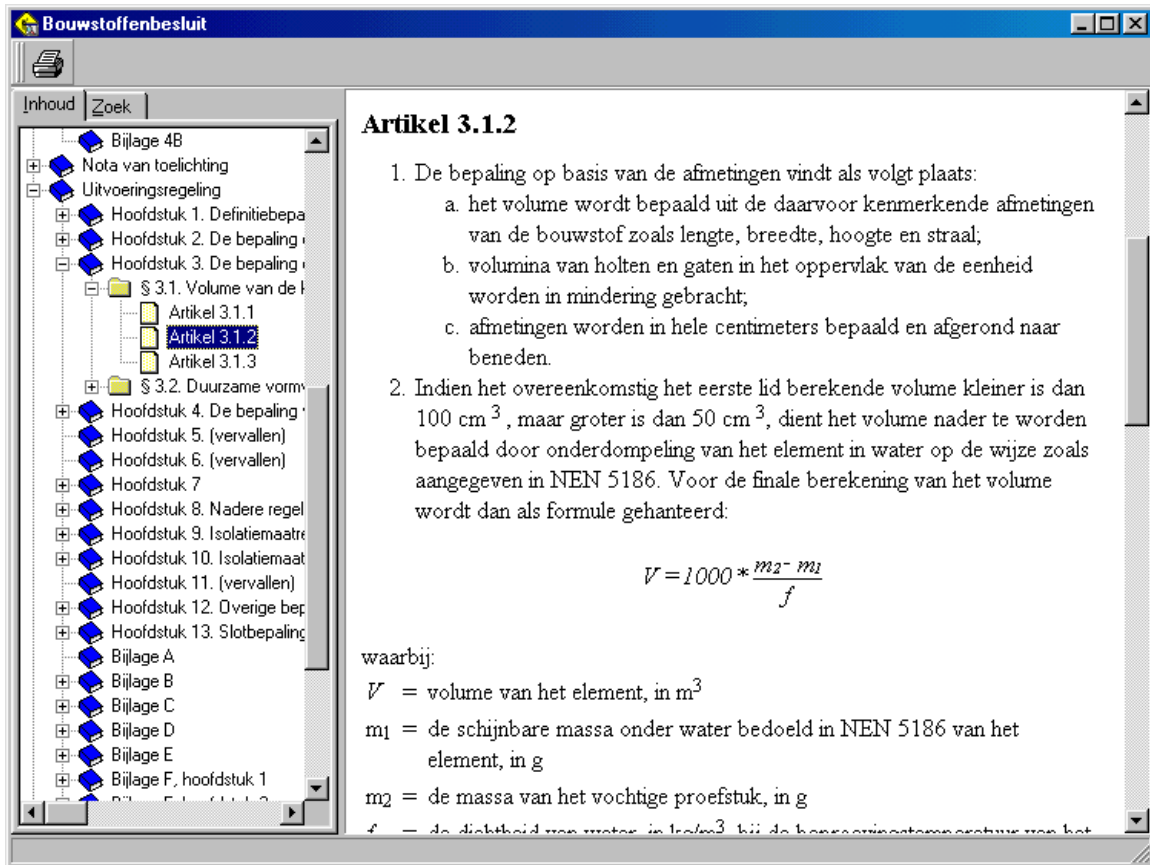
Figure 6-6 BOKS Report

The image above shows an example of one of the reports. The reports use the XML reporting facilities described in chapter 4. The experts can edit the template reports themselves, as part of the development effort.

The reports can be shown in either concise or extended form. The concise report contains only the relevant final conclusions, i.e. the class the material was assigned and other answers to the goal questions. The extended report details the answers given and provides intermediate conclusions.

## Documentation

BOKS further integrates the full text of the BSB for reference in HTML form. This enables an author of a report to integrate hyperlinks to this background info. All the pages of the BSB can be found there. This reference thereby provides all the relevant background to a case. From sections of the report, the user can then request the relevant background information.



**Figure 6-7 BOKS Documentation**

An example is shown in Figure 6-7 above. This form of the text of the BSB is also fully searchable.

## 6.2.4 Conclusions

The previous has shown an application of a KBE knowledge model developed into a knowledge system that supports a great number of users. It is the most advanced example of the ability to develop professional, industrial strength knowledge systems using both the continuous knowledge engineering approach and the tools. Furthermore it is an example of a legal knowledge system, of which there are apparently few successful examples (Visser, Bench-Capon & van den Herik 1997).

<i>People:</i>	Participatory modelling	<i>Cost:</i>	-
<i>Project:</i>	Finite, product-based	<i>Risk:</i>	-
<i>Product:</i>	Dedicated knowledge system	<i>Clarity:</i>	++
<i>Process:</i>	Cyclic modelling and development, with limited deployed versions	<i>Benefits:</i>	++
<i>Tools:</i>	KBE	<i>Bottom-line:</i>	++

**Table 6-2 Evaluation BOKS**

This case study shows quite clearly the significance that the expert participation. Feedback from application of the system had a definite effect on the knowledge model and knowledge system. Being able to add small increments of knowledge to an operational knowledge system is critical in this respect, because it creates a feedback loop. The gradual approach to knowledge modelling also makes it easier to develop the knowledge model and aids the communication between the knowledge engineer



and the expert. From his understanding, the expert can make constructive comments directly on the knowledge represented in the system.

As to the modelling ability of the expert, the conclusion was that the experts would be able to develop parts of the knowledge model independently. The granularity of such modelling changes would have to be a small, for example a single decision table. Lack of structure in the domain model as a set of parameters makes it difficult to form a conception of the organisation and interconnection in the model. This can make it difficult to locate the parts of the model that to be modified or understand the effects of a change to other parts of the model. Some separate experiments confirm this result. The gradual approach also supports the development of the knowledge system, as there were more opportunities early on to give attention to feedback from the users, on the content and the form of the knowledge system. Because a usable system was available early on, it was easier to elicit such comments, and direct the development by it.

Furthermore, the cost of development as well as the risk associated with it was lower. This was mainly due to the shortening of path to change. As the criticisms of the experts were directly based on the knowledge model, and did not have to be elicited through circumstantial evidence. This lowers the cost of both the acquisition and the translation into the knowledge model. The cost of measures required to incorporate changes in the knowledge model were of low enough to allow alternatives to be realised in the model and tested, before settling on a specific one. The large number of small cycles is evidence of this. Coupled to the clear idea of the resulting product and the benefits the introduction of the system would create, the decision on additional investments in the knowledge system were based on actual value rather than predictions.

It further shows how from an initial default implementation of a knowledge model, an iterative strategy can explore the knowledge, by experimentation and evolution. This improves not only the knowledge model, but also leads to significant insights for the domain expert and can improve the domain knowledge.

## **6.3 Masonry Damage Diagnostic System**

The Masonry Damage Diagnostic System (MDDS) is a knowledge system for diagnosis of damage to old buildings and monuments, to determine the cause of damage and provide a starting point for restoration. This is an example of an incremental development process and participation of the user, but one where the key product is not a knowledge system, but the knowledge model itself.

### **6.3.1 Overview**

Conserving ancient building and monuments for posterity warrants a significant amount of effort, to keep a view of the past otherwise only found in history books and old paintings. Our bustling post-industrial society has placed many of these buildings under further stress, such as acid rain, and wear-and-tear from vibration.

Conservation of historical buildings is made harder than it could be because the knowledge of it is distributed over a small group of specialists from different disciplines: building physics, architecture, history of art, chemistry, geology and biology. They lack a common language and their current mastery is incomplete.

A knowledge system does not merely improve the availability and accessibility of knowledge of ancient brick masonry structures. It can also provide a catalyst for detecting lacunas in the knowledge of the experts, and aid the formulation and discovery of new knowledge. Consolidating the knowledge and integrating the different fields into a set of shared assumptions and a common vocabulary makes interdisciplinary communication a possibility. Such communication is a first requirement for these experts to start sharing their knowledge and experience. The task for the knowledge system is therefore much broader than merely enabling users to diagnose masonry damage.

To this end MDDS, was developed both in the interest of the science of restoration and for practical restoration activities. MDDS originates in the EC Environment-project entitled 'Expert System for Evaluation of Deterioration of Ancient Brick Masonry Structures', dating from 1992 to 1995. The development of the system continues until today. Plans for enhancements and further extension are actively pursued. Over a period of 8 years approximately 1500 hours of work over the total period spent on the actual knowledge model. The knowledge creation and discovery component in the development of this system's knowledge model is therefore clearly one of the main goals, next to the usage of the system.

The intended users of MDDS are quite diverse:

- professionals in charge of the maintenance of historic brick masonry monuments;
- specialist in other fields, occasionally confronted with the specific problems of historic brick masonry structures;
- persons responsible for restoration campaigns, such as architects;
- students, following courses on conservation and maintenance of buildings;
- representatives of national bodies in charge of the conservation of the cultural heritage.

The development of MDDS is a prime example of continuous development, and is an instance of an insight-inspired approach. The fact that the knowledge model in MDDS was developed by the experts themselves was an important aspect in making that possible. Mainly because of that, the project is continuous in nature and as yet part of ongoing research and development. The main product is primarily the knowledge model itself; the knowledge system plays an important but secondary role. The knowledge model deployed using the default consultation environment and using an Internet-based consultation system.

### **6.3.2 Knowledge Modelling**

#### **Task**

Damage to masonry is due to many different chemical and physical processes. Exposure to caustic materials, like acid rain or damp conditions, is an important recent cause of problems. Structural problems can also cause deterioration of the masonry. These different problems can lead to an array of cracks, deformations, fungal infestations and other problems. The diagnosis of the causes of these problems is the first step in being able to counter-act and repair the damages and to prevent the problems from reoccurring.

The problem that the expert faces during diagnosis is to locate the damaging process. Visual observations and background information on the structure play an important

role in this. They provide an impression of the kind of damage, which is an entry point to determining the actual cause of these symptoms.

The knowledge system uses the same kind of diagnostic process and information utilised by an actual expert. The focus taken is that the information provided by the user must be observable visually or easy to determine otherwise. What's more, the knowledge in the system, coming from so many different sources, should represent a consensus, which meant not only reaching some accepted norm for the system, but also scientific standardisation, as the conclusion of different scientific research projects was reported to the community as decision-tables coming from MDDS (e.g. van Hees, Pel & Lubelli 2000).

### Domain Model

There are a great many different manifestations of damage to masonry, each with its own specific characteristics. It is possible for the user to see more than one type of damage, but the system is only configured to deal with one type of damage at a time. This is due to the limitations of the KBE that cannot adequately represent those kinds of multi-causal diagnoses. Consulting the system the user should then normally select the most representative form of damage seen.

Type	Sub-type	Specific damage type
surface change		
	discoloration	fading, moist spots, staining
	deposit	soiling, lichens or liverworts, algae, mosses, moulds, unknown biological deposit, graffiti, encrustation, efflorescence, crypto-florescence, undefined
	transformation	patina, crust
disintegration		
	layering	delamination, exfoliation, spalling, scaling
	loss of adhesion	loss of bond, blistering, peeling, peeling hide, push out
	cratering	cratering
	loss of cohesion	chalking, powdering, sanding, brick-blistering, erosion, crumbling, pulverization, bursting, void
cracking		rupture, hair crack, crazing, crack, star crack, network cracking, vertical cracks
deformation		bulking, bending, twisting, leaning, displacement, bulging
mechanical damage		scratch, cut/incision, puncture, splitting, chipping, mechanical damage (general)
biological growth		
	higher plants	higher plants
	living exogenous material	lichens or liverworts, algae, mosses, moulds, unknown

**Table 6-3 MDDS Damage Type Decomposition**

The table above shows the hierarchy of damage types and sub-types. There are 53 individual types of damage, which a Damage Atlas defines through textual and

graphical information. It provides different translations for each of them, to disentangle their use in different contexts and discussions. Although in majority the hierarchy is tree-like, the type 'biological growth' and the sub-type 'deposit' share some of their types.

- frost damaging process
- salt crystallisation process
- environmental pollution chemical process
- surface erosion process
- water penetration process
- mechanical damaging process
- surface deposition process
- condensation process
- structural damaging process
- iron corrosion process
- biological process

**Figure 6-8 Damaging processes**

The damage types are manifestations of one or more of a set of possible damaging processes. The table above shows the complete list of these processes incorporated into MDDS. This shows the chemical, biological and structural forces playing on a masonry structure leading to degradation and destruction of the stone works. The damaging processes in their main categories are 11 in total; a few of them have sub-divisions into more specific process determinations.

### Task Model

	Masonry Damage Diagnostic Sy...	R1	R2	R3	R4
C1	any damage	yes			no
C2	determination of type of damage	performed		not performed	-
C3	determination of damaging process	performed	not performed	-	-
A1	diagnosis for masonry damage	performed	not performed	not performed	not performed
A2	COMMAND display(remarks, 'This consultation has been completed. You can perform further analyses on the subject by using the what-if functionality.')	×			

**Figure 6-9 MDDS Top Table**

The basic structure of the task is to determine the damage category, and use this category to test different associated damage processes as hypothesis. The system enforces this structure by using the decision-tables to derive sub-goals. These order the steps taken in the evaluation, where each condition is concerned with a specific sub-task.

The system starts by asking the user the general type of damage, making it possible to dismiss many of the non-suitable categories of damage offhand. If the user leaves the

question unanswered, the system will ask questions for each of damage categories, until the right type of damage is determined through analysis. By querying for specific symptoms, it is able to perform this classification task. This provides for use of the system by novices and experts.

By determining the type of damage, it is now possible to start the determination of the damage process or processes. The system treats each of the processes as a hypothesis of a process that may be the cause of the damage. Because only certain damage types conform to a specific damage process, they filter the evaluation of the damage processes.

	determination of damaging ...	R1
C1		-
A1	<diagnosis>	{frost damaging process}
A2	<diagnosis>	{salt crystallization process}
A3	<diagnosis>	{environmental pollution chemical process}
A4	<diagnosis>	{surface erosion process}
A5	<diagnosis>	{water penetration process}
A6	<diagnosis>	{mechanical damaging process}
A7	<diagnosis>	{surface deposition without chemical process}
A8	<diagnosis>	{condensation process}
A9	<diagnosis>	{structural damaging process}
A10	<diagnosis>	{iron corrosion process}
A11	<diagnosis>	{biological process}
A12	determination of damaging process	performed

Figure 6-10 Determination of Damaging Process

For the determination of possible causing processes, a central decision-table is present in the knowledge model for each of the processes. These decision-tables contain knowledge about the possibility that the damage process is relevant and possible. The decision table 'determination of damaging process' evaluates all of eleven processes in order, and as a side effect infers a parameter denoting the conclusion for the process. This may lead to the conclusion that more than one process is the cause of the damage. An example of such a table is below for the frost damaging process.

frost damaging process		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11
C1	relevant type of damage for frost damaging process	yes										no
C2	typical frost damaging winters may occur in this geographic z...	yes									no	-
C3	can damage be related to a typical frost damaging winter	yes				unknown				no	-	-
C4	high moisture content in material(s) possible	yes			no	yes			no	-	-	-
C5	material vulnerable to frost	yes	unknown	no	-	yes	unknown	no	-	-	-	-
A1	frost damaging process	possible	possible	impossible	impossible	possible	possible	impossible	impossible	impossible	impossible	impossible
A2	COMMAND display(diagnosis,['Damage may be caused by frost action.'])	×				×						
A3	COMMAND display(diagnosis,['Damage may be caused by frost action; to confirm hypothesis perform proper frost test.'])		×				×					

Figure 6-11 Decision-table ‘Frost Damaging Process’

The form and content of this decision table for this particular process is characteristic of all the central decision-tables for damage processes. First, a damage process must be relevant, i.e. fit the damage type. For example, frost damaging is only relevant as a damage process for 16 out of 53 damage types (see the decision table below). When it is relevant, the system considers the environmental and material related situation in which the problem occurred. A damaging process is a cause of damage only when it meets favourable conditions. Frost damaging only occurs when winters occur in the region, it is possible that the material used had high moisture content, and the material is vulnerable to frost.

The screenshot shows a 'Table Editor' window with a toolbar containing icons for help, undo, redo, and other editing functions. The table below is displayed within the editor.

	relevant type of damage...	R1	R2	R3	R4
C1	type of damage	delamination OR exfoliation OR spalling OR scaling OR sanding OR crumbling OR chipping OR loss of bond OR push out OR peeling OR cratering OR bending OR bulging OR displacement	crack OR hair crack		ELSE
C2	further description of damage-crack	-	crack(s) in individual bricks or short cracks in pointing OR parallel cracks in laying mortar ...	longer crack(s) affecting several layers of brick (+pointing)	-
A1	relevant type of damage for frost damaging process	yes	yes	no	no
A2	COMMAND	display(process, 'frost damaging process')	display(process, 'frost damaging process')		

At the bottom right of the editor window, there are 'OK' and 'Cancel' buttons.

Figure 6-12 Decision-table ‘Relevant Type of Damage for Frost Damaging Process’

A single parameter ‘relevant type of damage for {processname}’ makes the link between one of the damage types and the process. If the damage type is in one of the categories that may be relevant, it is a valid hypothesis. As is visible in the decision-table above, the link is sometimes more than merely a mapping between type of damage and processes. For the damage type ‘crack’ or ‘hair crack’, the table

contains a special situation, where frost is not always a possible hypothesis. After the relevance of the process is clear, the actual testing of the hypothesis is started.

The remainder of the evaluation of the process in majority examines whether favourable conditions occur, for example, the geographical zone should have frost damaging winters, high moisture content of the material should be a possibility and the material should be vulnerable to frost.

### **Process**

At the start of the project, the knowledge engineers developed an initial model for the diagnosis of the masonry damage, based on interviews and discussions with the experts (Lucardie 1995, pg. 199-236). The experts rejected the model from this initial effort, not because it did not function correctly. It did not sufficiently reflect the way an expert operated in real-life or was easy to understand as such a model by the expert. They considered it to be 'too detailed', as it goes into the chemical processes at work in the brick masonry. The level wanted by the experts is more abstract as is seen in the current knowledge model.

The experts commenced to develop a new knowledge model independently, only occasionally supported by a knowledge engineer. A small team of ambassador-experts represented the full team of experts that were involved. These ambassador-experts were located at TNO in the Netherlands, and were the people responsible for making the actual changes. The other experts involved themselves with reviews of the knowledge model and testing versions the knowledge system. From their discussions on the content of the knowledge model, they would propose adaptations changes to the tables or definitions. The ambassador-experts at TNO reviewed them and incorporated the changes into the model. Only the ambassador users make these changes, and therefore constitute a mediated consensus process coupled with a review-critique approach.

Two other activities started in parallel with the development of the knowledge model, partially in support of the knowledge modelling. First was the development of a list of definitions for the damage types and processes in the model, with translations to different languages. Another effort was the development of a damage atlas, where each damage type featured using a visual example, to aid the user in determining the damage type. The definitions and visual catalogue form an integral part of the knowledge model.

The abstract task model, determination of the damage type followed by a process-based diagnosis as described above, was already present in the system at that time. Initially these changed concentrated on the abstract structure, development and definition of the damage types and processes. Once the structure for the system was more or less complete, the focus turned to the determination methods for the damage types and processes. These discussions between ambassador-experts and the other members were mostly on paper. The reports, describing the knowledge model, made extensive use of the printouts from the system. In addition, meetings within the project concentrated on demonstrations of the system to elicit comments and requests.

When the project got further along, regular distributions of the system distributed regularly using the default consultation system to allow all concerned to get to grips with the dynamic behaviour of the system. This did not allow them to examine or change the knowledge in the system.

One aspect that noticed in this period was that the first version of a decision table that conceived was invariably flawed. The first version of a chunk of knowledge represented as a table would create an initial sketch. After the integration into the system, inconsistencies in the decision table would surface. Positioning of a decision-table relative to other tables reveals requirements to the table not thought of before. A further source of adaptations comes from anomalies that become evident in the behaviour of the system. This would make clear that the decision table should be organised differently or that certain aspects were missing, for example, additional conditions required in the determination of the answer. The initial thoughts on how the reasoning would be organised would justify answer found in the action alternatives, rather than aim for a complete story. This was no flaw of the experts modelling independently; it is though that this is a fundamental feature of knowledge acquisition (see also Compton et al. 1992).

Another observation during modelling is that a certain table would undergo changes and slowly start to resemble the 'form' of other tables. Eventually the experts picked this up and started to create these types of decision table explicitly. Since then, they are growing in number and importance. These decision-tables make the system easier to understand, as certain classes of decision-tables are easily recognisable. For the experts this made modelling much easier, by this self-imposed additional structure.

A problem that is negatively affecting the system now is its lack of changeability. Time has shown certain choices to be wrong, as new insights have revealed the appropriate subdivisions between the different damage types. The interconnected nature of the system makes it hard to remove elements, making the cost of a change too great. On the other hand adding a new damage type or process is quite easy because of the chosen abstract task-model. This model allowed incremental development of the system.

In the current phase of the project, with the content of the knowledge system fleshed out and in an acceptable form for the partners, the focus changed from building the system, to updating it for new insights. This was also a starting point for use of the system to direct some of the research activities. One of the areas in which knowledge was lacking, was the effect that the application of repair mortars could have on the masonry in monuments. Many cases of failure showed that the knowledge of the compatibility of repair mortars for historic masonry was lacking (Hees, Pel & Lubelli 2000). Repair mortars used for *repointing* may have been responsible for making the masonry vulnerable for frost damage, by interaction with the existing substrate.

The use of decision tables and the inclusion of a preliminary version of the knowledge into the system supported the formulation of two testable hypotheses:

- The repointing mortar may have changed the drying behaviour of the wall
- The original lime mortar may have changed its properties over the years and has become prone to frost attack.

Confirmation of the first hypothesis arrived after an advanced NMR system, with an especially high spatial resolution. This showed how repointing mortar affects different properties of the existing material, leading to high moisture content. The second hypothesis is currently under investigation. This shows an example of how an explicit knowledge model aids in locating and describing lacunas in the knowledge and how this can direct these investigations.



### 6.3.3 Knowledge System Development

This section described the development of two knowledge systems based on the knowledge model.

#### **MDDS Demonstrator**

The first system constructed based on the knowledge model described above is the MDDS Demonstrator. The users received several versions of the demonstrator, for the most part after a significant amount of change to the knowledge model. The users were limited to the partners and experts involved in the development of the system. They used the system to verify and validate the system in practical situations. This form of the system replaced the paper-based discussions of the knowledge model.

The complete activities concerning knowledge system development and deployment were under the control of the experts as well. The default consultation system used did not require any additional visualisation. The only auxiliary issue was the damage atlas, a collection of pictures of damage types. These are used as additional information for the questions for damage type determination. The system handled display of pictures quite well as an advanced option. As this default system was already available, this constitutes no development effort in itself.

In addition, the experts could perform deployment activities independently. The deployment of the system as a set of diskettes requires limited effort. Even so, it formed an obstacle, leading to postponement of deployment until sufficient changes to the knowledge model warranted it.

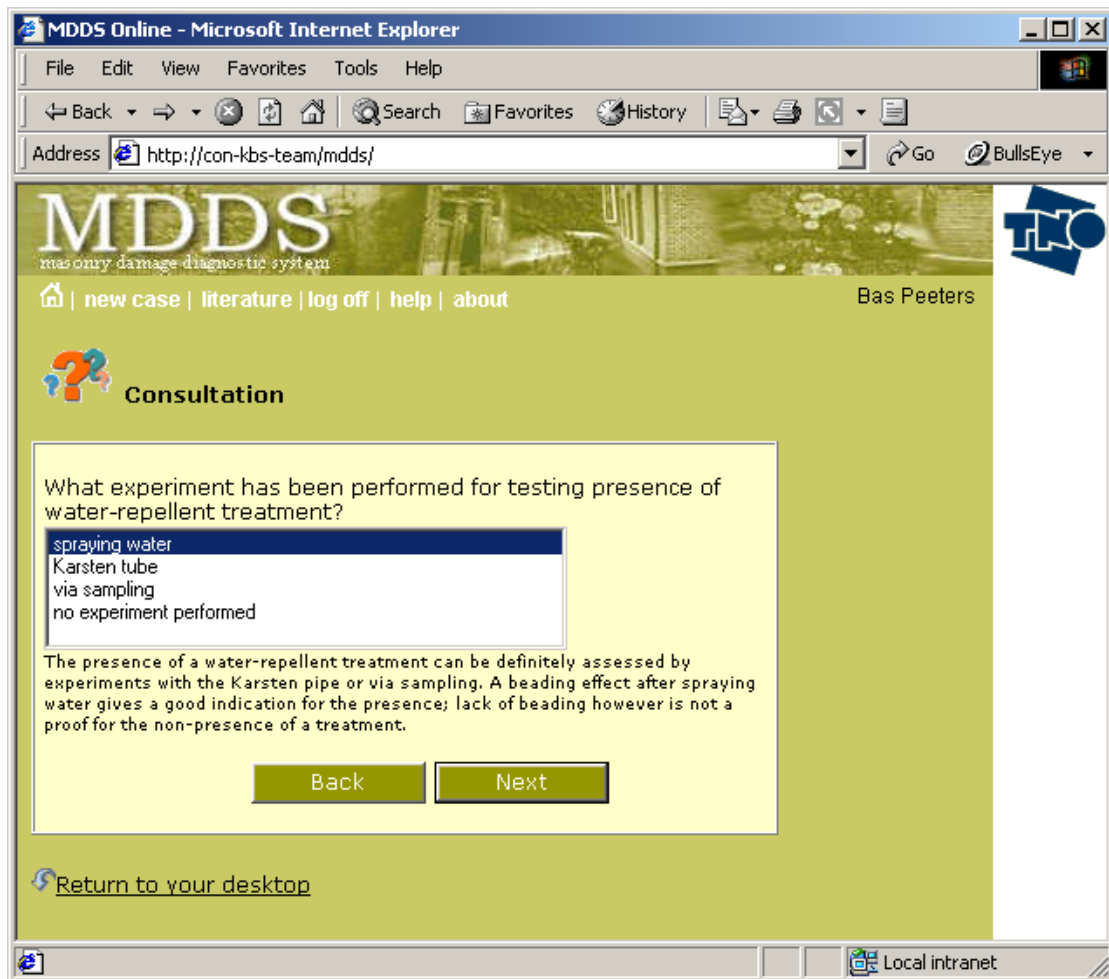
Feedback from the visual review of the paper-based versions of the knowledge model initially dominated as a source of new knowledge, and localisation of flaws. After the knowledge system was available for use, it turned into the main source of knowledge discovery.

#### **MDDS Online**

The development of the MDDS knowledge system started in earnest after the Internet technology for consultation of knowledge systems over web was completed. The knowledge engineer initially performed the development of the user interface. This was only a limited effort due to the support present in the system for customisation. After that, they became the responsibility of the experts.

The reporting facilities used to create an elaborate diagnostic report required considerably more effort, based on the XML-report presented earlier. The current report gives an explicit account for the consultation. The report has been prepared based on initial example texts from the expert. The knowledge engineer developed a subset of the reports, and transferred to the expert responsibility. The experts developed the majority of the report after that, with limited support from the knowledge engineer.

Initial comments from the expert criticised the technical nature of the approach, but with some support for questions and suggestions on best practices, the experts now also have this part under their own control. Modifications to the report are still necessary, and the experts perform them independently.



**Figure 6-13 MDDS Internet Screen**

Deployment on the Internet server means the replacement of a limited number of files on the system. Even though the effort for the demonstrator system was already quite limited, this lessens the threshold for deployment quite considerably and makes the deployment cycle very short. It negates the need to make a sufficient number of changes before warranting the re-deployment the system.

### 6.3.4 Conclusions

The MDDS system has shown its worth as a centralized clearinghouse for the knowledge on masonry degradation and damage processes. In this sense, it has been very successful in representing an evolving consensus model. From the point of view of application of the system, the results are harder to formulate as the system is not yet introduced into a large-scale user community. As a system used to diagnose actual damages, it has been successful already, as the experts put the system to the test on many actual problem cases, to test the knowledge model and validate the knowledge contained therein.

This case study shows a knowledge system that has an extensive knowledge base, independently maintained and extended by the expert. It further shows two completed knowledge systems, one using the default consultation system and another Internet-based system, which includes a customised HTML-based user interface. The experts are now in complete control of the further development and enhancement of the

knowledge model and the knowledge system supported by a centralised deployment scheme.

Some problems appear during the development of the system, which are the result of the representational poverty and the incomplete support for evolutionary approaches. Overall, it shows the KBE to be a good tool from the perspective of continuous knowledge engineering in its empowerment of the non-specialist to build user-friendly systems with assistance of a knowledge engineer.

The advantages of the approach used here are similar to those stated for collective memories (Steels 1992, Steels 1986). The development of the system creates a common terminology and guarantees continuity, as the knowledge is no longer dependent on a single person. It eases and improves communication between experts, exchanging knowledge and experiences, and through shared definitions and assumptions, speed knowledge transfer.

<i>People:</i>	Independent modelling and participatory development	<i>Cost:</i>	--
<i>Project:</i>	Continuous, process-based	<i>Risk:</i>	--
<i>Product:</i>	Knowledge model as scientific theory. Default and customised knowledge systems	<i>Clarity:</i>	++
<i>Process:</i>	Cyclic with many iterations, coupled with extensive verification, validation and testing	<i>Benefits:</i>	+
<i>Tools:</i>	KBE	<i>Bottom-line:</i>	++

**Table 6-4 Evaluation MDDS**

MDDS shows how the experts can develop a knowledge model and how this improves the quality of the knowledge, as well as support the further development of the knowledge. The MDDS system is an example of a knowledge engineering project, that clearly falls under the heading of continuous knowledge engineering. The incremental development employed there and the independent modelling by the expert, especially its explicit use in knowledge discovery. The only problem is the lack of user feedback, which has negated the ability to learn knowledge from the insights developed through application of the knowledge system. MDDS is also evidence of a project where the knowledge acquisition bottleneck seems to have disappeared and one where the cost of deploying the knowledge system anew is negligible.

## 6.4 Moisture Damage Diagnostic System

VDES is a diagnostic knowledge system for problems related to moisture in dwellings and other types of inhabited buildings. The acronym is an abbreviation of its Dutch name 'Vocht Diagnose Expert Systeem'. The knowledge engineer, the expert and an ambassador user have developed it cooperatively. The knowledge system uses a stand-alone consultation system, and an Internet server application.

### 6.4.1 Overview

Moisture damage is one of the most frequent problems surfacing in many old and new buildings. The rent tribunals in the Netherlands receive approx. 45.000 notices of objection to rent increases per year, and on a third of these, problems with damp and moisture are forwarded as reasons why the increase should not be allowed. These notices of objection form the proverbial tip of the iceberg. The front offices receive

massive amounts of complaints, many of them concerning moisture related problems. They resolve most of the complaints. The front office forwards irresolvable cases to the back-office and the rent tribunal is involved if they cannot be resolved even there.

Besides the unpleasant sight and smell, and potential damage to the dwelling, moisture related problems have a definite and clear effect on the health of the inhabitants. The determination of the cause of the moisture related problem could also have serious financial consequences as it contains in it the attribution of responsibility for repair. Diagnosis of such problems is not a straightforward affair, and experts in the field, including those at TNO, regularly provide evidence in court-cases to give an independent professional view on these matters. The number of experts with sufficient knowledge to make such diagnoses on principled grounds is small. The domain constitutes an area of ongoing scientific discovery.

The problem addressed by the system is that of diagnosing moisture related damage and fungicidal infestations that can arise from this. This knowledge is unavailable to those confronted with these problem situations. A knowledge system could help resolve many of the problems. The potential user-base of a knowledge system for diagnosis of moisture related problems is therefore quite extensive and could even extend to the full community of tenants. This may prove to be an even greater user community than that of the housing associations.

The VDES system has gone through some initial user testing. A number of housing associations are assessing the system for evaluation. The system is primarily a means of communicating the knowledge to the users. In some cases, it is not possible to make a diagnosis without complete information on the properties of the construction materials and configuration. Like MDDS, most damage is categorized into specific damage types and attributable to one or more damage processes.

The development of VDES is a prime example of continuous development, and influenced by the insight metaphor. Both the expert and user have participated in the knowledge modelling throughout the project. The project is continuous in nature although not in the same sense as MDDS. Providing a knowledge system to communicate the knowledge in usable form is the main objective. This is not to say, that the knowledge model itself is not important. The quality of the knowledge model was a great concern and the expert sees it as an important instrument for the improvement of the scientific knowledge of moisture damage diagnosis. Deployment of the knowledge system used the default consultation environment and an Internet-based consultation system.

As a case study, the example given by VDES provides a good contrast with the other systems. It illustrates the ability of experts to model knowledge independently and with a high degree of quality. Furthermore, it shows that the development of a knowledge model can stand on its own to provide the benefits required.

## 6.4.2 Knowledge Modelling

### Task

The task that lies before the expert in any damage situation is to locate the cause of the problem. In the case of moisture problems, there are a number of possible causes, from bad ventilation to broken pipes, even leakage from adjacent properties. Any of these can surface in different ways, simple by running water, or by the formation of cracks or fungal infestations.

An expert in diagnosis of moisture problems uses basic observations to determine the type of problem. This restricts the possible damage processes that can be causing the problem. Then the expert examines each of the possible process hypotheses to ascertain whether they could apply, by examining the specifics of the problem surroundings and by looking at the construction itself. The damage type uniquely identifies a number of damage processes. This specific context precludes the re-use of determination methods over different damage types.

The knowledge that experts possess for the most part derives directly from their experience and the theoretical grounding from their education. The analysis of these problems requires knowledge of building physics of heat distribution and ventilation but also chemistry and biology. It is not surprising that the field is still developing new knowledge, with new insights gained and new experiences with changes in housing patterns. Some of these advances are relatively recent and have been incorporated into the knowledge system even before they were published (Adan 1994).

### Domain Model

The distinction in the domain is between damage types and damage processes. The domain is therefore quite similar to MDDS. There are four main categories of damage, each with possible sub-categories. These are show in the table below:

Main type	Sub-type
Visible water	Running water, Condensation, Drops from seams and joints, Puddle.
Dry discoloration	Dry rings, Moulds, Salt bleeding, Sallow, Soiling.
Moist discoloration	Damp spot and mould, Damp spot and wallpaper letting go, Damp spot and plaster letting go, Damp spot and paint losing adhesion.
Structural Change	Wallpaper letting go, Paint losing adhesion, Plaster letting go, Wood rot, Floor sagging, Floor succumbing, Roof succumbing, Bulging floor, wall or roof.

**Table 6-5 VDES Damage Type Decomposition**

Where MDDS proceeds to determine which of a set of processes was responsible for the damage in an attempt to disqualify the process as a possible candidate, in VDES this initial classification is used to very specifically use a determination method, singularly to determine the cause. Most of the time, this then continues as a determination of that type of damage in a class of situations, e.g. condensation...

- On building construction,
- On building construction behind, beneath of above furniture or behind curtains,
- On windowsills, doors and windows behind curtains
- As caused by a specific climate-class

There is a number of process hypotheses for each of these subtypes, rather than a series of processes for all types of damage, as was previously seen in MDDS. The system uses the combination of damage type and situation as the starting point for the determination of the process or processes that are causing the damage. The set of possible processes is different for each combination, although some overlap and reuse of decision-tables exists.

### **Task Model**

The knowledge model contains a task model that initially comes to a classification of the problem situation. The system then investigates a number of hypotheses, possibly leading to the evaluation of further sub-hypotheses processed in a similar way. One can plainly see that the development process has a great influence on the development of the system and that it possesses a strategy for evolution, based on an adding new structure tables to the control knowledge.

Beyond the hypotheses and sub-hypotheses, there are the testing methods. They determine whether the current hypothesis is possibly the cause of the problem. In some cases, only a partial answer is possible. This also leaves the door open for attributing the problem to one of several damage processes.

The way that damage type and determination of the damage process influence each other is reminiscent of the context discussion that was also seen in the GARVAN ES-1 system, where new additions were made in the context of the last fired rule (Compton & Jansen 1990). Although this leads to redundancy in the knowledge model, it does allow very fine-tuned responses, quite specific to the damage type found and other properties of the problem situation. Some reuse is evident within the knowledge model, mainly in those cases where a cause is related to all kinds of moisture damage, such as leaks in plumbing and precipitation.

One of the problems that surfaced during the modelling of the knowledge has to do with the instance problem. When discussing a problem in one of the rooms in a dwelling the manifestation may be the result of a problem originating in one of the surrounding rooms, or even in a neighbouring dwelling. For example, a bedroom may develop colour spots on the walls because of a bathroom with a shower in it that does not ventilate well. The diagnosis of a single room is not possible, without taking into account its surrounding rooms, essence diagnosing them as well. The KBE does support representing this recursive style of reasoning well, as the configuration of rooms is not fixed.

### **Process**

The knowledge model was developed using a cyclic development style, with a team of developers consisting of a knowledge engineer, an expert and between three and four users. This team cooperated over the total period of time that the system was developed. In the first sessions, the main objective was to create a common vocabulary as each of the group came from a different environment. The users were very much the ones confronted with the daily practice of the problems. Later sessions

would discuss changes made to the knowledge by the knowledge engineer and errors found by the users and the expert. The expert would often discover a flaw or something seemingly out of the ordinary, and locate the position within the knowledge base as well. This would lead to suggestions for modifications. Discussions of new knowledge or adaptations to be made to existing knowledge would not necessarily be restricted to the format of decision-tables. The group saw the system's representation structure in the task model, and structure in the domain model. Even though this was only implicit in the knowledge model itself, they made explicit use of it.

With certain proposed adaptations, it sometimes was the case that the knowledge engineer would be in disagreement with the expert and user about the way specific changes should be modelled. In most instances, this would concern system's ordering of the questions. The inference engine dictates this order based on its needs, whereas the expert and user in some cases expected a specific question to surface based on their view of the domain. This eventually led to a principled decision to take the expert and especially the users opinion as the preferred form. The behaviour of the system should make sense to the user and the expert, rather than insist on the complete correctness of the knowledge model. The current version of the KBE circumvents these problems by allowing the order of questions to be influenced from outside the knowledge model, in the preference file for a knowledge model.

Test cases were used to validate the system, but care was taken not to use and over-use a fixed suite of cases. Therefore, the group tried out as many different, actual cases as possible. One of the users got a name for being able to unearth strange cases from practice, such as attics where people created an inside garden, with pond and fountain. The cases stretched the extent of the system to cover as many different types of situations as possible.

### 6.4.3 Knowledge System Development

VDES was deployed using a standard consultation system, and is currently still being evaluated by its users. An initial deployment carried out to give the users an idea of the eventual system was a useful source of feedback on the functionality of the knowledge system, as well as its visual presentation. At first, a number of workshops provided a controlled environment for user training. The workshops also enabled the developers to hear their reactions and ideas on possible changes. Several housing associations are now testing the system in practical problem situations.

The system has shown that the quality of the knowledge model itself is quite good, as the users did not report any serious errors in any of the diagnoses. Extensive testing on actual cases had already taken these out of there. The users did request changes to user-interface of the then default consultation environment. These requests concentrated on making the system easier to use. For example, the location of the moisture problem is easier to provide using a picture of a room rather than through a succession of textual questions.

Some questions required too much expertise on part of the user. One of the reactions to this is to integrate an arithmetic unit to calculate a critical heat coefficient value, rather than let users make an educated guess. Only some of the people associated with housing associations are capable of doing this. By performing a simplified finite element analysis using some simple observable measurements as parameters, this turns into a reasonably easy question. This is however still a future issue.

Plans exist to prepare the system for general use by a far greater user-population and beyond making it easier to use, it is essential that the users have access to the latest knowledge for diagnosis. That is one of the reasons why an Internet based consultation system is being considered. This includes the development of a report using the XML report generator, with more attention to background information. Currently all the reporting consists of display commands present in the knowledge base.

The further separation between knowledge representation and knowledge system presentation gives advantages in maintaining the system and making it easier to work using a continuous knowledge engineering approach. A further development possible step is a remodelling of the VDES knowledge model in IO. This solves some of the problems experienced in the system, in particular the problem with surrounding rooms. Without this transformation the current problems facing VDES will not be solved, as many revolve around the fundamental instance problem.

#### 6.4.4 Conclusions

VDES is still under development today. It is considered one of the systems that would be most helpful to the public. Therefore, it may in the future be deployed over the Internet as an e-commerce application.

<i>People:</i>	Participation of expert and user on knowledge model	<i>Cost:</i>	--
<i>Project:</i>	Continuous, product based	<i>Risk:</i>	--
<i>Product:</i>	Default and customised knowledge system. Knowledge model as scientific theory.	<i>Clarity:</i>	+
<i>Process:</i>	Cyclic with many iterations, coupled with extensive validation, verification and testing.	<i>Benefits:</i>	++
<i>Tools:</i>	KBE	<i>Bottom-line:</i>	++

**Table 6-6 Evaluation VDES**

The knowledge system is an example of strong participation in the knowledge modelling by an expert and user. It shows the combined effect of the process on the quality of the knowledge of the knowledge system and of the expert. In fact, VDES provides further evidence of the necessity of the participation in developing a knowledge model. It also shows the effect that an explicit role for a user early on in the development can have on the quality of both the knowledge model and the knowledge system. Furthermore, it makes clear that in a knowledge system, the user and expert may make other choices than a knowledge engineer might make.

The development of VDES is a mixture of scientific insight-based concerns and engineering concerns. The level participation of the expert and user speaks most strongly of this. The project however is continuous in nature in concordance with the evolving nature of the knowledge contained in the system. While the knowledge system may require some presentation improvements, the majority of changes made to the system originate in the advanced made in the field of moisture damage. The process that was used to develop the system was very cyclic, and a wish to deploy more of the versions developed is felt but not easily realisable without a central Internet deployment scheme. For the KBE VDES is another such story as it shows a system that is used by a true user population and with a few adjustments this user-population may grow to include any home-owner and tenant in the country.



## 6.5 Fire Regulations Advisory System

The Fire-safety Regulations Advisory System (FRAS) is another example of a system supporting the use of legislation. This knowledge system was one of the first knowledge systems developed at TNO. The first phase employed the KBE technology to develop a knowledge-model, to be used as a design for the development of a standard software system. The second phase saw a re-implementation of that system using IO. This latter phase aimed to solve some of the problems experienced in the earlier version, mainly to do with reduction of the cost of maintaining the system.

### 6.5.1 Overview

One of the earliest laws in existence, enacted by Hammurabi, King of Babylon, already contained an article calling for the death penalty for the builder or designer who would deliver an unsafe house that collapsed and killed its occupant. The current legislation having to do with the safety of buildings in the Netherlands, which is the subject of consideration here is the Building Decree (in Dutch: Bouwbesluit). The decree adopted in 1983 aims to collate the existing collection of local council regulations into a single piece of national legislation. Introduction of this decree aspired to provide simplification and unification of the rules and regulations that buildings must meet. It states the rules to uphold when a building is to be safe for the people that use and inhabit them.

Every architect that puts his pen to paper, every builder constructing a new building or altering an existing building, and every owner of any building, must abide by the Building Decree. It is the law that most people involved in the process of building and construction find on their path on a regular basis. A fire in a café on New Year's Eve killed several people and scared many more for life, rendering crystal clear what the consequences are of not being up to code.

The law is sizeable and complex, and is not easy to understand, even though the authors paid a lot of attention to this aspect. The wording is difficult because of the necessity to state the intent of the legislation in an unmistakable manner. A single word in an article can drastically influence the meaning of an article. The articles become quite technical in points, filled with graphs, tables and formulas, and littered with references to norms and standards.

This situation was the starting point for contemplating the development of a system for support of professionals that require the Building Decree in their daily work. This document-information system called BCS (in Dutch: Bouwbesluit Consultatie Systeem) support those people employing the Building Decree. In fact, it is the main form for the majority of that employ it. The BCS contains regulations from the Building Decree, supplemented with information on related technical standards, standards of quality, jurisprudence and officially accepted alternative solutions. The BCS also integrates some other software components that are relevant to users of the law.

One of these systems is FRAS, a knowledge system for the inspection and design of buildings according to the part of the Building Decree the fire regulation (also see (Lucardie 1992)). The fire regulations are one of the most used parts of the decree and often by people not formally trained in building physics, chemistry, etc., for example firemen.

The intent with the inclusion of this knowledge system is twofold, as the two different phases provide contrasting approaches to knowledge system development. The first phase shows an example of a first generation knowledge model as a design for a standard software system. This gives some insight into the problems and benefits attached to such an approach. The second phase gives a view of the realisation of a knowledge model using the IO system. This shows a system development more in accordance with the insight-based metaphor.

## 6.5.2 Knowledge Modelling

### Task

The expert employing the building decree has to support someone using the decree during the design of a building or to inspect an already existing building. For the design-advice and critique may be necessary. For control, a verdict on the acceptability of the building under the decree is necessary. In many cases, the user is interested in evaluating a single aspect of the legislation, for example, in a setting where the design of a building is considered only certain aspects of the law are relevant at a certain level of detail in the design. When examining such a ‘problem situation’ there may only be a small number of relevant articles. To support locating these relevant articles within the legislation and to make it more transparent and understandable, the authors gave a great deal of attention to the internal structure of the legislation. Furthermore, the formulation of the law was fashioned to incorporate the rationale behind the different articles.

Different subdivisions within the legislation allow one to focus on a specific subject or aspect of the legislations. By combining different subdivisions, very specific cross-sections are possible.

The first subdivision uses the *purpose* of the building activity:

- New buildings
- Existing buildings
- Alteration

The second subdivision is according to *function* of the building:

- Dwellings and residential buildings
- House-trailer and fixed stands
- Not meant for habitation
  - Office buildings
  - Accommodations and hotels
- Other constructions

Orthogonal to this is the structure of the subjects within a chapter, divided into sections, articles and paragraphs. Each chapter describes a specific purpose and function combination, for example a newly developed office building. Each such a chapter divides into at the most four sections, with subject matter arranged in paragraphs as described below:

Section 1: Requirements from the perspective of *safety*:

- constructive safety
- user safety
- fire safety

- social safety
- equivalency for safety

Section 2: Regulations from the perspective of *health*:

- protection against harmful or irritating influences
- protection against harmful or irritating substances
- harmful or irritating pests
- water supply
- natural light
- equivalency for health

Section 3: Requirements from the perspective of *utility*:

- surface area of the trailer stand
- accessibility
- spaces and stand areas
- communal areas and stand areas
- telecommunications
- displacement and distortion
- equivalency for utility

Section 4: Requirements from the perspective of *energy conservation*:

- reduction of heat loss
- equivalency for energy conservation

The above divisions on purpose, function and by the structure of the legislation itself means that a choice on one of each of these three dimensions can give access to an specific situation dealt with by the law. For example, with the purpose of newly constructing a building, with an office function, and concerning the aspect of fire safety yields article 6.15.1.

Providing the rationale behind the article as additional information makes the legislation even more transparent. In support of this, within the document a set of basic principles expands into qualitative functional requirements, implemented by quantitative technical requirements. The legislation formulates the technical requirements in such a way that they always give information on the part or subdivision they apply to, the rationale behind the requirement, what the threshold value of the required performance is and how to measure this performance.

A high level principle is that people should reasonably be able to leave the building before they are overcome by either smoke or fire, when such an event takes place. Functionally, this means that people on for example an office building should be able to reach a safe place within 30 seconds. As a technical requirement on office buildings, a maximum distance from any point in a location to an exit of a fire-compartment is applied. The design of a building must incorporate this to comply, and an existing building is in violation of the building code if it is not the case. Beyond making the different articles more transparent, this supports innovation, as different technical solutions can be functionally equivalent. If someone can provide an alternative technical solution that can take people out of that location to a safe place within 30 seconds, it is allowable under the building decree. Of course, only after extensive testing has proven the effectiveness of that alternative.

An outer dividing construction of an accessible area, a toilet space of bathroom (1), to limit heat-loss by transference or conductivity (2), determined in accordance with NEN 1068 (3), a heat resistance of at least  $2.5\text{m}^2\cdot\text{K}/\text{W}$  (4)

1. Part: the context, or condition of the requirements
2. Rationale: the functional requirement for the performance requirement
3. How: the method by which the performance target must be determined
4. Threshold: the performance target

**Figure 6-14 Anatomy of a Decree Article**

The example above shows a schema as present in the anatomy of the corresponding article in the Building Decree. Typically, the first paragraph of an article states the general case, as in the example above. The later paragraphs contain exceptions, and the final paragraph contains information about possibilities for equality.

The context is determined by the first opening of the article and the location of the article in a chapter and section, as was discussed above. The second part of the article is rationale behind the article. It provides a background reason for the incorporation of the requirement. The next element is the method used for verification of that requirement. If the determination method is simple, such as the distance between two exits, the how is not specified, for more complex determination methods a link is made to a separate standardised norm (NEN-norm), that describes in detail how the property must be measured. Finally, the threshold value gives the point at which the building satisfies the article.

The principles and their elaboration for different parts of the construction are dependent on the intended use of the building or part of the building. This is especially valid for the performance requirements; at the level of functional requirements, the intent is general. For example, to be able to safely exit a building is a valid functional requirement for any building. The level of the demands is amongst other things dependent on the number of people, their physical state, their level of attentiveness and their familiarity with the environment.

Different types of buildings have different characteristics. An office is populated and constructed in a different way than a typical dwelling. People inhabiting know the way around their house and they are typically small. The legislation reflects this by realising functional demands through differing technical requirements. Another difference exists between buildings predating the introduction of the legislation and those building considered new.

### *Fire Safety*

Fourteen chapters in the legislation contain in total some 420 articles of law. About a third of these are concerned with fire safety. As the law provides for differences in the evaluation of existing and new buildings, and further sub-divides into different types of building, with different demands on each, there can be a distinction between general and specific articles of law. For example, because the inhabitants of accommodation are less known to the environment, they require shorter exit routes and more in terms of fixed exit pathways. As can be understood from the previous, the

fire-safety requirements are found throughout the legislation and form a cross-section of the different chapters.

With the specific subject of fire-safety, the following aspects play a role in a functional sense:

1. Limit the possibility of the start and development of fire
2. Limit the possibility for the expansion of fire
3. Limit the start and expansion of smoke
4. Presence and layout of escape routes
5. Prevention and limitation of accidents in fire-situations
6. Enabling fire fighting

These functional demands unfold into performance requirements for the properties of parts of the constructions. The materials used for some parts of the construction should have proven properties in flame resistance, flame retardation, and smoke production. The construction should be fire-resistant to collapse, have resistance to fire penetration and transfer, and have resistance to smoke transfer.

These high-level principles and functional demands also conform to aspects for either design or inspection. An example of such an aspect is the compartmentalisation to stop the spreading of smoke in a building. About fifteen different articles of law are concerned with smoke compartmentalisation. To say that a building fulfils the requirements of these articles is to say it complies with the decree for this specific aspect. When a building complies with all aspects, it complies with all the fire regulations in the building decree.

### **Knowledge Modelling – FRAS-KBE**

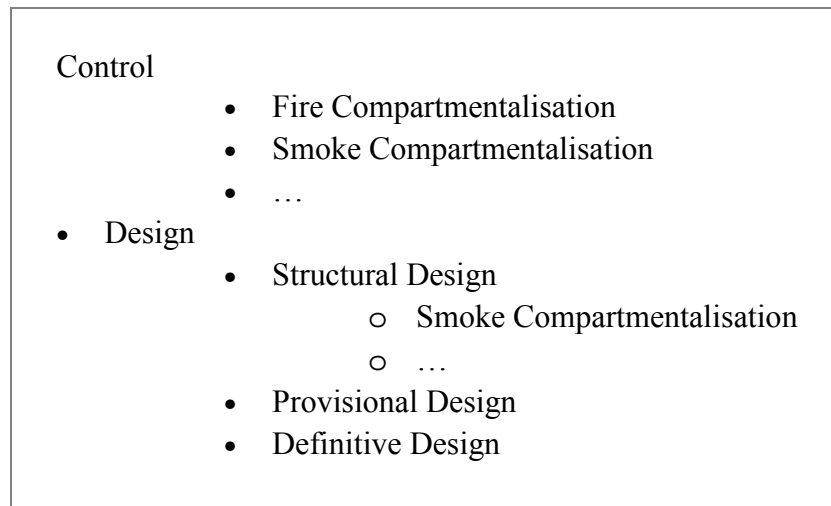
The KBE supported the first phase of the FRAS development. The knowledge model in this phase constituted an operational design specification for a standard software system. This is very much like the approach proposed in CommonKADS.

#### *Domain Model*

The KBE split the domain model of FRAS into three separate knowledge models. The different types of sub-divisions of the legislation were flattened into a more generic theme tree. The division into different knowledge bases intended to make the knowledge models more manageable in development and enhancement.

The order in which they were developed was office building, followed by hotels and lodgings and finally dwellings and residential accommodations. Offices, hotels, and lodgings have a great similarity between them, whereas dwellings overall do not have many demands placed upon them by their limited scale and the familiarity of the inhabitants with the dwelling.

The first subdivision in the theme-tree is by the task, either control of an existing building or support with the design of a new building. The following divisions are by subject, as known from the aspects of the fire regulations. Design divides these aspects among the relevant design phases. For the Control theme there is no such subdivision. When examining a building for control purposes the full extent of the law, i.e. all the relevant articles are employed. The theme tree therefore captures some important logic about which parts of the legislation to employ, when faced with a particular task. This is made explicit in the theme-tree



**Figure 6-15 Excerpt from Office Building Theme Tree**

This tree organises all the possible ways to employ the legislations, for purpose, function, and aspect. The theme-tree leaves commonly contain the knowledge to perform the task of evaluation a very specific subject within the legislation. Some of the intermediary themes in the tree may have some knowledge that helps determine the exact subset of legislation, to accommodate certain specific arrangements in the law.

The theme tree allows the knowledge engineer and expert to have a large number of small knowledge bases, rather than one integrated one. This supports the incremental development and helps in the localisation of the relevant part of the knowledge model. This can be very important when making changes and enhancements.

#### *Task Model*

1. Select house type
2. Select design or control mode
3. Select theme until leaf is selected
4. Evaluate rules from theme

In effect, this means that the theme tree organises a large number of very small knowledge models, rather than constitute a single large model. This simple task model suffices because the remainder of the evaluation is implicit and depends on the specific of modelling of the articles. The building must comply with each of these rules to pass the test for this theme.

All three KBE knowledge models contain the same kind of theme-tree. Although similarities exist between the hierarchies they define, they are not in agreement. The use of terminology is inconsistent in places and sub-divisions are organised differently. The division in three separate models makes it hard to appreciate these inconsistencies, but even in a single integrated model, it would not have been much clearer. Duplication exists within each knowledge base because each article is included once for design and once for control. A further redundancy is present because of the overlap between the three knowledge models. These factors multiply the effort required for a single change to an article. Such a change requires updating six representations of that article in the different knowledge models. This combines to

make changing the knowledge model hard, and difficult to verify. The side effects of a change to a knowledge model become especially hard to appreciate.

#### *Process*

Knowledge acquisition proceeded in a very traditional fashion. The knowledge engineer organised many different sessions with the experts, and developed a prototype model. The first product of this activity was the idea of the theme-tree structure. This tree grew more detailed as the knowledge model evolved. After the first part of the theme tree fleshed out, the knowledge engineer started with the development of the different small sized knowledge models on the articles of law. A typical module in the knowledge base as a leaf of the theme tree treats one article, but some treat between two and five.

When the first version of the office building system was finished, it was frozen and used to develop the software. The knowledge model only served as the specification of the software. After this, users and experts alike formulated the critique based on the behaviour of the system. The developers made modifications in the software to solve the problems, which subsequently had to be incorporated into the knowledge model was changed to keep the specification synchronised. Of course, in latter stages, the feedback to the knowledge model became limited. Eventually they became too much out of sync to be able to say that one was the design of the other.

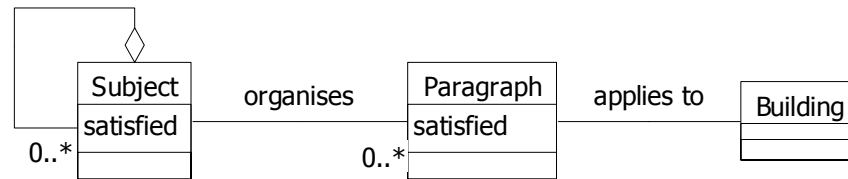
#### **Knowledge Modelling – FRAS-IO**

In the second phase, a single model was developed to contain the complete model, rather than divide it into a number of models. The intent was to create a system with the same type of functionality as the previous but with solutions to the maintenance problems and supportive of approaches where the expert make modifications to the model independently.

This was necessary, not only because the original system was showing high cost of maintenance. The government proposed to introduce a completely new edition of the legislation in 2001. This version contains not three building-types, but twelve or more. Furthermore, the legislation aimed to incorporate a kind of inheritance hierarchy for these building types, so that there would be as little duplication in the legislation as possible. The developers easily appreciated what this entailed within the approach as described in the above section. Maintenance would become impossible in such a setting. The secondary goal of empowering an expert to be involved in the modelling or model independently was also not one of the options within a design-implementation track.

Thus, the new model in IO would have to demonstrate maintainability and participation of the expert. It would also have to make clear that accommodation of several different building types would not entail a problem. To limit initial development of the knowledge model to a realistic subsection of the legislation, it was determined that the initial modelling would concentrate on smoke compartmentalisation. This part of the legislation constituted a major part of the knowledge in the phase I models and contained some more interesting modelling problems.

#### *Domain Model*



**Figure 6-16 FRAS Abstract Model**

The model in this phase was more explicit about the structures found in the domain, leading to a high-level of the task structure, seen in the abstract model above. It contains the subject tree, by an aggregation-relationship from the Subject class to itself. Furthermore, it shows a Subject organising the different Articles. It further elucidates the fact that an Article makes use of the Building under consideration to make judgement. The first activity involved general discussions of the domain, the regulations and their organisation. This led to an abstract model of the domain, shown in the above figure. The different descendant subjects are organised into a tree of sub-category subjects, which in turn organise the paragraphs. A Subject is satisfied if all the paragraphs it organises are satisfied, where a specific member has access to the knowledge to determine whether a Building is satisfied according to it. An Article can perform this determination by looking at the type of building and its different properties. This abstract model makes it possible to test a single Article separately, by querying its satisfied attribute. Based on this abstract model, the system's knowledge model therefore divides into three sub-models: the building model, the regulation model and the subject model. This model is not the ultimate model for a legal domain. The ability of such a framework to assimilate and bring together knowledge as well as provide a guideline to what must knowledge must be acquired is essential (Visser, Bench-Capon & van den Herik 1997).

### *Task Model*

**context: Subject**

```

satisfied := (relevantArticles ->
  ForAll(article | article.satisfied)) and
  ((chosenSubject = nil) or
  chosenSubject.satisfied)
  
```

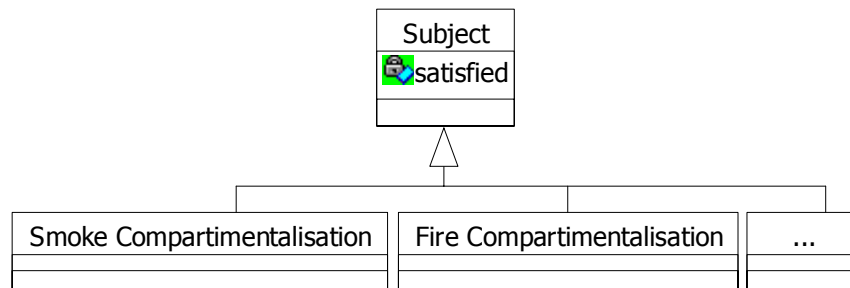
**Figure 6-17 Main Algorithm for Task Model**

By calling the satisfied property of the root subject, the selection of chosenSubject starts and following that determination of the satisfaction of contained subjects, or relevant articles of law. By virtue of this procedural representation this is more direct and informative than by forcing it in a decision table.

This is one of the major differences with the modelling in the KBE. First, its simple representation precisely does not allow reasoning over multiple instances such as the example illustrates here. The natural expression for the expert does however revolve around this kind of manipulation of concept. Secondly, the categorisation principle enshrined in the decision table would be going against the grain of the task that has to be performed.

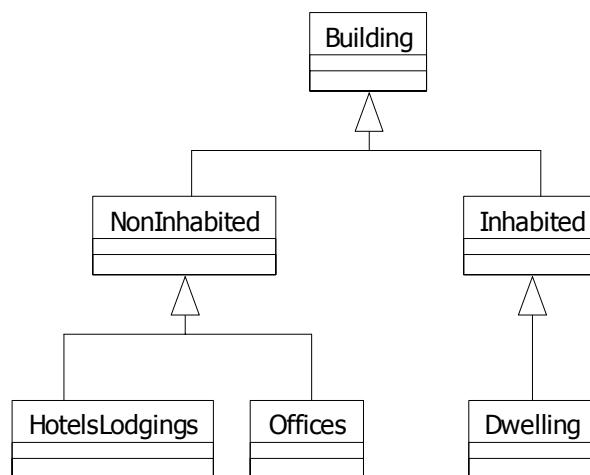


The task model shown therefore contains the earmarks of the vivid modelling it aims to support. These models remain easy to understand by experts and can easily be explained to external parties. This further supports the verification and validation of the system as well as provide a greater sense of trust in the operation of the system with users and clients.



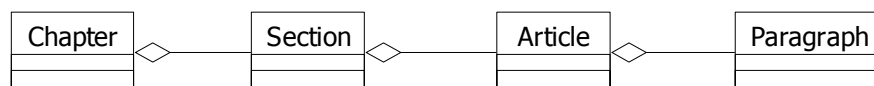
**Figure 6-18 Subject Model**

The evaluation of specific rules is impractical for the standard user, so a hierarchy of subjects is used. This organises the knowledge of the expert on categorising the sections according to subjects. These include fire compartmentalisation and smoke compartmentalisation.



**Figure 6-19 Building Model**

The building model contains three building types that are not abstract: office building, lodgings and dwellings. The evaluation of a subject uses properties of the specific type of building under consideration. The building model can easily be adapted to accommodate new building types. This is necessary because the second phase of the Building Decree will introduce many more types of building.



**Figure 6-20 Regulation Model**

The regulation model mirrors the organisation of the paragraphs, articles, sections and chapters. Each of the relevant articles and paragraphs is contained in the regulation model, with the possible to query its contents separately. This allows testing and verification of individual parts, which aids the development process. It is possible to

use this to determine whether a user's building satisfied a specific subsection of the legislation, in a manner analogously to the current use of the subject tree.

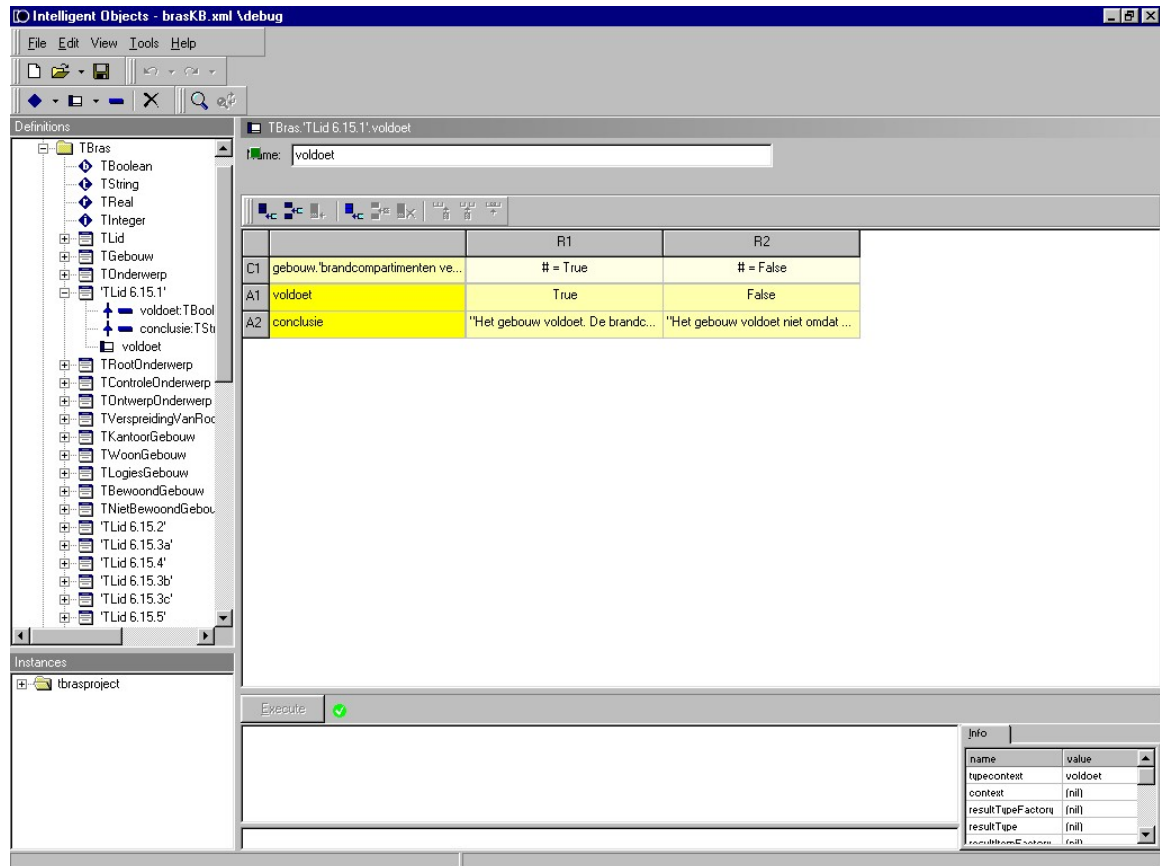


Figure 6-21 Modelling of FRAS in IO

In the above screen, the modelling environment of IO contains the FRAS knowledge base. It is displaying the decision-table used in the determination of the satisfied property of article 6.15.1.

		R1	R2	R3	R4	R5	R6
C1	gebouw.'hoogte hoogste verblijfsgebied'	# <= 50	ELSE				
C2	gebouw.'verblijfsgebied met toegangsluis'	-	# = True				ELSE
C3	gebouw.'lengte van toegangsluis'	-	# < 2	ELSE			-
C4	gebouw.'toegangsluis besloten'	-	-	# = True		ELSE	-
C5	gebouw.'toegangsluis is rookcompartiment'	-	-	# = True	ELSE	-	-
A1	voldoet	True	False	True	False	True	False
A2	conclusie	"Het gebouw voldoet aan Lid 6.15.2."	"Het gebouw voldoet niet aan <a href='http://www.bouw.tno.nl' target='_blank'>lid 6.15.2</a>. Een toegangsluis moet een lengte hebben van ten minste 2 meter."	"Het gebouw voldoet aan Lid 6.15.2."	"Het gebouw voldoet niet aan <a href='http://www.bouw.tno.nl' target='_blank'>lid 6.15.2</a>. Een besloten toegangsluis moet ingericht zijn als zelfstandig rookcompartiment."	"Het gebouw voldoet aan Lid 6.15.2."	"Het gebouw voldoet niet aan <a href='http://www.bouw.tno.nl' target='_blank'>lid 6.15.2</a>. Indien de hoogste vloer meer dan 50 meter boven het meerniveau ligt, mogen trappenhuizen alleen via een toegangsluis worden bereikt."

Figure 6-22 IO Decision-table Detailing Member 6.15.2

The figure above details the article 6.15.2. It determines that if the highest elevated living area of an office building is higher than 50 m above ground level, and the living area has an entry lock, the length of the lock must be less than 2 m. or the lock has to

also be a smoke compartment. Otherwise, it does not comply. The conclusion attribute contains the different explanations of why the building does or does not comply, through the last action in the decision table.

### *Process*

The knowledge modelling commenced with a small number of discussions between the knowledge engineer and the expert. This led to the abstract model described earlier. The expert then proceeded to develop the three hierarchical models. Partly because the expert was the one that could better appreciate the different aspects of these models but mainly to accustom the expert to the object oriented modelling formalism. The knowledge engineer incorporated the proposals for the building and regulation model into the model, but at first added no properties or decision tables.

The skeleton of the system in place, the knowledge engineer and the expert started with modelling a single article, namely article 6.15.1. The representation of this article as an individual class enabled the validation of its representation through testing. This specific article is quite simple as it states that the fire compartments in a building must be divided into smoke compartments. There is no actual inference here just a question for the user whether this is indeed the case. This simple article proved an excellent starting point to familiarise the expert with the decision table format, and the abilities of the IO system.

Using this article the knowledge engineer constructed a prototype implementation with only one root subject and one article of law. This was the basis of the eventual system, as further modelling would entail only enhancement of this basic structure. The expert subsequently verified and validated this first knowledge model. In essence, this system was the barest skeletal form of the system that was operational.

The design provides different extension points that prepare for further elaboration. Adding more subjects to the theme tree, adding articles to the leaves of the tree and introducing new types of building to enhance this simple system is quite simple. The knowledge engineer and expert filled the subject tree in part, adding design and control subjects the collection of main subjects: smoke compartmentalisation subject and its sibling themes. Smoke compartmentalisation was the only one that was implemented with relationships to actual articles.

<b>Any Building art. 6.15</b>	<b>Office art. 7.5.3</b>	<b>Accommodations art. 7.6.7</b>	<b>Dwellings</b>
6.15.1	=	+7.6.7.1	=
6.15.2	=	+7.6.7.2	=
6.15.3a	+7.5.3a	=7.6.7.3 (overriding)	=
6.15.3b	+7.5.3b	=	=
6.15.3c	=	=	=
6.15.4	=	=	=
6.15.5	=	=	=

**Figure 6-23 Relevant Articles Smoke Compartmentalisation**

The different relevant articles showed unevenness in overriding behaviour in the legislation. Some articles for a specific building type define additional requirements while others replace existing general articles. The article 7.6.7.3 overrides the more general 6.15.3a article. The knowledge engineer and expert discovered this through communication with the other experts and the authors of the legislation, as this was

not obvious from the text. This and other insights into the legislation text were forwarded to the legislative body.

During the final stages of the project, as a test, the expert modelled some of the knowledge independently. Little instruction was required because the expert was present during almost all modelling activity. This made it easy to perform certain tasks, change a prompt for ask-user questions and adding new articles to the knowledge model.

This prepared the expert for a demonstration of the new knowledge model for the client. During a final evaluation of the knowledge system, the expert performed several small and medium sized tasks on the knowledge model as an evaluation for the client. The expert made clear that “using this approach the experts could perform the majority of maintenance activity.” He attributed this to the ease with which he could find the location of the position where changes needed to be made and the fact that the form of representation in was not difficult to master “just a couple of days of training are sufficient to work with the current version of IO” (Janse 2000). The expert further reported the knowledge model to be more compact, concise, and conveniently arranged.

### **6.5.3 Knowledge System Development**

As the development of this system has seen a two very different phases, this section discusses each separately.

#### **Knowledge System Development – Fras-KBE**

The first phase initially used the default stand-alone system for the consultation of the system, with limited abilities for the customisation of the user-interface and did not possess any possibilities for changing the order of the questions from the presentation. It interacted with the users purely by a question-answer interface, providing limited reporting facilities, based on the display commands in the knowledge base. The more advanced features of the KBE were not available at that time.

As the client required a dedicated system that integrated with the remainder of the BCS, this was unacceptable. The desired system had to present the questions and responses in a more visually attractive way and allow customisation of the complete communication with the user, including the order. Therefore, a decision to used the KBE to develop and test the knowledge model, but to use the knowledge model as an operational specification for software development.

The FRAS system was the result of that development. It is based on a hypertext-document architecture. The hypertext documents are in a proprietary format based on RTF and meta-tag notations, with a custom-built system that could navigate hyperlinks. This involved conditional expressions and open text input. This enabled complex paths through the hypertext tree (equated by the designers as a form of reasoning). This was before XML was introduced.

With the decision table form of the knowledge kept as the original and maintained separately as a design model, all changes to the design were reflected in modifications to the RTF files. This was not an automated process, partly because of these visualisation and implementation issues. Using this approach, the developers were able to create a system integrated into the BCS document information system.

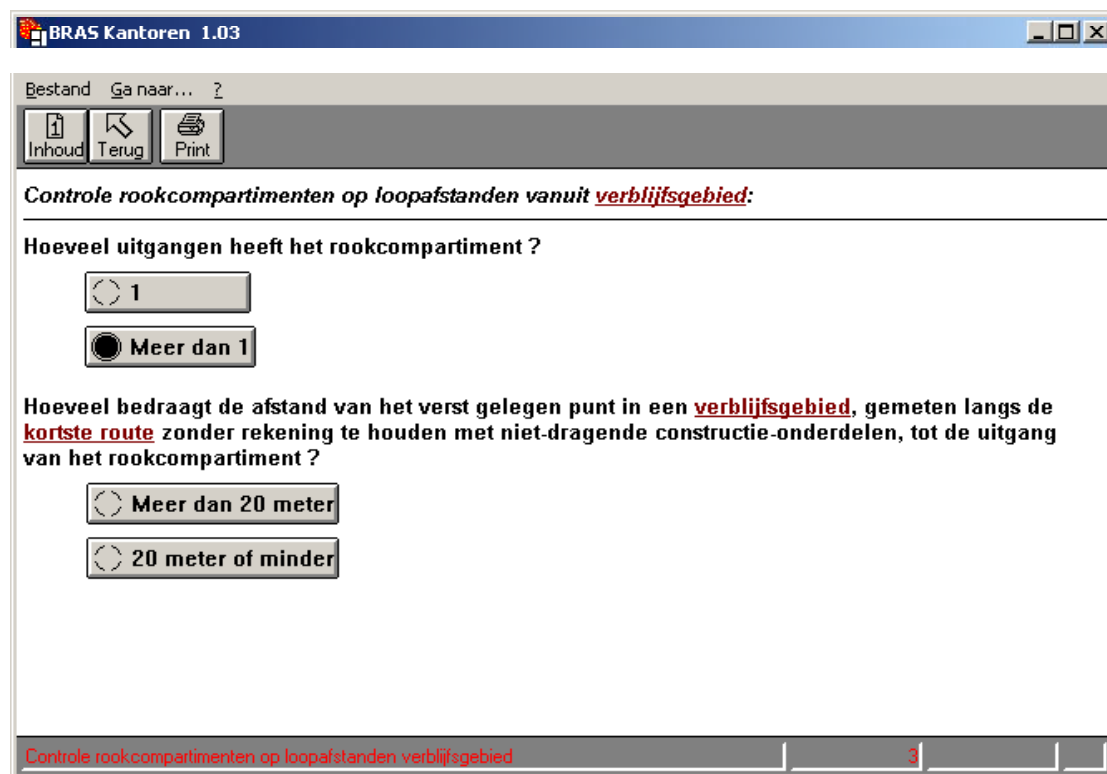


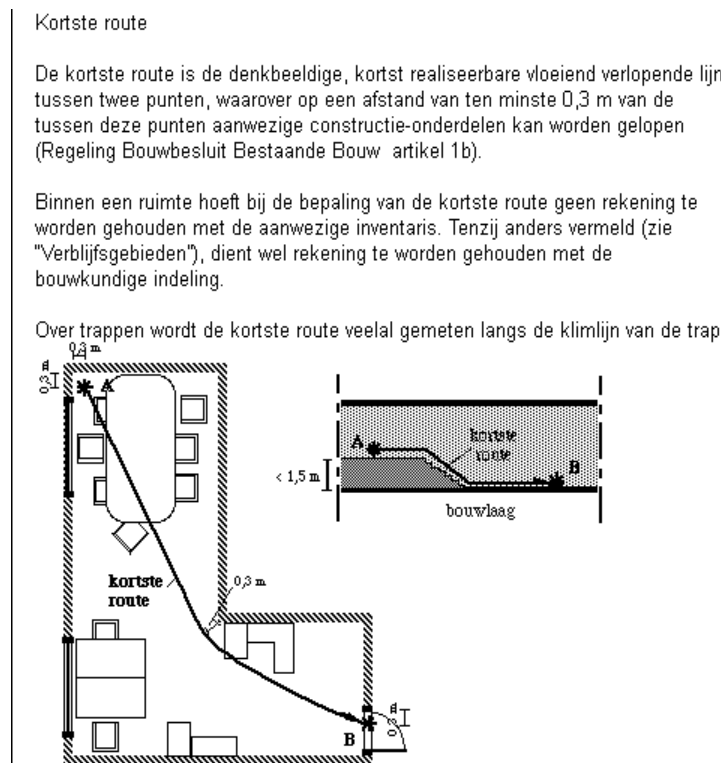
Figure 6-24 FRAS-KBE User Interface

FRAS enables inspection of and design support for office buildings, accommodations and dwellings:

- regulations for the confinement of fire
- regulations for the facilities to flee a building
- regulations for facilities and adaptations to allow fire fighting and rescue
- regulations to determine proper procedures and measures to confine hazardous materials from a viewpoint of fire or other dangerous situations

Additionally the explanations given in terms defined in the legislation allows use of both text and imagery. The material originates in introductory course materials, for training in the use of the Building Decree. It was thought that this would help explain some difficult terms, as an example, the initially considered easy to perform determination of the distance between two points is in actuality dependent on more involved definitions.

For example, the entrance of a smoke compartment and a place in a location in a minor living compartment is calculated without regard to non-permanent elements, from the farthest point in the space along the shortest route, staying 30 cm from any wall. Figure 6-25 shows the explanation of this in the form it is contained in the system (in Dutch). In this form, FRAS was deployed to all of its users, about a hundred people in total.



**Figure 6-25 Explanation Example**

The amount of effort involved in maintenance is significant. The main cause is the many locations where changes need to be made for a single change in the law. The same applies to a certain extent to errors found in the design of the decision-tables, or flaws present in the translated result in this approach. To make a change to one article, twelve places in the knowledge model and implementation have to be located, assessed and perhaps changed. For each of these changes verification and testing of possible side effects is necessary, which can require further changes.

Awareness of the maintenance cost of the system grew over time, even though it was not expected to be as bad as it was. The volume of modification was not overwhelming, but it required too much work. The consequences for any large-scale modifications were such that any approach would most likely be a complete re-implementation effort, or a maintenance effort with more risk and uncertainty associated with it that would turn out to be as expensive as re-implementation.

### **Knowledge System Development – FRAS-IO**

The second phase knows little in terms of knowledge system development, as it employed the default consultation environment incorporated in IO. As this is in fact the iKnow consultation environment, in theory this default application could already serve both standalone and Internet use of the FRAS system. The customisation facilities were also available but because the focus was mainly on the maintainability in this project, they were not employed in this early phase. The next phase of the project will pursue the finalisation of the knowledge model, and the development of a customised user-interface.

### **6.5.4 Conclusions**

FRAS has shown an application with an extensive development history. The system treats a relatively simple subject in size and complexity, but to make it a useful

operationalisation the system needs to be customisable in its presentation to the user and maintainable in its knowledge model.

### FRAS-KBE

In its initial phases, the approach has all the earmarks of the current standard in knowledge system development, and is a clear example of the engineering metaphor's influence, where the knowledge model is a design specification. It shows direct evidence of the problems that can arise by this approach, and the barriers that it raises for participation of the experts on the content of the model. The development cycles to propagate changes become especially long, and very intensive (i.e. expensive). The system furthermore shows the importance of customising the user-interfaces for a knowledge system and auxiliary issues. This was solved in this phase of the development by realising it as a standard software system, allowing all kinds of adaptation to the users needs. In addition, it shows a KBE approach reaching the limits of its possibilities, detailing some of the maintenance problems experienced.

<i>People:</i>	Indirect knowledge acquisition	<i>Cost:</i>	-
<i>Project:</i>	Finite, product-based	<i>Risk:</i>	=
<i>Product:</i>	Knowledge model as specification for a software system	<i>Clarity:</i>	=
<i>Process:</i>	Incremental design, waterfall implementation followed by maintenance	<i>Benefits:</i>	=
<i>Tools:</i>	KBE	<i>Bottom-line:</i>	=

**Table 6-7 Evaluation FRAS-KBE**

Knowledge modelling in this phase was quite indirect. The participation of the expert was mostly circumstantial and restricted to the first phases of development. The project commenced as a finite project, with a clear-cut hand-off point after finishing the knowledge model. This failed and created a situation that can only be described as chaotic and unstructured development at high cost and risk. The knowledge system functioned as a product, with critique based on the behaviour of the product. This was the case even though changes in the knowledge required repair or changes in the knowledge model. This phase of FRAS initially followed a waterfall model, but quickly and unexpectedly turned into an incremental model. The developers did not intend such an extensive maintenance activity, which led to planning and control problems in the project.

### FRAS-IO

The second phase shows an application of the Intelligent Objects system. The description had made clear the added complexity of employing Intelligent Objects. There is not merely a cost attached to this complexity, but the additional support for making structures and procedures explicit supports the discovery of new insights, eases the evolutionary development and facilitates localisation of parts of the that are affected by a change. It further demonstrates the conciseness introduced by the object-orientation and its inheritance.

FRAS-IO is a small-scale realisation of the same knowledge found in FRAS-KBE. This case study has fashioned a structured, explicit representation of the knowledge. It shows that this supports the expert participation because the model is easier to understand and it allows the expert to locate elements within the model more easily. Furthermore, the object-oriented representation's encapsulation and abstraction allow

more concise modelling, reducing the redundancy found in many KBE models. The second modelling phase also shows that IO supports an expert to model parts of the knowledge model independently.

<i>People:</i>	Participatory and partially independent modelling	<i>Cost:</i>	-
<i>Project:</i>	Continuous, product-based	<i>Risk:</i>	-
<i>Product:</i>	Default knowledge system	<i>Clarity:</i>	+
<i>Process:</i>	Cyclic with limited number of iterations	<i>Benefits:</i>	+
<i>Tools:</i>	IO	<i>Bottom-line:</i>	+

**Table 6-8 Evaluation FRAS-IO**

The second phase is then also an example of a participatory modelling. It also shows in a limited sense the ability to perform independent modelling by an expert. The project is from the onset meant to be infinite in nature as is evidenced by the approaches focus to make the expert responsible for the development of the knowledge system. The products are in the first place is the knowledge model and in second place the knowledge system. The process of development is from the very beginning cyclic with complete releases at every point in the development. This has also influenced the design within the knowledge model, to exploit this style of development to the full.

## 6.6 Mebis

The Mebis system supports the production of concrete. It calculates concrete mixtures based on user requirements and available raw materials. This encompassing system integrates a dedicated knowledge system as a component. The KBE provided the first basis for the realisation of this component; IO supported the development of the second version. This case study shows either tool's capabilities to support such an approach.

### 6.6.1 Overview

Concrete is a versatile building material, used in all manner of constructions since its first use by the Egyptians in 3000 B.C. Its ability to be shaped into any conceivable form, in addition to its potential for immense strength, and its resistance to fire has made it one of the most common building materials in the world. It consists of a very hard mixture of cement with sand, gravel or stone parts and water. Different additives can be included in the mixture to give it attributes that are even more exotic. This makes it possible to deliver an almost endless variety of concrete products.

Concrete is produced to order a set of requirements, for its use in a specific situation. Potentially many different recipes produce concrete that will satisfy a client's requirements. The expertise in producing concrete lies mainly in knowing which recipes to use. The ingredients never are the same, for example the distribution of the size of granulates; therefore, the recipe needs to be adapted for that. The analysis of a concrete-recipe is a recurring activity, either to validate it against the requirements of a client, or to find the most optimal recipe to comply with such requirements.

While the first task is what the experts are well equipped for, the later repetitive task is hard to do by humans in a consistent and complete way. The proposed solution to solve this problem was a system that could analyse many different recipes and compare them. Because the composition and analysis of these recipes is a knowledge intensive task, this part of the system had to be knowledge system. The goal was to



find recipes that can satisfy the customer's requirements and allow them to be optimised, for instance on the production cost. The complete system would perform many more tasks than this, but a knowledge system would be an ideal choice for the difficult and potentially mutable knowledge that was relevant in this part of the system.

### 6.6.2 Knowledge Modelling

The development of the system and the modelling of the knowledge was first performed using the KBE, followed by a re-implementation using IO. This section discusses each of these in order, after a general description of the domain.

#### Task

The expert normally has to create a recipe that will be able to fulfil functional criteria. In some cases, the client will only give some information about its intended usage the expert will have to map onto functional criteria. These criteria divide into standard and additional sets as shown below:

<b>Standard requirements:</b>	<b>Additional requirements:</b>
<ul style="list-style-type: none"> <li>• environmental class: 1-5d</li> <li>• strength class: B5-B65</li> <li>• consistency area: 1-4</li> <li>• nature of the construction: unreinforced, reinforced, prestressed</li> <li>• transport: kubel, pump, pumptmixer</li> </ul>	<ul style="list-style-type: none"> <li>• work is in clear view</li> <li>• use of granular material?</li> <li>• maximum granulate-diameter</li> <li>• volume mass concrete</li> <li>• concrete colour</li> <li>• cement type</li> <li>• strength development</li> <li>• hydrocrete</li> <li>• closurefree concrete</li> <li>• additive fibres</li> <li>• addition of MHK</li> </ul>

**Table 6-9 Standard and Additional Requirements for Concrete**

The expert also has knowledge of what usage situations will require what kind of concrete. Some clients also ask the supplier to determine the requirements from a description of the site and purpose of the concrete. The starting point for the actual recipe is the description in terms of standard and additional requirements. These requirements cover about 80% of all concrete mixtures, the remainder represent specialist recipes for which the support of an expert will be required. In time, the system may become equipped to deal with such recipes.

<b>Category</b>	<b>Description</b>
Cements	Binding agents
Granular material	Sand and gravel of different sizes and distributions
Water	Water, warm water or slurry water
Auxiliary material	Materials aimed at making the concrete more manageable
Additives	Inert and binding agents

**Table 6-10 Concrete Component Material Categories**

Concrete contains several component materials, divided into six main categories. Every category contains several individual raw materials. Combining one or more individual materials from one such category creates a compound material. For

example, one type of cement *CEM I 52,5R* and another *CEM III/B 42,5 LH LS* in an 80-20 ratio yields a specific compound cement. Each point of production has a 'material package', which states which types of materials are available in stock in the different categories. From this material package, it is possible to generate all possible 'compound packages'.

A recipe for concrete consists of a certain amount of compound from each of the five categories. The specific types of material in the compound within each category and the relative proportion between compounds determine the properties of the resultant concrete. An expert has the ability to analyse a recipe, containing compounds as ingredients and know to what degree the resulting concrete will conform to the requirements. This expertise is developed over time and experience. The knowledge has significant competitive value for the expert's company, which is why no specific details on the actual formulas will be given.

The expert can also develop a single recipe based on a fixed procedure from a given compound package. For each of the compound materials, it is possible to calculate its relative proportion in order for the mixture to conform to the user's requirements. This procedure contains in total 33 formulas for calculating different aspects of the preliminary recipe. This delivers both specific values, for certain compounds, as well as describe constraints on the compounds such as the one above, which describes the acceptable granular distribution. In the end, the procedure generates a specific recipe that satisfies the requirements or yields the answer that no such recipe exists for that compound package.

However, an expert cannot easily perform a complete analysis of the possible recipes, from all possible compound packages, to choose one with maximum utility. To circumvent this, the experts employ a library of recipes. The expert chooses recipes that satisfy the criteria and fit available ingredients. The expert can analyse a subset of these recipes as to their ability to fulfil the requirements, and choose one based on cost/profit criteria.

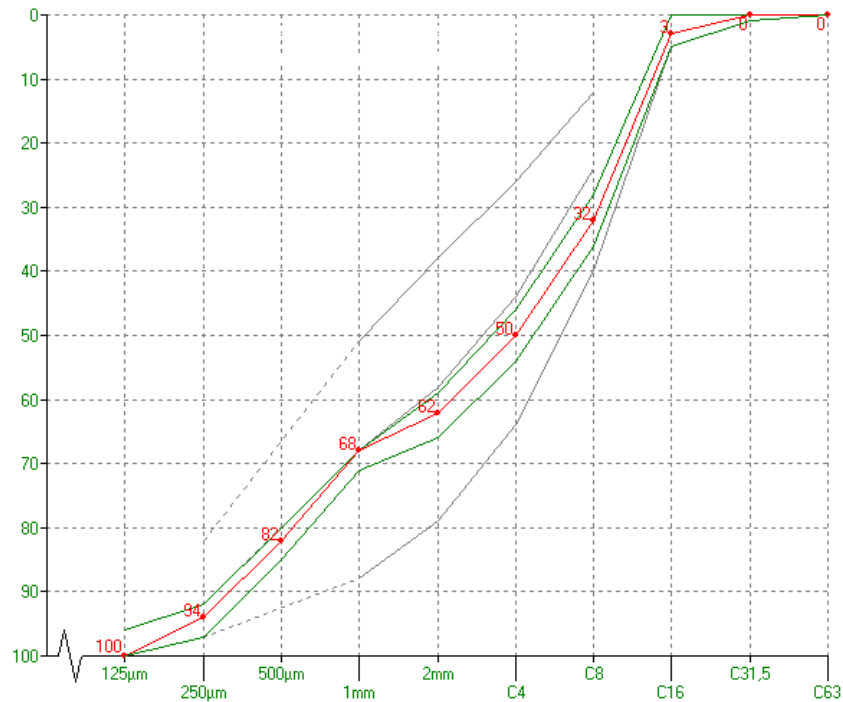
This approach has a number of inefficiencies. This approach is very sensitive to quality of the library and the ability of the expert to find the right ones. The expert is not equipped to analyse the recipes fully, and optimise the possible recipes. Furthermore, when second order utility issues become targets for increasing efficiency, such as logistics and load distribution, the task is impossible for a single expert to perform.

### **Knowledge Modelling – Mebis-KBE**

#### *Domain Model*

The formulas mentioned earlier are the basis for the domain model for the knowledge system component. The domain model contains all the relevant variables mentioned in the formulas.

Unlike most other models discussed until now there is no discernable implicit structure in the domain-model. The derivation from the formulas, as noted by the expert, already contains a model consisting of some general variables. There is no reason to form this model into more aggregate models. Rather it is advantageous to remain close to the expert model which consists mainly of these formulas.



**Figure 6-26 Granular Distribution Graph**

The area for allowable granular distribution results from a number of base lines: an ideal line, a lower boundary and higher boundary. The ideal, upper and lower granular distribution graphs are dependent on the maximum granule diameter, the processing method and the desired consistence of the concrete. Any combination of materials that is not within this area is not suitable as a possible recipe. This graph is used as a filter to remove unsuitable recipes.

#### *Task Model*

The Mebis task-model allows the development of a recipe that will satisfy the requirements given by the user.

- Determine granular distribution
- Determine provisional water cement ratio
- Calculate other compounds
- Determine actual water cement ratio
- Apply business rules

**Figure 6-27 Task Model Mebis**

The first step is the determination of the granular distribution. The determination of the granular distribution is done separate from the recipe determination procedure. This particular part of the task model creates the graph of the granular distribution as a number of lines that form the acceptable area for the granular distribution. The task model consists of a linear structure table that performs each of the required calculations in order.

The next step is to create an estimate of the ratio between cement and water. It then continues to calculate values for the other compounds, which require some initial

insight into the water-cement factor. With these values known the actual amount of cement and water are determined. The model then has all the characteristics for the recipe. Finally, it applies a filter to examine whether the recipe does not violate some common-sense rules, legislation or company specific business logic.

The application uses this to generate a covering list of possible compounds. The encompassing application initialises the model with the correct values through the KBC's API and application extracts the results in the same way.

With these compounds, it then generates all possible recipes from the possible compound combinations, with some impossible combinations filtered out by the first model. All of these are fed through the second model, yielding the compound proportions for the recipe. From this and information on the individual materials a number of properties can be calculated, including the cost of the recipe. The application performs these latter calculations.

### *Process*

The knowledge modelling process was the closest to a standard knowledge engineer-expert interaction, at some distance from the actual knowledge model. The textual report consistently modelled the formulas and tables as a complete set (Borsje et al. 1999). The knowledge engineer translated these into a knowledge model, and noted some omissions and flaws in the knowledge purely on missing values and internal inconsistencies, as well as some common sense issues. The knowledge engineer could not detect domain specific flaws, because he did not have matching skills and experience to the expert. The errors were first repaired in the document and subsequently incorporated into the knowledge model. Some of the knowledge ended up into some of the code of the Mebis system directly, because the KBE model was unable to perform the full task.

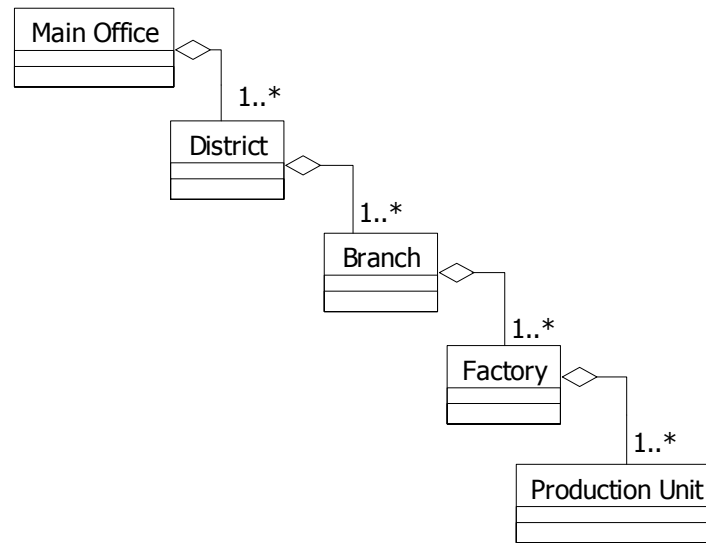
### **Knowledge Modelling – Mebis-IO**

The knowledge model in the second phase is in part a re-implementation of the knowledge in an IO format. In addition, the knowledge model incorporates knowledge about production facilities and their organisation to complete the model. This knowledge used to be in the software rather than the knowledge system component. The re-implementation of the model saw far more influence from the knowledge engineer. He applied appreciation of design rules and experience with object-oriented models to the, initially limited, abstract model of production facilities and their organisation. The expert and the project organisation perceived the resulting model this as a significant improvement. This system then enhances the capabilities of the first knowledge model by allowing optimisation over production points at different plants located all over the country.

### *Domain Model*

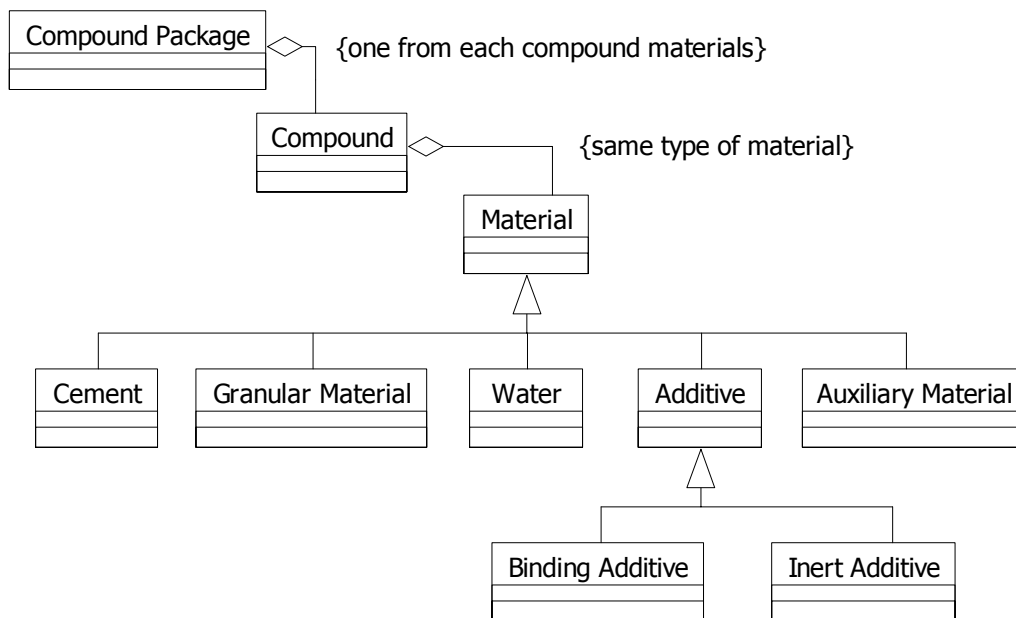
The domain model for Mebis as implemented in IO is far more detailed and more inclusive than the initial version developed in the KBE. The object oriented features of IO allowed the capture of the knowledge of the expert quite directly. What was noticed is that the model developed together with the knowledge engineer is more systematic in certain cases than the initial views of the expert's themselves. This is probably caused by the different background knowledge of the expert and knowledge engineer. It would be easy to say that the knowledge engineer's version is better, but

the distance to the understanding by the expert is greater. This is therefore not by definition a good step to make.



**Figure 6-28 Production Model**

This domain model contains the facilities from the first model for calculation of a single compound package and the resultant recipe. The domain model includes the information on the production points and geographical organisation of the production facilities. This enables some logic to operate on finding the best solution considering different production points.



**Figure 6-29 Material Model**

The different materials are now also explicit within the model. They are used to determine all possible combinations of materials into compounds packages. The constraints in the model denote the restrictions on their composition.

*Task Model*

```

context: MebisKnowledgeBase;
forall(productionPoint | productionPointSet) do
begin
  compoundPackageSet :=
    productionPoint.
    generateProductionPackageSet;
  recipeSet := expert.calculateRecipes(
    compoundPackageSet );
  resultSet->add( expert.selectBestRecipe(
    recipeSet ) );
end;
bestRecipe := expert->selectBestRecipe(
  resultSet );

```

**Figure 6-30 Task Model Mebis in IO**

The task model is similar to that of the first phase. Contained in a language method and described in the IO language, the operation of the system is now explicit and easier to verify. In a KBE model this logic would be distributed over several decision tables, or impossible to represent. In this procedural form it is open to discussion, verification and validation by the knowledge engineer, and even more importantly, by the expert.

This task model now also deals with the resource constraints at different locations. The knowledge in the knowledge model is now more complete and does not require the source-code of the application as an escape opportunity. The analysis that it can make is therefore more complete.

*Process*

The second phase of knowledge modelling had as a goal to incorporate some of the functions previously implemented in the code of the Mebis system, mainly the feeding mechanism for the analytical model. Furthermore, the knowledge model incorporated some of the structure of the production point model. This meant that much of the knowledge that was present in the code and required restructuring for usage within the IO knowledge model.

The expert reviewed the UML models as part of the model now, not in the same way as before. He reviewed them and proposed some small modifications.

**6.6.3 Knowledge System Development**

The different phases of the development of the knowledge models themselves did not have a great deal of development effort attached to them, since they were deployed as components. Their incorporation into the system was through use of their API in a standard programming language.

### Knowledge System Development – Mebis-KBE

The knowledge model that was developed in the first phase can only assess the ability of a recipe to fulfil the requirements of the client. The limitations of the KBE mean it cannot analyse more than one recipe at a time, due to the instance problem. The knowledge model therefore functioned integrated as a component system in a larger system.

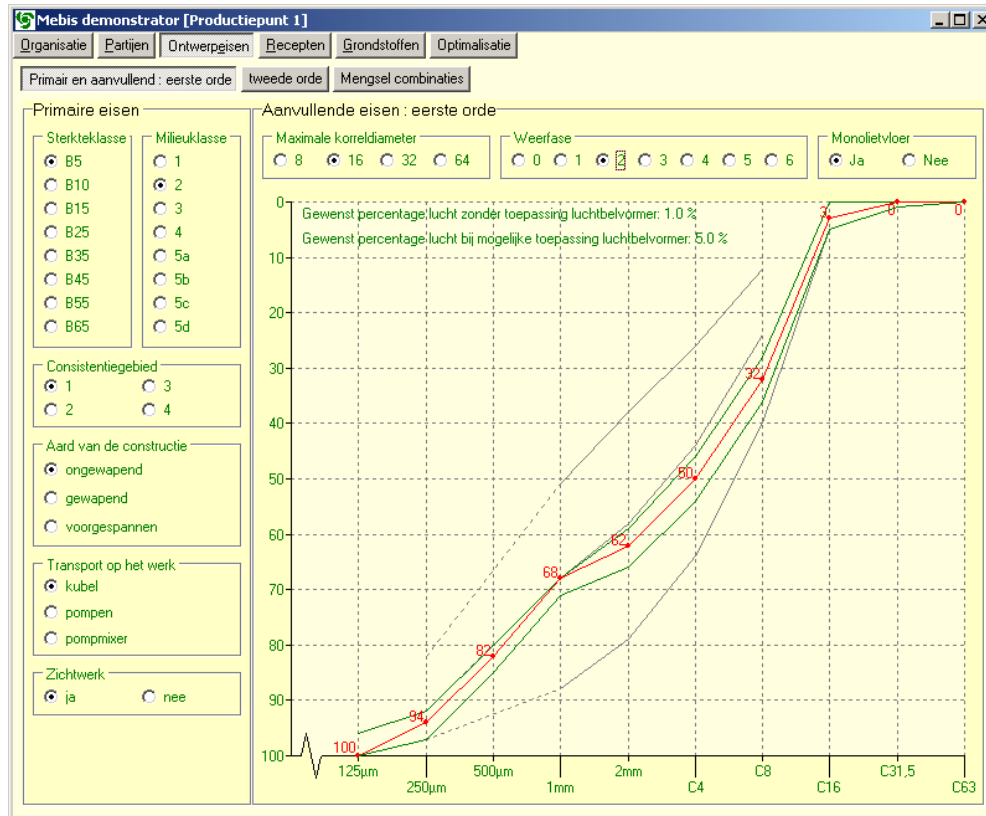
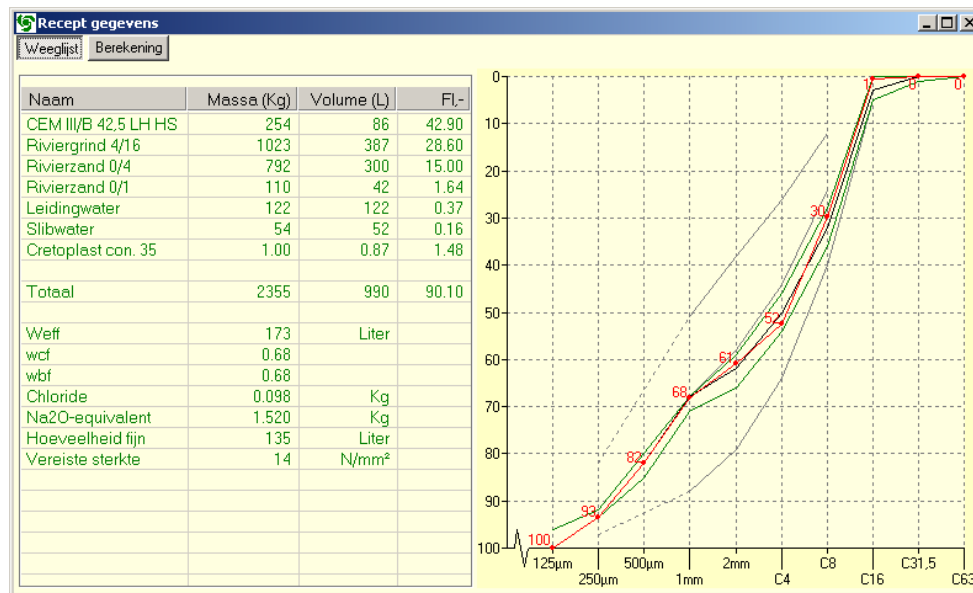


Figure 6-31 Mebis User-interface

The software used the component by supplying many different recipes, analysing them into those that will and those that will not this provide a set of correct recipes. The application has the ability to order the recipes along different criteria, one of which can be the cost of the recipe. Other criteria are available.

The complete software system was provided with a standard Windows interface, that allowed the user to set the characteristics of the requirements for the concrete. The system then analyses the proposed recipes and yields a list of them ordered by the goal criterion, which in this simplified setting is the cost of the recipe.

The Mebis knowledge model was initially set-up to contain the knowledge to analyse one combination of ingredients for concrete to its properties such as cost, strength class and environmental class. This is also included to let the expert user validate recipe, which can be necessary as sometimes counter-intuitive recipes can arise. The system surprised the experts with some of the recipes that came up, which they would never have considered.



**Figure 6-32 Recipe Properties**

This system is being evaluated by the experts at this time with the intent to arrive at conclusions on the usability of such a system and possible improvements and enhancement that can be made.

### Knowledge System Development – Mebis-IO

The second phase of this project attempted to incorporate this model and stretch its capacities some more. In effect, it relocated the software solution that was required in the first phase to the model as well. This was necessary to allow consideration of logistical issues associated with production points and available stocks. The knowledge model incorporation into the system is expected in the next phase of its development project.

### 6.6.4 Conclusions

The Mebis system shows the ability of both KBE and IO as component knowledge system. Furthermore, it displays a very engineering based development and contrasts this with a more insight-based approach. In part, this is evidence of the ability to integrate the knowledge system in other systems, and displays the diversity of functions that a knowledge system can play.

#### Mebis-KBE

Mebis KBE showed a pure usage of the possibilities of the KBE, where the knowledge model was solve the problems by the interconnections between the parameters. The amount of imposed structure was kept to an absolute minimum. It also showed some of the limitations inherent in the usage of the KBE. The mono-representation limits what can be represented and how it is represented. Certain problems can also not be handled within the knowledge model, and are moved to source code solutions.



<i>People:</i>	Indirect knowledge acquisition	<i>Cost:</i>	-
<i>Project:</i>	Finite, product-based	<i>Risk:</i>	=
<i>Product:</i>	Integrated component for larger system	<i>Clarity:</i>	=
<i>Process:</i>	Incremental design and development	<i>Benefits:</i>	=
<i>Tools:</i>	KBE	<i>Bottom-line:</i>	+

**Table 6-11 Evaluation Mebis-IO**

Looking at the chosen approach for the development of the knowledge system and the acquisition of knowledge the following observations can be made. The experts involved are participating in a distant way, although they have been introduced to the knowledge model at times and have had an opportunity to criticise it. Using the textual way of representing the knowledge in a semi-formal manner allowed the knowledge engineer to concentrate on getting the different formulas right. The approach centred initially on getting the design right, and went on the assumption that the formulas found in this would be final.

The development of the knowledge system followed a similar path. No notion of completing a full cycle before going on was used. The development of the knowledge model was completed almost independently from the development of the knowledge system.

In actual fact, this case-study is the most like a standard knowledge engineering approach of all the case-studies. Beyond usage of the KBE's visual knowledge representation and facilities for development of the knowledge system, it incorporated very little of the continuous knowledge engineering approach.

### **Mebis-IO**

Beyond the change in tool in the second phase, not much changed in the approach taken. Therefore, the only changes that can be reported on are the effects of the differences in tool support.

<i>People:</i>	Indirect knowledge acquisition	<i>Cost:</i>	-
<i>Project:</i>	Finite, product-based	<i>Risk:</i>	=
<i>Product:</i>	Integrated component for larger system	<i>Clarity:</i>	=
<i>Process:</i>	Incremental design and development	<i>Benefits:</i>	=
<i>Tools:</i>	IO	<i>Bottom-line:</i>	+

**Table 6-12 Evaluation Mebis- IO**

The main differences therefore are the more explicit and more inclusive nature of the knowledge model. Furthermore, the models were better in their representation of the experts' concepts. While the participation followed the same pattern, it was augmented with UML models that have a close resemblance to the actual models in IO. It is possible to perceive this as a more direct use of the model, but the possibility to interact with the models was lacking.

In conclusion, this case-study shows how well possible it is to employ the benefits of the tools to a certain extent. The full benefits of the continuous knowledge engineering approach have however not been realised.

## 6.7 Conclusions

This chapter has shown a number of different cases applying the approach and tools described in the previous chapters.

<b>System</b>	<b>People</b>	<b>Project</b>	<b>Product</b>	<b>Process</b>	<b>Tools</b>
BOKS (legislation)	Participatory modelling	Finite, product- based	Dedicated knowledge system	Cyclic modelling and development, with limited deployed versions	KBE
MDDS (diagnosis)	Independent modelling and participatory development	Continuous, process- based	Knowledge model as scientific theory. Default and customised knowledge systems	Cyclic with many iterations, coupled with extensive verification, validation and testing	KBE
VDES (diagnosis)	Participation of expert and user on knowledge model.	Continuous, product- based	Default and customised knowledge systems. Knowledge model as scientific theory	Cyclic with many iterations, coupled with extensive verification, validation and testing	KBE
FRAS-KBE (legislation)	Indirect knowledge acquisition	Finite, product- based	Knowledge model as specification for software system	Incremental design, waterfall implementation followed by maintenance	KBE
FRAS-IO (legislation)	Participatory and partially independent modelling by expert	Continuous, process- based	Default knowledge system	Cyclic with limited number of iterations	IO
Mebis-KBE (configuration/ optimisation)	Indirect knowledge acquisition	Finite, product based	Integrated component for larger system	Incremental design and development	KBE
Mebis-IO (configuration/ optimisation)	Indirect knowledge acquisition	Finite, product based	Integrated component for larger system	Incremental design and development	IO

**Table 6-13 Relation of People, Project, Product, Process and Tool to Case Studies**

The five case studies cases come from everyday practice and vary quite significantly. Each has different goals, constraints and expectations. They have different kinds of

expert and user populations. To allow their analysis, each of the descriptions of the cases in the previous sections has provided a concise analysis of its results. These provide a summary of their features and help to find meaningful similarities and differences to be recognised. The goal is to bring all of these cases to a common ground. The results show evidence of the tool support for the development of knowledge systems, the ability to employ a continuous knowledge engineering approach as well as the benefits of the . To further structure that analysis, the results are collected in this section.

Table 6-13 shows a structured summary of the case-studies as to their position on the people, project, product, process, and tool dimensions. As can be discerned from the table the different cases show considerable diversity while retaining some important cross-similarities. This helps to draw conclusions of a more general nature about the different choices made in the various dimensions. While the applications take different positions, none belong purely on the side of the engineering or science metaphor. They incorporate measures from either side to a certain extent. It is also proof of the possibility to ‘mix-and-match’ aspects according to the resources and constraints that are determined by the practical reality of the situation.

System	Cost	Risk	Clarity	Benefits	Bottom-line
BOKS	-	-	++	++	++
MDDS	--	--	++	+	++
VDES	--	--	+	++	++
FRAS-KBE	-	=	=	=	=
FRAS-IO	-	-	+	+	+
Mebis-KBE	-	=	=	=	+
Mebis-IO	-	=	=	=	+

**Table 6-14 Bottom-line Results**

The above Table 6-14 shows the assessment of the case studies as to their bottom-line. This is an estimation of the systems developed compared to a standard development using run of the mill techniques. The systems show a mixture of results on these dimensions. Most systems show an improvement in their bottom-line. The least successful system is FRAS-KBE, attributed to its development process where the knowledge model is only used as a specification. The systems that appear to give the best results are BOKS, MDDS and VDES. They show the effects of the measures most positively.



# Chapter 7

## Evaluation

---

*If a man will begin with certainties, he will end in doubts;  
but if he will be content to begin with doubts, he will end in certainties.*

– Francis Bacon (1561-1626), *Advancement of Learning*

The evaluation is final step in the research program. Based on the earlier demonstration of the solution in the case studies, this chapter carries out this evaluation by performing the evaluation program specified in Chapter 3. In the first section, the criteria formulated as tests in the evaluation program are answered to provide clarity on each of the research dimensions: people, project, product, process, and tools. This will inform the discussion on the scientific and engineering metaphors. It will also shed some light on the two tool philosophies. The second section examines the practical economic results, the meaning of the solution in terms of clarity on benefits, cost, clarity and risk. The final section collates these results into a number of general conclusions.

### 7.1 Results of Criteria Evaluation

With the results of the case studies properly presented and arranged, it becomes possible to finalise the evaluation program. The results on the criteria of this program make it possible to reach a conclusion on the role of each of the dimensions of people, project, product, process and tools. The tool dimension is given detailed attention, to evaluate the tool philosophies. The results on all of the dimensions make it possible to form a conclusion about the continuous knowledge engineering approach and the role of the two metaphors.

#### 7.1.1 People

The engineering metaphor proposes that one of the reasons that knowledge acquisition is difficult is because experts are incapable of expressing their knowledge directly. The scientific perspective forwards that the experts are the most capable at extracting new insights from the knowledge modelling process. Furthermore, the continuous knowledge engineering approach pivots on the ability of the expert to participate in the knowledge modelling.

*People Test:* Ascertain whether experts are capable to participate in knowledge modelling or can independently model their own knowledge.

In Mebis, the expert is farthest from the knowledge model. The knowledge model derives from earlier textual formalisations made by the expert. The knowledge engineer translated the knowledge into a knowledge model, both in the KBE and in IO. During this time, inconsistencies noticed by the knowledge engineer lead to changes and discovery of new insights. These inconsistencies were those that would contradict earlier statements, incompleteness in the knowledge surfacing in the representation or errors that only surfaced during the operationalisation of the theory and application to test cases. Many of these are automatic for the visual knowledge

representations used, and are not domain-specific. Over time, the knowledge engineer has developed an appreciation for the domain itself, allowing some domain-specific insights to be derived. However, this contribution should not be overrated.

BOKS, FRAS and VDES show participation of the expert in developing the domain model. Instead of the knowledge engineer querying the expert based on general inconsistencies, the expert would examine the represented knowledge and initiate the need for changes. This requires the knowledge to remain closer to the original perception of the expert. The suggestions made by the experts in this case would often be constructive in nature, proposal to amend the model in specific ways. The automatic contributions of the representation only add to this.

MDDS shows experts modelling knowledge independently. In addition, FRAS-IO shows an experimental setting for independent knowledge modelling by an expert. In this setting the knowledge engineer, when present can make suggestions for change based on the same principles as earlier, but the main changes originate in the insights of the experts themselves, while searching for appropriate locations to position their newly found knowledge. In this setting, the representation in decision-tables often made the general completeness and correctness much easier to verify, negating to some degree the knowledge engineers need to be involved as evidenced by their very limited involvement in MDDS.

The applications have shown examples of systems developed by a knowledge engineer, by a knowledge engineer with participation of the expert, and independent modelling by the expert. The magnitude of insight and improved understanding of the domain is most pronounced when the involvement of the expert is a structural contribution and when it is more directly involved with the knowledge model. The experts involved have had ample experience using the systems. In time, they may therefore adopt enough knowledge engineering knowledge to become responsible for the knowledge model's further development. This also is evidence that there is no reason to assume that problems with knowledge acquisition are caused by the inability to articulate their own knowledge.

Therefore, the answer is that both participatory and independent modes are possible. In turn, this supports one of the basic foundations of continuous knowledge engineering. Its benefits on this singular aspect have already been shown. Additional evidence is required before this is conclusive, but there is for now no reason to assume that experts are incapable of modelling their own knowledge, given the proper tools.

### **7.1.2 Project**

The engineering metaphor assumes that a knowledge system project is finite, and similar in nature to a software engineering project. This includes the handing over of the knowledge system to another maintenance organisation after deployment (hand-over assumption). This assumption would become invalidated when most knowledge engineering projects showed signs of greater efforts after deployment of a knowledge system than before. Approaches based on the scientific metaphor claim that a knowledge system project is continuous as insight is acquired constantly throughout the development process, and requires constant knowledge acquisition. It is then not possible to hand a system over to a maintenance organisation after deployment. If successful knowledge systems exist that have a considerable maintenance history but

have seen little or no change during that time, this would seriously challenge the scientific metaphor.

*Project Test:* Ascertain whether a project knows more change than building activity.

BOKS shows a system that is considered finished after every deployment, but that has had three full deployments already. Further development is not required, besides obvious flaws and minor extensions. For now, maintenance seems to prove no threat for the system. The explanation for this lies in the assumed static nature of the legal knowledge it contains. The knowledge in BOKS as a carbon copy of the legislation will only have to change when the law itself changes. This requires a longer period to determine whether the knowledge does indeed remain static or whether alternative interpretations of the law require changes to the system, while the law remains constant. The case study has shown limited evidence of this already, but nothing conclusive. Nevertheless, even in BOKS, most decision tables have been changed at least once during their lifetime.

MDDS and VDES show evidence of a developing body of knowledge based on new insight discovered throughout the project. Here the knowledge truly has the character of a theory, as a representation of insights gained with an appealing form to the experts involved. This kind of project will perhaps reach a stage where further development and knowledge acquisition is smaller, by a limited number of the original team. However, this will only stop when the system is discontinued. These knowledge models are products in its own right, that warrant safeguarding. The systems also have an organisation that works to perform this safeguarding. These systems have also clearly shown more change than building activity. Here too, each decision-table underwent changes several times during the lifetime of the system.

FRAS in the KBE version, as a specification for a software system, is therefore oriented as a product. Again, the knowledge is based on legislation and this keeps the system more static. The maintenance problems show how hard the changes on knowledge are to incorporate in an implementation of the knowledge in the software system. As this is what is proposed as one of the avenues of knowledge engineering, this system provides ample warning of the possible consequences. The volume of change is too great for the approach to muster.

Mebis in both its incarnations shows another role for knowledge models as a component in a larger system. As a sub-system, the integration is paramount and the communication with a user directly is not included. Being required to follow the needs of the surrounding systems adds an additional reason for changes to the knowledge model.

For knowledge that has the character of a scientific theory character, the mode found in the latter systems is the most appropriate. The development of insight is necessary as there exists no formalised model and the model remains open for discussion. Unless a domain is defined to be static, the theory like character of the knowledge will therefore force a continuous need to keep the knowledge up-to-date. A factor that also appears to play a role is the fact that the knowledge system changes the environment into which it is introduced, leading to a further need for changes as a result of that feedback loop.

### 7.1.3 Product

Engineering approaches treat knowledge systems like any other engineering product. This product is an artefact: specified, designed, implemented, tested, deployed and maintained. The knowledge model is merely a part of the design specifications required for that system.

This presupposition would come under doubt if cases should show that the changes made to a knowledge system after deployment do not originate not in the discrepancies between the system and the specification, or from specifications deriving from requests for new features. Approaches based on the scientific metaphor perceive a knowledge system as a medium to communicate knowledge. This view of knowledge system would come under doubt if the changes to a knowledge system would be originating primarily in the errors and incomplete specifications, rather than through new discoveries and evolving knowledge invalidating part of the knowledge model.

*Product Test:* Determine whether the changes to a knowledge system originate in its environment or from the invalidation of the knowledge contained within.

In an engineering context, the knowledge acquisition is only in support of the quality of the eventual product. There is no incentive to improve the model because of its own quality as a model of the domain. BOKS, FRAS and Mebis each show this approach to knowledge acquisition.

The BOKS project treats the knowledge system as the singular product, also because in a client-developer relationship this is productive way of treating such a project. The functionality is directed to the users of the system, where the quality of the knowledge is supposed to be the exact same as that of the legislation. Bar errors, the knowledge in the model follows the further development of the law. Any discrepancies between the system and the law are therefore errors in the knowledge by definition. This fixates the knowledge and creates a different situation to the one found in the other systems.

FRAS-KBE with an explicit purpose as design specification shows that modifications are treated as originating in faulty behaviour from the users' point of view. Many of the changes are required in the model not in the system's usage of that model. The question on origin in this case is a matter of opinion. The model in itself is never used as a source of insight.

Mebis assumes to be a static translation of the specifications as found in the textual representation of knowledge. Changes to it occur, and are therefore places outside of the model. In effect, the knowledge changes through insight, leading to better formulations.

MDDS and VDES show systems where the knowledge model itself is the pivotal element. The ability to make sense of the domain using the model and expressing an understanding of the way things work is the principal goal. The knowledge system provides users with access to this operational theory of the domain and enables them to employ it in problem-solving tasks. The insight the model generates is the reason for the system to exist, and it only exists because it generated that insight. The insight drives the knowledge acquisition process.



Their knowledge engineers and developers often saw the systems that were developed as a product. Nevertheless, in a few, the product was most definitely the knowledge model, and the creator of the insights that were gained.

#### 7.1.4 Process

The engineering metaphor assumes that the incremental development plays only a role in risk management, but is not necessary property of knowledge acquisition. This means that in this viewpoint, the incremental approach functions purely as a divide-and-conquer strategy. Incremental development approaches, whether they focus on incremental methods within phases or whether they operate over full cycles of development should make no difference. Additionally, the engineering metaphor is design-focused. It therefore sees the design as the primary phase to be approached in an incremental fashion. It does not perceive applications of the design as an opportunity to test the system for errors and incompleteness, beyond validating the system's conformance to the design. Counter this, the scientific metaphor envisages that cycles of successive adjustments coupled with critical review form an essential ingredient in the discovery process of knowledge acquisition.

If the case-studies would show that better results are reached through cyclic rather than incremental approaches this would put the engineering assumptions in peril. Furthermore, if it could be shown that design is not only important phase to approach in an incremental or cyclic fashion, this would also place question marks with those assumptions.

*Process Test 1:* Establish whether some phases are more important than others are or whether they are equally important.

Posed alongside these assumptions are the notions stemming from the science metaphor that the complete cycles matter more, than others are. Therefore, the discovery of knowledge depends on having operational models at every moment in time.

*Process Test 2:* Establish whether cyclic development is more important than incremental division (the latter means executing phases incrementally).

BOKS and Mebis have seen a small number of large-scale cycles, by their version-based approach to development. They are more in line with the first category with the modelling itself performed in an incremental manner, rather than through complete incremental cycles.

MDDS and VDES have seen the highest incrementality by the use of small cycles, sufficiently to warrant the term continuous development. This was an important factor in their development as this allowed the changes to set and used on practical cases to test them before implementing new changes.

The FRAS-KBE maintenance history is extensive but its history is clouded by the fact that it was used to specify a software system. Its specification nature places it within the category of incremental design focused systems. FRAS-IO belongs in the cyclic systems, but as a knowledge system it is too small in scale and has too little actual development to truly comment on this aspect of development.

It is hard to disentangle the incrementality from the other dimensions examined, as all projects had some form of iterative process. The cyclic form of incremental development combined with expert modelling seemed to provide more of a basis for

the development of insight than solely a lower number of cycles, incremental design, Also, participation of the expert or knowledge engineer modelling by itself would seem to have less of an impact in such a situation. Cyclic development seems therefore to be highly connected to the insight creation derived from knowledge modelling. However, the use of these tools also biases this, as both the KBE and IO allow consultation of the models during the modelling. Nevertheless, the cyclic development seems a highly important aspect of knowledge modelling, as it allowed the early finalisation of a single, small-scale change and incremental testing.

The application of the knowledge system generated less pronounced effects on the creation of insight than the participation of the expert. The applications that they examined were test cases or 'cornerstone' cases used to test the system. Conversely, they would specify a specific problem-solution combination that the system was unable to handle before. These would sometimes be provided by users of the systems, when such an incompleteness was detected. However, the user base of these systems was much smaller and the feedback from application was therefore not as strong as it might have been. However, the feedback from BOKS and the effect this has had on the development was only limited. Although it must be said that some feedback was indeed incorporated back into the system. The nature of the BOKS system however means that this was seen as an error that was reported, rather than an opportunity for new insight to be reached.

In conclusion, application experience and cyclic development have an important part in generating the required insight. Something that is not possible with an incremental design approach. Smaller complete cycles are the best basis for building and developing a knowledge model.

### 7.1.5 Synergy

An important question is the manner in which synergetic effects influence the effects of the measures that were taken. The engineering metaphor naturally makes no statement about any such synergetic effects. The approach based on the scientific metaphor goes on the assumption that each insight is an enabler for other insights. Thus, the total amount of support is indifferent to the specifics of the measures taken, rather the package as whole can be seen as a measure of insight support, and each of these strengthens the others.

*Synergy Test 1:* Determine to what extent the measures on the different dimensions are necessary.

*Synergy Test 2:* Determine to what extent the measures are influencing one another.

The synergy between the guidelines and the different results for each of the dimensions is clear. It is not necessary to address measures in each of the dimensions in unison, as insight will be created even when only one of them is addressed, for example incrementality. Nevertheless, it is clear that the effects will strengthen each other, validating the assumption made by the scientific metaphor. The most pronounced effects are visible when incrementality and expert modelling combine.

### 7.1.6 Tools

Tools enable and support the proposed changes in process, product, project and people that may be necessary under the science metaphor. In a manner of speaking, tools embody the approach and make it practical and productive. They often

determine the practical reality of any methodology: what parts of the methodology are used in what manner, by whom and to what end.

The tool support given for the continuous knowledge engineering approach aims to perform two tasks. First, it aims to make continuous knowledge engineering viable. Secondly, it provides some clarity about the methods best employed to realise this.

While the tools in this thesis are directed towards realisation of an approach based on the science metaphor they employ a pragmatic view, focusing on direct support for building, changing knowledge models, and gaining insight through modelling and application experiences. This also means that they are still usable in an engineering perspective without any significant problems. The case studies illustrate this. The system do however realize a number of the requirements originating in the alternative metaphor by incorporating different features thought to be useful in this respect.

The tools must support the development of insight. Allowing experts to model the knowledge themselves, enabling them to understand a model and the effects of changes to that model is only a part of this approach. Extracting meaningful knowledge through practical experiences of the knowledge model, embodied in a knowledge system is another. In addition to this the tools must enable incremental development of the knowledge model and knowledge system. This must be done so both small and large increments can be fashioned and made available to the users. Finally, the knowledge systems have to be adaptable to the needs of the user, in a cost-effective, flexible and incremental manner.

Rather than directly provide arguments for and counter the metaphors, these tools seek to support these requirements. Their purpose is not limited to providing evidence in support of the approach, and to enable the comparison of different approaches to development. For example, a number of the capabilities of these systems feature in other knowledge system development environments not specifically built for this purpose.

*Tool Test 1:* A tool that can allow knowledge modelling by experts must be simple.

*Tool Test 2:* A tool that can allow knowledge modelling by experts must use a vivid modelling language, to aid the discovery of new insight and support the changeability of knowledge models by experts.

Either tool described in this thesis implements a philosophy on the way the support should be given. The respective tests above represent the two philosophies under investigation. The first states that such a system must primarily offer simple mechanisms, 'dumb down' the possibilities that may exist and thereby making it easier for the expert to understand what is going on in the model.

The second assumes that although simplicity of use is important, the ability to generate insight in every way possible is most important. This can be supported by a vivid form of representation, which is in this case the principal objective.

Examining the results of both these tools then aims to yield some answers as to which tool philosophy is more successful in providing the conditions most suitable for the discovery of insight, and support the continuous knowledge engineering approach.

## **KBE**

The KBE is a tool that implements a set of features deriving from the requirements formulated in Chapter 3. The KBE focuses on providing an easy to use system to

allow non-specialists to use it to create, modify, and extend a knowledge model. By providing a simple tool, it was thought that no overwhelming learning curve was presented. The simplicity would allow navigation of the knowledge models, as well as facilitate understanding of the effects of changes.

The graphical knowledge representation of the decision table makes it easy to comprehend, validate and change the knowledge about the simple facts in the knowledge base. The primitive facts that are at the disposal of the modeller leave little room for complex design decisions in the description of the world. The straightforwardness of the inference engine also contributes to the ease of use of the system. The backward-chaining inference mechanism strategy is fixed and can only be adapted in minor ways.

The minimum system is tiny, consisting of a single parameter. This means that as soon as the user has entered the first fact, the system can be consulted. Adding new parameters is relatively easy and adding new knowledge without necessarily affecting other parts of the knowledge base is possible. The system remains to be consultable after each change. This allows direct testing of each change, and means that in a sense every change and subsequent test constitutes a development cycle.

The facilities that are shared with IO for developing knowledge systems based on these knowledge models allow the development of the visualization of the system to be as advanced as necessary, and can also be implemented incrementally. These facilities for developing and deploying knowledge models range from small to medium sized knowledge systems. The simplicity supports this initial trajectory and allows an incremental way of building knowledge systems, either by small or by larger granularity increments. The previous chapter has shown examples of these abilities. As the knowledge model grows the ability to change the model diminishes. The ability to comprehend the effects of changes to the knowledge becomes harder and harder. These shortcomings of the editor originate directly in its simplicity. The representational poverty of both factual and reasoning knowledge creates problems in the modelling of knowledge. The structures and relationships that are available in the domain cannot be modelled directly. This requires that much of the expert conceptualisation of the domain needs to be translated to the chosen knowledge representation formats. Furthermore, it means that problems like those that require reasoning over instances can only be accomplished in part and this requires redundancy in the knowledge base.

The KBE therefore does not have scalability of change, when knowledge models accrete during their lifetime. The lack of structuring mechanisms makes a knowledge base hard to navigate although some support can be given by provision of a tree of the decision tables. Making changes to the system also becomes increasingly difficult as the locations of the affected areas become harder to find.

Additionally, there are no facilities for decomposition and encapsulation. This means that changes can cascade through the model. The representational poverty also means that the decision-table is used as an all-purpose tool. This leads to further cluttering of the knowledge base. As much as 60% of a knowledge base can consist of structuring tables. Combined with a limited amount of vividness of the model and navigation through related areas of the model this wreaks havoc on any maintenance and evolution. In effect, this means that there is a ceiling for the size of a knowledge system.

The limitations of the modelling environment in its primitive domain model and singular knowledge representation in the form of decision tables do not allow the representation of task model or modelling of concepts and relationships as they play a role in the domain. The latter is a fundamental problem of the representation. Problems that contain variable number of elements, as many do, are impossible to solve given the chosen representation form. Approaches to circumvent these problems in the KBE lead to further maintenance problems.

In conclusion, the KBE is easy to use and the facilities to support the non-specialist user are definitely present. Nevertheless, these features alone are not sufficient to support the user in the end. In particular, the evolution of knowledge models is undermined by the lack of structure, impeding changeability and other long-term requirements of an evolving knowledge model.

### **Intelligent Objects**

IO realises a philosophy, which embraces vivid knowledge representation as a means to support the discovery of new insights. Vivid models can stay close to the experts' understanding of the domain, providing every opportunity to be explicit in the formulation of that knowledge. This allows reality to inform the knowledge model and vice versa. It is thought to make it easier to find inconsistencies and incompleteness in the model, because more of the knowledge is made explicit. To realize this philosophy IO incorporates a number of the features present in the KBE capitalising on its strengths while attempting at the same time to improve upon some of the weaknesses of that system. The philosophy that underlies IO redeploys the features in combination with additional features. The primary difference between the KBE and IO are the facilities for expressing knowledge. They are greatly expanded both in the structural knowledge as in the behavioural part.

Therefore, IO focuses on providing a vivid modelling language to express the knowledge, based on object oriented modelling constructs. The complete realisation of UML modelling constructs, especially the inclusion of the association, allow the user to model the domain knowledge at a conceptual level. The object-oriented model conforms to the understanding that many have of their domain, and this allows them to map new insights within the model to the real world and vice versa. This makes it easier to connect new insights either derived from the expert own experiences, insights in conceiving the model and exploring different possible formulations of knowledge within the context of an existing model. The modelling language serves other purposes such as the navigation of the knowledge models, the facilitation of understanding of the effects of changes and support for continuous changeability through modularity, encapsulation and abstraction.

The knowledge model further has features to aid the modeller in understanding the effects of actions on the model and to remove the need for transformation to an executable format. By virtue of its live model, the user can examine the effects each operation on the model and ascertain the correctness of elements. This guarantees feedback on each operation immediately and throughout the model. Furthermore, this guarantees that the model does not require any transformation for implementation in some other language. Models retain the capability to be executed at any point during their development.

Additionally, the system's architecture performs basic but essential functionality to support non-specialist users, such as undo/redo mechanisms and multiple view user-

interfaces. The IO environment further retains the visual representations of knowledge such as the decision-table and shares the same approach to knowledge system deployment and the components that allow the development of default, custom and dedicated systems to in an incremental manner.

Therefore, IO provides a development environment more advanced in modelling vocabulary and additional facilities that it offers in the feedback from the model and the undo/redo support. It provides an architecture that grants many different knowledge representations. Additionally it aims to provide the different elements of the model in a form and presentation that is easy to understand and is conceptual in nature, coupled with visual representations of reasoning knowledge. Thereby it provides a more explicit knowledge model, building on the additional structure and diversity, provide sufficient flexibility in representation to allow vivid models to be created that conform to the expert's perception of the environment.

The minimal knowledge model consists of a declaration of some type and an instance of the project. Compared to the minimal system of the KBE this is already an indication that operating this system is more difficult. For one, the distinction between a type and an instance level is sometimes hard to make. For some types of reasoning, this may not even have to be a useful distinction.

IO focuses on the diversity of representational facilities and proposes some features to strengthen the ease of use of the system. In short, it aims to improve the evolution of knowledge systems, while retaining simplicity of use. Even so its learning curve is much more steep that that of the KBE. IO suffers from the opposite problem to the KBE. Here it is the complexity, introduced by new options and possibilities that enable knowledge model evolution, that make it a difficult system to use for any non-specialist. The choice introduced requires understanding the options and making a choice. This may not be the user's strong suit. Furthermore, more steps are required in unison to implement particular changes. Changes made in the KBE are often monolithic and far more direct.

The problems that surface in larger models in IO relate to the increasing difficulty with which changes are possible. This is isomorphic to the problems in the KBE, but in IO, they occur much later. Some of this is moderated by the compartmentalisation and encapsulation, but this does not remove the problems completely. The structure that has been placed in the domain model is now explicit but at some level, it becomes a fixed bias and can be hard to change while maintaining the integrity of the model.

A conclusion for the tools therefore is that the IO models are more resilient to change, are more understandable and allows a high level of insight. This allows better support for experts especially when the model grows in size. The higher learning curve does make it more difficult for experts to make the first step. The KBE makes it very easy to build a minimal system and build from this. It does not have any facilities to deal with systems that are more complex. IO has a more difficult initial development, and requires additional work for systems that are very easy in the KBE. IO does handle those systems with ease, which remains impossible in the KBE.

### **7.1.7 Conclusions**

Two distinct levels of conclusions can be derived from this evaluation of the criteria. The first is the evaluation of the tools, and the second the evaluation of the continuous knowledge engineering approach.

### **Knowledge System Development Tools**

The models developed using the KBE all have a large amount of implicit structure. It is necessary to know this structure to be able to understand the knowledge model. In many cases this structure develops during the first phases of the project. Sometimes this leads to the abandonment of a first model, to be able to restructure and improve the system while scuttling the existing knowledge model is still a viable option. The structure is implicit, but it is determined by re-examining the order in which the tables are called in their execution hierarchy. In many cases, the structures that are used also provide an explicit basis for incremental development, by creating extension points.

The experience so far with IO makes it hard to make any final conclusions, although the case-studies show several key-points about the system. IO is a system that can be used by an expert. Furthermore, the knowledge models that are constructed with it contain much more explicit knowledge than is present in similar KBE knowledge models. This makes the internal relationship in the knowledge model much clearer, and further allows them to be subject of explicit discussion. This also is helpful in communicating the knowledge to others better, which aids cooperation with a knowledge engineer or expert.

Further research would be to allow a KBE like knowledge model to slowly transform into an IO knowledge base. Enhancing the instance level model with abilities to define decision tables of their own could do this. An instance of one particular room found in the model could then slowly transform into the type for all rooms.

### **Continuous Knowledge Engineering**

Overall, the previous brings the principles of continuous knowledge engineering in view in different settings, even though none of the systems explicitly employed the development process. It does provide ample evidence that each the principles of the continuous knowledge engineering each is at the very least possible. The case-studies further show that the advantages thought to exist when employing these principles are realised.

The evaluation of the different criteria through the tests is successful in many aspects of the continuous knowledge engineering approach and thereby the scientific metaphor. While this is not intended to be taken as proof that the science metaphor is somehow better than the engineering metaphor, it makes clear that providing for insight development is an essential factor in knowledge acquisition. Knowledge engineering must not just account for how knowledge systems should be built but also account for their development and change over time.

## **7.2 Evaluation of solutions**

The research in this thesis started with the formulation of a number of problems of an economic nature that made it difficult to offer knowledge systems as a viable solution to organisations. The problems are in turn caused by a number of root scientific problems in knowledge engineering. This section will first consider the solution that this thesis offers for the latter scientific problems, before going into the evaluation of the solution to the practical problems.

### 7.2.1 Scientific problems

The four main problems in knowledge engineering affecting the ability to offer knowledge systems as an economically sound solution are:

- The knowledge acquisition and maintenance problem
- The gap problem
- The brittleness problem
- The complexity problem

The direction of the solution on the level of metaphors attacked these problems on all fronts, making each of these problems of paramount importance within the approach taken. The previous section dealt mainly with the individual parts of the approach, dissected into different dimensions. The aim of that section was mainly to answer the question whether the continuous knowledge engineering approach was viable and practical. That has been established, as well as the role of insight in knowledge system development. The question that remains is whether the solution actually solved the problems that were identified in the first place.

#### **Knowledge Acquisition and Maintenance Problem**

The knowledge acquisition and maintenance problem is one of the main problems in knowledge engineering since its inception. Knowledge acquisition is said to be an intrinsic problem of knowledge engineering, making the development of knowledge models difficult and hard. It is an important factor adding to the risk and cost of knowledge system development.

It already turned out to be impossible to extract knowledge from an expert's brain directly, by asking them for the rules that they themselves use. Subsequently, the modus operandi changed to a modelling or design process. Modelling this knowledge requires monitoring and analysing the expert behaviour. The expert's knowledge is incorporated into a model of the domain and the task by a knowledge engineer, without having anything to do with the actual knowledge that may be present in the expert's head. The expert is not trusted to do this himself because of his supposed inability to formulate and formalize the knowledge correctly.

The solution proposed in this thesis, reformulates the problem. In the normal view on knowledge engineering, knowledge acquisition is perceived as a requirement for knowledge engineering, enabling the development of the end-product of this activity, the knowledge system. Conversely, this thesis treats knowledge acquisition as the goal of knowledge engineering. Within this view, the knowledge model is the end result. The knowledge system 'merely' functions as a medium for the knowledge model, supporting the knowledge acquisition process by providing actual experiences and feedback.

Although this may seem an artificial, almost deceptive, solution to the problem, it has provided a path to the creation of an approach for knowledge system development without suffering from the knowledge acquisition bottleneck. In analogy to software engineering, (Brooks 1987), programming is not about formulation and formalisation of knowledge like programming syntactically correct structures; it concerns construction of complex conceptual arrangements. In knowledge system development, the actual problem is not formulating and formalising knowledge to be represented in a knowledge system; it is the discovery of the knowledge in the first place.



To support these statements, the case studies have shown the possibility and practicality of such an approach to knowledge engineering. Treating knowledge acquisition as a goal for knowledge engineering, changes the way knowledge systems are developed. This alternative approach, by virtue of its gradual character transforms knowledge engineering from the development of a product into a path of discovery. Experts have shown to be quite capable to develop knowledge models, and moreover, are more in tune with their domain. Their participation, continuous nature of the project, the view of knowledge systems as a medium and a cyclic process for development, as well as tool that support this and the discovery of insight, facilitate the discovery. The knowledge acquisition and maintenance bottleneck is solved, because using this approach, it is a virtually effortless activity.

### **Gap Problem**

The engineering view perceives the development of a knowledge system based on a knowledge model as a technical issue. Its scientific relevance is limited, and therefore it relegates the problems either to the tool producers or more generally to software engineering. The long period before a knowledge system is complete, in a normal development, complicates the realisation of a learning process. It increases the distance between the knowledge model on which the system is based, and the actual experiences that might be gained by the application of that model.

In its role as a medium, the knowledge system acquires another additional function. The system turns into one the sources of insight. The cases to which it is applied even as test cases, fall into the category of application experiences. Insight discovery is supported by critical feedback, which can come from the users of the system.

The research in this thesis shows knowledge systems as essential in generating feedback from the users of the knowledge system, both in terms of insight as in the ability to support the users in their tasks. In most cases, these are one and the same. Knowledge systems have to properly solve the problems of its users. Facilitating the understanding and ease of use of a system through an intuitive user interface, and integration with auxiliary facilities, is an important aspect of insight development. Both the problems they are used to solve and their inability to solve certain cases in a source of new knowledge.

The realisation of the knowledge system based on the knowledge model is pulled to the front of the development. This is supported by the tools and allows the development of default, customised and dedicated user-interfaces. The case studies have shown examples of industrial strength knowledge systems, and have shown the process of their development. Although these by necessity originate in the tools, the facilities are in support of the approach. This is also the reason that these tools are shared between the KBE and IO.

Attention is given to the development of the system in a form presentable to the user from the first modelling action. With the ability to incrementally develop this knowledge system based on default consultation environments through to customised systems, and up to dedicated systems, the effort can be spread over time and directed towards the wishes of the user. The technical capabilities and the incremental development approach bridge the gap between a knowledge model and a knowledge system.

### **Brittleness Problem**

The brittleness problem as the inability of a knowledge system to perform beyond the knowledge it possesses is within the engineering view a problem that can only be solved by deep knowledge models and/or a store of commonsense knowledge. These can provide the depth required to function acceptably beyond the domain knowledge of the system.

Seen from the perspective of the scientific metaphor and the continuous knowledge engineering approach, this problem is artificial. To expect a system to work on outputs that it was not built to work for gives the same results as when a scientific theory is used to predict values for which it was not created. Newtonian physics fail when stressed to its limits. Parts of this failure has led to the theory of general relativity and the discovery of quantum effects.

Two answers to the brittleness problem derive from his. First, the brittleness problem is a problem with the boundaries of knowledge. When a problem is posed to a knowledge system for which it should be able to provide an answer is a possibility for learning. The growth of knowledge that results from this insight is testament to the nature of knowledge.

Second, many engineers still use Newtonian physics even though better theories exist. For their purposes it suffices to use these laws, and this will not adversely affect their ability to develop new devices, as long as they remain aware of the underlying assumptions. Similarly, a knowledge system may operate under the assumption that its task is limited to a certain subset of all possible inputs it could theoretically be subjected to. If these implicit assumptions are violated, the possibility is created to become aware of such an assumption. It can then be explicitly modelled into the knowledge model.

### **Complexity Problem**

Many knowledge systems are developed for tasks that would require humans training and experience to solve. Dubbed the complexity problem this is an inherent problem of knowledge system development. The task to perform within domain of the system does not become any simpler merely because of the introduction of a knowledge system. While new discoveries may go some way to increase understanding of the domain and simplify performing certain tasks, this can not be interpreted as a solution to this problem.

For those knowledge systems that orient themselves to good performance in non-linear domains, with no deterministic procedures to calculate a solution, the continuous knowledge engineering approach has merely provided another way of ‘... making the hard possible’. The base complexity of knowledge system development for complex tasks remains however.

## **7.2.2 Practical Problems**

This part of the chapter concerns the evaluation of the results related to the bottom-line problem associated with benefits, cost and risk of knowledge system development. The practical problem for which a possible solution was sought, lead to the scientific questions answered above. The question addressed here is whether the global solution, proposed as the scientific metaphor and its realisation in the

continuous knowledge engineering approach shown in the case studies, solves the economic, practical bottom-line problem.

It was determined that an incremental approach inspired by the engineering metaphor was in principle a sound approach to overcome the difficulties of knowledge system development that lead to the uncertain bottom-line. The benefits of the approach depend on the granularity of incrementality, the magnitude and cost of each increment and the risk of each increment. It does not address the problems that cause the cost and risk.

The insight-based metaphor partially extends incrementality proposed by the engineering approaches. The questions that then remain are whether an insight-based approach is an actual improvement. Does it make the benefits clearer, lower the cost and risk of development, and provide additional control? This section attempts to concentrate on these issues to provide a validation of the results of this thesis based not on its scientific rationale, but on its actual effect on that which instigated the research in the first place.

### **Benefits**

The cases studies show a great diversity, but the actual benefits of developing a knowledge system remain hard to ascertain on face value. In some cases the benefits can be approximated, for instance with BOKS where the value of the knowledge system can be offset against the cost of having to train thousands of people in applying the legislation. In MDDS, there is no actual knowledge system as a product with a large user base; the discovery, formulation and formalisation of the knowledge is the actual benefit. This intangible can be attributed a value in the same way a value can be assigned to a monument; either as a priceless human creation or as stone rubble, as a matter of opinion. When it can be employed in a number of cases, its worth may become more evident as it then becomes based on its current utility. Therefore, the benefits should be seen as the commitment that can be found to continue the system's upkeep.

As the insight-based approach is focused on creating a usable knowledge system or operational knowledge model as early on as possible, it allows this utility based approach to be used with more force. There should be no large initial realisation, and each increment should be as small as possible, both of which are supported by the metaphor and visible in its realisations.

A problem found with small initial knowledge model is that a knowledge system based upon it may not provide enough content to be usable. A regularly quoted engineering principle is the 80-20 rule (otherwise known as Pareto's principle). This heuristic states that the common category of problems takes a small amount of time to solve, but that the remaining, often exceptional problems take much, much longer. This principle applied by the science metaphor. The case studies show that the usability of a first system can be quite impressive with only a limited amount of knowledge, acquired with minimal effort. The problem the knowledge system solves is often not the lack of knowledge, but its availability to others. Therefore, in most situations incomplete knowledge models can enable a functional knowledge system that can be ascertained to its utility in an early stage. Furthermore, its role in providing a stepping-stone in further knowledge acquisition has shown to be especially beneficial.

## Cost

An additional factor in determining the possible bottom-line is the cost, although the actual cost of a project will be a function of the effort of each step and the risks that are relevant to the success of those efforts. Therefore, these paragraphs will examine the *effort* necessary to perform certain tasks; the relevant risks will be discussed later. The effort of each of the developments over for the majority of cases remains to be an involved enterprise when taking into account the complete lifecycle. The continuous nature of the development and the smaller increments allow more even spreading of the cost over the complete lifecycle and strengthen the benefits derived from the incrementality.

However, there is also reason to assume that the approach lowered the base cost. The reduction of the knowledge modelling effort and the development of the knowledge systems visualisation are the most effective targets, as they concertedly form approx. 80% of the total cost. This is also visible in the case studies. In all of the cases the acquisition and maintenance of knowledge was one of the major tasks in the development. In those systems with a proper user community the effort for development of the outer shell of the system was an activity that was at least as great in size as that of knowledge acquisition.

The cost of knowledge acquisition is much lower than in standard approaches for three main reasons: business focus, deployment of people, improved changeability.

The insight-based science metaphor forces the knowledge acquisition to be focused on the model itself. It does not require the use of 'circumstantial' knowledge gathering, like interviews and repertory grids, in fact it de-emphasises from this entirely. This is also clear from the case studies. The amount of work to get to a unit of knowledge into the model is therefore reduced. Furthermore, the model focuses the knowledge acquisition to the system that will be used rather than unearth the full knowledge and then derive an operational version from that. The sole aim is to create an operational knowledge model for a specific task and user population.

Under an engineering metaphor, the role that a knowledge engineer plays as a scribe for the knowledge model in a knowledge acquisition process is necessary to create a part of the specification for a system. Within the science metaphor, such an approach becomes nonsensical, as it would require the knowledge engineer to be permanently attached to the project, part of the upkeep organisation. Transferring some of these tasks to the expert serves as an important cost saver. Especially when the knowledge engineer can be completely removed from the process, this is a very effective approach to reduce the cost, as is evidenced by MDDS.

Much energy has been directed to reducing the effort spent making changes to the knowledge model, and to reduce the effort required to incorporate these changes into the knowledge system as used by its users. Modifying and extending a model was a special issue for both the KBE and IO, and this has been a subject of discussion in the description of the case studies. Furthermore, deployment of the model has also been an area of attention.

The detachment of a knowledge model and the knowledge system's visualisation also means that one change can affect many knowledge systems, and that a change in a single representation can be limited to that particular representation. This lowers the cost of changing the visualisation of a knowledge system while increasing the benefit of keeping the knowledge model up to date.

Next to this transfer cost, other costs in this development are related to the level of interactivity, graphical user-interfaces, and integration with other systems. These costs are determined mainly by the support that is present in the tools that are used, but also are affected by the timescale in which these must be implemented. A more gradual scheme can allow piecemeal realisation, rather than require a big bang approach, which when it fails is ever so costly.

The inclusion of facilities for knowledge system development, in an incremental fashion, does also contribute to reduction of development cost. Less effort is necessary to attain proper levels of user support and understandability, and more effort can be focused on those areas that require change from the perspective of the users. Because of these factors, it is thought that the development of the example knowledge systems has been lower than would otherwise have been the case.

### **Risk**

The risk is a factor in determining the proposed bottom-line for any knowledge system development. Knowledge engineering knows many risks, which differ in situation to situation in importance. The approaches inspired by the engineering metaphor use iterative, incremental development to overcome and lessen the risk. The same approach is used here, although with a different aim. The benefits with respect to the risk however remain to be the same. Besides this general approach to the risk, some specific amendments exist. The question here is whether the continuous knowledge engineering approach, embodying a science metaphor, realised in the knowledge systems of chapter five offers less risk than the standard approach. Two factors are important when determining this: does the risk concern a smaller cost, i.e. reduce the effect of the risk, and is the risk itself reduced.

Smaller increments provided by the tools, are less costly to reverse, and the inherent support for change allows for a decrease in rework effort required for a change. Therefore, any support for smaller increments and added changeability reduces the risk taken.

Further specific risk reduction effects are also supposed to occur. The expert is sometimes hard to motivate to participate and remain with the process, as fears of being made redundant play a role. The approach shows reduction of risk by motivating the expert to assert ownership of the knowledge system. This improves their bond with such projects and makes it much easier to form an organisation for the management of the system. The insight-based approach makes the knowledge system an extension of the expert, rather than its replacement. If the expert does not continue to feed the knowledge system, it will soon be out of date.

A risk that is not easy to take away is the inability of experts in operating the knowledge system development environment. It is possible to go quite a way in making such systems more useable, but it would not be correct to expect everybody to be able to use them. The case studies and other experiences have however shown that experts with little or no computer skills can effectively make use of KBE and IO. As this risk is made obvious early on in the project, the effects of this risk can be limited. Furthermore, a participatory approach to knowledge modelling can be adopted with lesser but comparable benefits.

The expert may be capable of certain tasks that cannot be translated to a knowledge system. This can be a risk for any knowledge system development. For instance, the system can require the sensory perceptions of the expert, i.e. a tactile rather than a

cerebral task. In certain of the case studies, such issues have surfaced, for example visual inspection of damage in MDDS. Such problems can be circumvented as with the damage catalogue in MDDS, but one cannot suppose this is always possible. Therefore, this risk may be lowered, but is not removed. It is possible that some of these problems will not ever be solvable within the limited context of knowledge systems. Unless of course the notion of a knowledge system is stretched beyond recognition.

A further risk may be that the knowledge system requires a large amount of technical sophistication, e.g. high levels of interactivity, graphical user-interfaces, and integration with other systems. This type of risk is addressed by the tools that were proposed and receives special attention. The knowledge systems show industrial strength applications that can deliver the same types of user interfaces that other software shows nowadays. Integration of the knowledge system in other systems and with other system has also been shown. These were all realised without specific problems, and therefore pose as the same type of problems as they do for standard software development.

### **The Bottom-Line**

As the benefits can be made more clear, the cost can be spread more effectively and is lowered, the risks have been reduced as well as the effects of the risk, this means that the approach proposed in this thesis has realised significant advances in this respect with respect to the bottom-line problem.

The difference then is the following and in this difference lays the solution. The cases show systems that do not attempt to 'get it all in one go'. They attempt to develop a first model of development knowing that it is not complete and correct, consistent and provable. But rather than attempt to get it right before the product is placed in the hand of the user, the deployment is seen as one of the means to get it right. This change in perspective is the actual solution.

When employed the metaphor and its practical implementation in the continuous knowledge engineering approach can alleviate the knowledge acquisition bottleneck to the point that is no longer is a relevant parameter. The focus on building and changing knowledge systems resolves some of the intrinsic difficulty of formulating and formalizing knowledge. The appreciation of the nature and origin of knowledge allows for realisation of the manner in which knowledge systems should be engineered. In addition, the types of applications that can be created by applying the approach and the supporting tools can clearly generate professional, industrial strength knowledge systems.

In addition, the approach can be employed in situations where knowledge is yet to be discovered, to structure and support the insight creating process and knowledge creation. This can be a powerful incentive to companies to develop and employ knowledge technology in their organisations, for instance in research organisations.

## **7.3 Conclusions**

The evaluation presented in this chapter completes the research program. What remains is for these results to be reflected upon to reach some more general conclusions.

The basic question asked at the beginning of this thesis was why knowledge engineering is so difficult. The global solution that was found started from an understanding that the knowledge system is a special kind of engineering product, that remains fluid during its lifetime. It therefore requires other types of engineering practices than most. The engineering metaphor that underlies much of knowledge engineering needed to be exposed as the common denominator behind the current state-of-the-art in knowledge engineering. Solutions stemming from pure engineering inspiration are not sufficient and risk focussing on inappropriate directions, misinformed by initial explanations of the problems.

When examining the different dimensions that were looked into another perspective is afforded. The people dimension makes clear that there is no impediment for expert to play a role in knowledge modelling, to a far greater degree than is the custom or prescribed by accepted methodology. In fact, the benefits of this approach are quite substantial. The project dimension shows the continuous nature of development and the requirement of most knowledge system to remain in development. The product dimension shows that the results from a project are not limited to the knowledge system. The formulation and formalisation of knowledge through the discovery of insight delivers an important product in an explicated, knowledge model. The process dimension makes clear that mere incrementality is a valid and sound engineering approach directed to divide-and-conquer of risk and cost. For insight discovery, it is necessary to propose-and-critique. The tool dimension shows several things. First, the simplicity is not necessarily the best approach, and vividness can be a source of additional insight. This does not require deeper knowledge models that are more resilient.

Each of these dimensions individually shows the requirement to focus on the development of insight, which is a cycle of proposal and critical assessment. Both in the modelling and in the application of knowledge systems can such feedback be found. The obstacles professed by the engineering camp do not exist in these cases. Rather than do away with the engineering inspired aspects of knowledge engineering, it attenuates the need to appreciate the development based on a learning developing process. It strengthens what is particular about knowledge systems instead of focusing on the relationship with general engineering practice. The dimensions show the ways that a knowledge system differs from other software development.

By merely changing one's point of view on developing artefacts such as knowledge systems, different explanations are afforded for the problems that are experienced, inspiring alternative approaches to solve these problems. That insightful action was provided by the engineering metaphor itself, as the stepping-stone for this change of opinion.

The scientific metaphor has provided the starting point for a continuous approach to knowledge engineering, supported by dedicated tools. These have shown to alleviate the acquisition of knowledge and support the development of industrial strength knowledge systems. The cases show the beneficial effect this has on the cost and risk of knowledge system development, and the clarity on the benefits of a knowledge system.





# Chapter 8

## Conclusions and Recommendations

---

*Perfection is attained by small degrees;  
it requires the hand of time.*

*Voltaire*

This chapter summarises the results of the research program set out in the beginning. In the first section, the results obtained at the previous theoretical and engineering steps are analysed and held against the research criteria, based on which a number of final conclusions are drawn. The second section makes a number of recommendations for further research and development. The third section finalises by presenting a short summary of the contributions of the research.

### 8.1 Conclusions

The incentive for this research was a practical, economic problem with the use of knowledge systems as a viable technology to employ to solve knowledge problems. This bottom-line problem reflects the unclear benefits of knowledge systems coupled to the high cost and risk of development. The analysis in the second chapter has shown that four problems that belong more clearly in the area of scientific concern cause this uncertain bottom-line.

The knowledge acquisition and maintenance problem is the first and main problem to be solved. The gap problem defined as the difficulty to transform a knowledge model into an industrial strength, professional knowledge system was mentioned as the second problem. The third problem was the brittleness of knowledge system when confronted with problem situations just outside of their working area. The fourth and final problem responsible for the difficulty and risk associated with the knowledge system is the complexity of a the task that many knowledge system have to perform.

This thesis established the problems are as part of and partly caused by an ingrained engineering metaphor. This metaphor sees a knowledge system as an artefact. Much like any engineering product, a knowledge system can be designed, built and deployed in a sound and principled manner. It is the thesis of this research that knowledge systems do not conform to this metaphor, and that the solutions that are inspired by this metaphor do not solve the practical or the scientific problems that are experienced.

In the search for a global solution to the bottom-line problem and its root-problems an alternative metaphor was proposed based on the development of insight, rather than the engineering of a product. This metaphor perceives a knowledge system as a medium and knowledge system development as a continuous activity of knowledge acquisition. The research has examined the changes this would effect in different dimensions of knowledge system development: people, project, product, process and tools. It showed a number of changes that would be required to the interpretation of these dimensions. This included changes in the tools needed to support the changes in the other dimensions. The implementation and demonstration of this insight-based

scientific metaphor has been the further subject of research. This required the development of tools that support this type of development as well as the use of the guidelines and general ideas in knowledge system development projects.

### 8.1.1 Evaluation of the experiments

This section reiterates the conclusions reached upon analysis of the cases. These are in effect the tests with respect to the evaluation criteria. The results of these tests are used to determine the justification that there exists for the different metaphors.

#### People

In the explanation created from the engineering metaphor, knowledge is hard to articulate by experts. They require a skilled knowledge engineer to create a knowledge model, which constitutes a design by formulation and formalization supported by a modelling activity. The scientific metaphor sees an expert as the best person able to conduct the modelling. Furthermore, they are by virtue of their training and experience best able to appreciate possible new lessons to be learnt from the knowledge modelling and the application of the knowledge through the knowledge system. The engineering metaphor then predicts that experts cannot model, where the scientific metaphor would have that such a mode of operation is essential. Without it, continuous knowledge engineering becomes an impractical approach.

*People Test:* Ascertain whether experts are capable to participate in knowledge modelling or can independently model their own knowledge.

The cases show all three modes of operation: indirect, participatory and independent knowledge modelling. First of all, this invalidates the engineering metaphor's explanation of the knowledge acquisition bottleneck. Beyond this, the results are non sufficiently conclusive to declare a clear preference for independent modelling. It appears that both participatory and independent knowledge modelling has its merits. It does show however, that there needs to be no impediment for independent modelling, making schemes such as community knowledge bases (Steels 1992, 1986) and the knowledge producing next medium (Stefik 1986) possible.

#### Project

The engineering metaphor presupposes that knowledge system development is a finite process, followed by maintenance. The insight metaphor counters this by assuming that this development is necessarily continuous in nature. When most changes are necessary in a system before initial deployment, or conversely, afterwards, this would indicate support for one assumption while falsifying the other.

*Project Test:* Ascertain whether a project knows more change than building activity.

Each of the cases supports the view that building is a relatively minor activity compared to changing. This is even true when looking at the development running up to deployment. The extent of the changes that are made after deployment far exceed the effort in the developmental stage. It is as yet unclear whether this will continue to be so, or whether it will stabilise or converge on a minimal upkeep effort, but the longer lasting cases show no obvious signs of this. More extensive and longer duration examination will have to show this in more detail.

## Product

The product of knowledge engineering is a knowledge system, and the knowledge model is merely a part of the design to specify that system. This engineering view of knowledge equates a knowledge system with any other artefact: designed, implemented, tested and deployed. The discrepancies between the system's actual behaviour and the specification, caused by errors in the implementation or by changes in the specifications, are reasons to perform maintenance. This contrasts with the view that a knowledge system is a medium for communicating knowledge. In this view, the product of knowledge engineering is an ongoing knowledge model forming a theory of the domain, and a knowledge system is only a medium to communicate that knowledge. The theory will be replaced in time by a more predictive or more elegant theory. In this sense, a knowledge system is constantly changing as its knowledge model changes.

*Product Test:* Determine whether changes to a knowledge system originate in its environment or from the invalidation of the knowledge contained within.

Many of the cases show evidence that changes in the best theory are a main cause of change to the knowledge system. On the other hand, the systems also show external motivation for change, which is best explained by the engineering view. These changes were often limited to the look and feel of the system or missing auxiliary features. The latter does not diminish the notion of a knowledge system as a medium. The knowledge system may be *a* product, but the knowledge model is *the* product. It is similar to internet-browsers, which constitute products that nevertheless only provide a medium for Internet content.

## Process

While the incrementality is a feature of both engineering and scientific approaches, they differ on the reasons for using it. This leads to variations in the importance of phases, and in the ordering of the phases. The engineering metaphor with an emphasis on risk and cost minimization seeks to divide and conquer, professing no preference for either cyclic or phase-based incrementality. Most cases in literature show incremental design followed by incremental development (see also Chapter 2). The successive adjustments and critical feedback proposed by the scientific metaphor sees a greater role for cyclic incrementality. This leads to the question whether the preferred way is to perform complete cycles or be incremental within each of the phases or only specific ones.

*Process Test 1:* Establish whether some phases are more important than others are or whether they are equally important.

Posed alongside these assumptions are the notions stemming from the science metaphor that the complete cycles matter more. Therefore, the discovery of knowledge depends on having operational models at every moment in time.

*Process Test 2:* Establish whether cyclic development is more important than incremental division (the latter means executing phases incrementally).

The amount of insight, measured as the amount of change to a model shows that the cyclic mode is to be preferred to the incremental within design and implementation. The cases also show that none of the phases prevails over the others in generating the insight. It occurs over all phases, and every opportunity to do generate more must be

exploited. The case studies have shown the role that application experience can play to some extent, but this is a relationship that warrants additional study.

An important signal in this respect is in need for the change that is seen in the development of most knowledge models. The changes are not additions but seem to occur all over the model, leaving no part of the model in its initial form. This can be interpreted to mean that what can be considered correct at first may require change several times. This can also be taken as evidence that it is the behaviour of the system that must be correct, rather than the knowledge contained in the model. The inherent quality of the model, its consistency, is then merely a guarantee that the system will behave with the same quality in similar situations, rather than proof of its correctness. The view taken by this thesis is that the expert and the knowledge system learn together, sharing new insights that are discovered. This is the main reason for change.

### Tools

The investigation of the tools, by their different philosophies, is not meant to directly analyse the two metaphors. Both tools are inspired by the insight-based metaphor. They aim to elucidate the form of support that should be given to make adjustments in people, project, product and process possible, and thereby facilitate the investigation of the metaphors. The two forms of philosophy can be summarised as simplicity vs. vividness. The implied distinction is that it is not possible or at least hard to allow both simplicity and vividness.

*Tool Test 1.* A tool that can allow knowledge modelling by experts must be simple.

*Tool Test 2.* A tool that can allow knowledge modelling by experts must use a vivid modelling language, to aid the discovery of new insight and support the changeability of knowledge models by experts.

The dichotomy described by the tests above is that between simplicity and vividness. Increased vividness by providing more complete representation languages was thought to further complicate the task of non-specialist users such as the experts whose knowledge was to be modelled. The results show that tools based on either philosophy are usable by experts, with a difference in the learning curve involved. Furthermore, use of the KBE as a first generation tool shows definite signs of self-imposed structure, which is constructed out of the available ground materials. This is not unlike the results of the analyses of experts system performed by Clancey (1985, 1983), which showed different types of rules to exist in MYCIN and other knowledge systems over and beyond the rule formats and inference mechanisms.

The benefit of the structured representation is not an improved design or realisation of deep models. Rather the better analysability makes it easier to spot missing concepts and relationships, similar in a fashion to the decision table. In addition, it allows easier maintenance by modularisation, abstraction, and encapsulation. It is easier to understand a knowledge model, locate a place for change, and it reduces the ill effects of such a change (e.g. change cascades). Furthermore, it improves the predictability of the effect of changes. Therefore, its vividness is more important for the cognitive mapping and its insight producing characteristics, where the level of structure is more important in mediating the cost of changes.

### Synergy

*Synergy Test 1:* Determine to what extent the measures on the different dimensions are necessary.

*Synergy Test 2:* Determine to what extent the measures are influencing one another.

The synergy between the different aspects of the approach is clearly present. It is not necessary to address each of the dimensions in unison, as insight will be created even when only one of them is addressed, for example incrementality. Nevertheless, it is clear that the effects will strengthen each other. The most pronounced effects are visible when incrementality and expert modelling combine. In the MDDS system the knowledge acquisition bottleneck is solved to the same extent as it is in the GARVAN ES-1 system. The measures are therefore certainly not independent.

### 8.1.2 Metaphors

The scientific goal of this thesis was to research a global solution to the main problems in knowledge engineering. These two problems, the knowledge acquisition and maintenance bottleneck and the gap between a knowledge model and a professional, industrial strength application of that knowledge, were analysed and the insight-metaphor was proposed to explain these problems and provide direction for a possible remedies. The metaphor therefore stands as the global solution to these problems and has shown to be able to address these problems. The previous sections have shown the evaluation of a number of tests that were proposed to enable the differentiation of the engineering and the insight-based metaphor.

The results from the case-studies and the results from the analysis of the different tests show first of all that insight-based knowledge system development is not only a possibility, but that it is practical and a viable approach. It is an approach that has stronger requirements on the way knowledge systems are developed and how that development should be supported but these requirements are realisable, as can be seen from both the KBE and IO. Moreover, the approach does in fact realise benefits.

The analysis of the cases further shows that the perspective it gives can even strengthen the insight created in projects driven by an engineering outlook, because of the incorporation of the perspective in the capabilities of the tools. The effects are simply stronger when more aspects are directed towards insight creation, and in that case show a definite synergetic effect. This indicates that the perspective rather than any one of the individual measures taken is the factor that determines the effect. This can be taken as support for insight-based knowledge engineering, and evidence that the explanation the metaphor gives for the problems that are experienced in knowledge engineering are correct.

In addition, the cognitive model that was embodied by taking this approach, namely weak situated cognition is by virtue of this research strengthened as an attainable position, both as scientific model of cognition and a practicable next step for knowledge engineering. As research in this area is scarce, certainly when placed against the many systems created under the physical symbol systems hypothesis, this can feed the discussions with additional information.

The preliminary conclusion is that the standpoint represented by the insight-based metaphor is a valid alternative to the state-of-the-art and other accepted measures to solving the problems. Further research will be required to take what has been proposed here beyond these initial steps towards realisation of the metaphor and

complete its evaluation on a time-scale more fitting with the intentions of the approach.

### 8.1.3 Practical Problems and Solutions

The final conclusion of this thesis is reserved for the bottom-line problem associated with unclear benefits, and high cost and risk of knowledge system development. The problem for which a possible solution was sought, leading to the scientific questions answered above. The question addressed here is whether the global solution, proposed as the insight-based metaphor and its realisation shown in the case studies, solves the economic, practical bottom-line problem.

It was determined that an incremental development method was a sound approach to overcome the difficulties of knowledge system development that lead to the uncertain bottom-line, without actually addressing the problems that caused the uncertainty. It therefore is an approach that alleviates some of the adverse symptoms but does not cure the ground cause. The benefits of the approach depend on the granularity of incrementality, the magnitude and cost of each increment and the risk of each increment. As the scientific metaphor in many ways extends that which is proposed by the engineering metaphor, the questions that then remain are whether it makes the benefits clearer, lowers the cost and risk of development, and provides additional control given by the incrementality of the approach.

It has been shown that the benefits from the development of a knowledge system using the tools described in this thesis are more easily appraised as to their benefits in these cases shown in this thesis. The early availability of a usable system is seen to focus the development of the system to be in line with the current perceived utility of the system, rather than a prediction of that utility. In addition, the approach lowers the cost and risk of development and more easily spreads them by the smaller increments and direct application of the model. The cost of knowledge system development is due mostly to knowledge acquisition and knowledge system development. The first requires less effort per increment due to the lowering of the number of people involved in knowledge modelling and better support for modifications in the knowledge model. The increased support for knowledge system development lessens the effort to build a knowledge system. The risk is lowered by tying the expert more closely to the project, making acceptance of the system to its users a crucial factor in the development and serving knowledge which is much more up-to-date.

This has led to a more comfortable bottom-line shown in these case-studies, where clear decisions could be taken about which additional investments to make based on the proven value of the system in action. Because this proof of value is present from the first cycle, this clarity is there as soon as the first basic knowledge system is created.

## 8.2 Recommendations

In keeping with the program set out in the first chapter and in line with the insight-based metaphor, this first increment should inform and structure the next cycles in this research. This section therefore describes further avenues for research and goes over some of the inconclusive results that need further investigating. The greatest problem discerned, for which additional research is essential, is to look at projects, such as the ones described in this thesis, over a far more extended longer period. Concerning the systems and the outlook given in this thesis, one can only come to true conclusions

when the development over a system is monitored over its complete lifetime, from inception to its being decommissioned.

The tools examined in this thesis are not the only tools that could be employed; others may implement the features or provide similar mechanisms that may allow them to be employed. However, the tools described in this chapter were engineered for this specific purpose. Examination of the use of other existing tools can shed further clarity on the viability and practicality of the approach, and on the features that are usable to implement the guidelines.

The case studies as described in this thesis do not provide for an easy choice or preference for either the KBE or IO. Both tools have their benefits and disadvantages, making them more suitable for certain situations. The main problem found in the KBE beyond its fundamental limitations has to do with the lack of structure in its knowledge model. This may be solved by means a few shades shy of full-blown object-orientation. Perhaps a division of the model into separate modules can already stretch the capacities of the system to enable larger systems to be used more easily and strengthen the structure perceived and imposed by its users. The KBE is a very capable system for development of small-scale knowledge systems developed by the experts on their own.

IO, in all its expressive richness can be improved upon by taking the visual nature of the knowledge. The first instalment could be to include visual UML diagrams. This may even include further visual ways of entering language methods using program structure diagrams and the inclusion of other methods that allow highly visual presentations, such as cross-tables, and certain forms of case-based reasoning. Furthermore, assistance to the users may be given by realisation of wizards built in IO that operate on the reflective interfaces and aid the user in creating, maintaining and monitoring the system. For instance, such a wizard can aid the user when creating a new class or association, or in more complicated example may advise the user on restructuring the knowledge model. Further work on IO will however be likely to remove the learning curve associated with it. This will then make IO the tool of choice when compared to the KBE. The main reason for the favourable comparison is the ability to support the discovery of new insights, which IO aims for much more directly than the KBE.

Usage of a mature methodology such as CommonKADS adjusted to conform more to the process guidelines would allow explicit use of the continuous knowledge engineering approach, rather than the inclusion of the principles in ad-hoc or rapid prototyping settings. This will perhaps allow for a more principled application of the guidelines and create a basis for comparison to other systems built and maintained under CommonKADS.

A further research approach may be to promote tools such as these that have been developed further to be used by the common public. Perhaps then community knowledge model will arise (Steels 1994, 1986) or a knowledge producing industry such as described as the next medium by Stefik (1992). Such investigations may show other processes at work than are supposed now to be operational within rapid prototyping and methodological views of knowledge creation and discovery.

### 8.3 Contributions

The contributions of this research made the following contributions to the science of knowledge engineering:

- Signalled the engineering metaphor as an ingrained bias in the state-of-the-art and research of knowledge engineering and proposed an alternative metaphor for knowledge engineering based on insight-development to afford a further explanation of the problem of knowledge engineering and inspire new solutions;
- Presented a continuous knowledge engineering approach inspired by the insight-metaphor; Described a first- and second-generation tool for knowledge system development to support the approach;
- Showed a number of real-world practical cases employing the techniques and tools that were proposed, describing the quality of the proposed solution for both the scientific and practical problems.



# Summary

---

‘Towards Continuous Knowledge Engineering’, Ph.D.-thesis by K.O. Schilstra

Artificial Intelligence (AI) is hard to sell as a matter of fact solution. Even knowledge systems, AI’s economically most successful prodigy, have a bad reputation concerning their bottom-line. The main problems in the development of knowledge systems are the difficulty of knowledge acquisition and maintenance, and the gap that exists between prototype and industrial strength knowledge systems. These problems make their development a costly and risky enterprise, and create uncertainty as to the benefits of knowledge systems. This leads them to be perceived as experimental technology.

The wish for economically sound, principled development of knowledge systems coupled with a symbolic view of cognition has led to an ingrained engineering metaphor. This metaphor inspires the current approaches to these problems to employ structured methodology to design and realise them according to specification. Counter to this a science metaphor can be placed, motivated in part by situated theories of cognition. This metaphor sees knowledge system development as akin to working on a scientific theory, successively changed and subjected to critical review. This intends to indulge the learning character of knowledge rather than attempt to curb its fluid nature.

To investigate the differences between these two metaphors and their influence on the development of knowledge systems this research formulated a synthesis approach called ‘continuous knowledge engineering’. This approach is oriented to support an ongoing learning approach through its guiding principles: participation of experts in knowledge modelling, project management as stewardship over the complete lifecycle, use of knowledge systems as a medium rather than a product, and a cyclic development process. The approach does not aim to replace current practices, but reorients them, augmented with additional facilities.

To make the approach practical, it requires support from development tools. Two tools were developed with this in mind. The first is a simple tool based on a visual knowledge representation, and is limited in its scalability. The second is a more advanced system, building on the accomplishments of the first tool. It extends the number of knowledge representations and introduces object-oriented domain models to enable vivid models. Both tools were put into daily practice at the Knowledge Based Systems Group at TNO in a number of case-studies.

These case-studies allowed analysis of these tools and their support for continuous knowledge engineering. The results show that experts are indeed able to participate in the modelling of knowledge. The case studies further demonstrate the viability of stewardship through a dedicated organisation. In addition, the support to develop a professional system as a medium from the first knowledge model onwards is shown, with the knowledge system growing in specificity and completeness as the knowledge itself develops. Finally, the case studies show a highly cyclic process of development. The benefits of the approach are a gradual knowledge acquisition process, and support for creation and evolution of an advanced knowledge system. This lessens the cost and risk of knowledge system development, while it increases the clarity on the realised benefits. This significantly improves the bottom-line of knowledge systems.



# Samenvatting

---

‘Towards Continuous Knowledge Engineering’, proefschrift van K.O. Schilstra

Kunstmatige intelligentie is moeilijk te verkopen als een vanzelfsprekende oplossing. Zelfs kennissystemen, de economisch meest succesvolle afstammeling van kunstmatige intelligentie, hebben een slechte reputatie als het gaat om de rentabiliteit. De belangrijkste problemen bij de ontwikkeling van kennissystemen zijn de moeilijkheid waarmee kennis geacquireerd en onderhouden kan worden, en het gat tussen een prototype kennissysteem en een professioneel, bruikbaar software systeem. Deze problemen maken de ontwikkeling kostbaar en risicovol, en creëren onduidelijkheid als het gaat om de meerwaarde van kennissystemen. Dit heeft tot gevolg dat ze gezien worden als experimentele technologie, en reduceert mogelijkheden om ze te introduceren als mogelijke, praktische oplossingen voor veel van de kennisproblemen in professionele organisaties.

De wens om economisch aantrekkelijk en op onderbouwde wijze kennissystemen te ontwikkelen, samen met het gebruik van een symbolische theorie van cognitie heeft geleid tot een ingegroeide ‘engineering’ metafoor. Deze metafoor inspireert de huidige aanpakken om deze problemen op te lossen door het gebruik van gestructureerde methodologie voor het ontwerp en realisatie volgens specificatie. Daar tegenover kan een metafoor van wetenschap geplaatst worden, deels gemotiveerd door ‘situated cognition’. Deze metafoor ziet systeem ontwikkeling als verwant aan het werken aan een wetenschappelijke theorie, successievelijk veranderd en onderworpen aan kritische heroverweging. Dit heeft tot doel het leer karakter van kennis te omarmen, in plaats van het te vermijden.

Om de verschillen tussen deze metaforen en hun invloed op de ontwikkeling van kennissystemen te onderzoeken, is een synthese methodiek geformuleerd, ‘continuous knowledge engineering’ genaamd. Deze aanpak richt zich op het ondersteunen van een continu leerproces door een aantal principes te volgen: deelname van experts in het modelleren van kennis, beheer van de ontwikkeling als rentmeesterschap over de gehele levenscyclus van het systeem, gebruik van kennissystemen als een medium in plaats van als product, en een cyclisch ontwikkelingsproces. De aanpak richt zich er niet op om de huidige praktijken te vervangen, maar om ze te heroriënteren en aan te vullen met additionele faciliteiten.

Om de methodiek praktisch inzetbaar te maken is ondersteuning nodig uit de ontwikkelgereedschappen. Twee gereedschappen werden ontwikkeld met dit als doel. De eerste tool richt zich op simpliciteit, gebaseerd op een visuele kennisrepresentatie, en is beperkt in schaalbaarheid. De tweede tool is meer geavanceerd, en bouwt verder op de eerste tool. Met uitbreiding van het aantal kennisrepresentatie mogelijkheden en de introductie van object georiënteerde domein modellen ondersteund het de ontwikkeling van levensechte (vivid) kennismodellen. Beide tools werden in de dagelijkse praktijk van de groep Kennisgebaseerde Systemen van TNO in gezet in een aantal casestudies.

Deze casestudies maakten het mogelijk om de gereedschappen en de ondersteuning voor ‘continuous knowledge engineering’ te analyseren. De resultaten laten zien dat experts inderdaad goed in staat zijn om te participeren in het modelleren van kennis.

De casestudies demonstreren de mogelijkheden om kennissysteem ontwikkeling als rentmeesterschap te behandelen. Verder tonen zij de ondersteuning voor het ontwikkelen van professioneel systeem als medium voor een kennismodel, met het kennissysteem meegroeïend met het kennismodel in compleetheid. Als laatste laten de casestudies de rol zien van een proces met een hoge mate van cycliciteit. De voordelen van deze aanpak zijn een graduele kennisacquisitie process en ondersteuning voor het ontwikkelen en evolueren van een geavanceerde kennissystemen. Dit verlaagt de kosten en de risico's die verbonden zijn aan de ontwikkeling van kennissystemen terwijl het zicht op de gerealiseerde meerwaarde wordt vergroot. Dit verbetert de rentabiliteit van kennissystemen.

# Bibliography

---

- Aamodt, A. and E. Plaza (1994) Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *Artificial Intelligence Communications* 7(1):39-52.
- Aarts, M., R. Böhmer, K. Schilstra and P. Spronck (1999) Kennismanagement en Kennistechnologie. *Informatie* 41:18-27. (in Dutch)
- Adan, O. (1994) On the fungal defacement of interior finishes. Ph.D. Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands.
- Alexander, C. (1979) *The Timeless Way of Building*. New York: Oxford University Press.
- Bachant, J. and J. McDermott (1984) R1 revisited: Four years in the trenches. *AI Magazine* 5(3):21-32.
- Bobrow, D.G., S. Mittal and M.J. Stefik (1986) Expert Systems: Perils and Promise. *Communications of the ACM* 29(9):880-894.
- Boehm, B.W. (1988) A Spiral Model for Software Development and Enhancement. *IEEE Computer* 21(5):61-72.
- Booch, G., J. Rumbaugh and I. Jacobson (1999) *The Unified Modeling Language User Guide*. Reading, MA: Addison-Wesley.
- Borsje, H., R. Vanhauten, H. Soen and A. Wendholt (1999) Prinduceb III: Betoncentrale van de 21<sup>e</sup> eeuw. Technical Report TNO-1999-BT-R1403. TNO Bouw, Rijswijk, The Netherlands. (in Dutch)
- Breuker, J. and B. Wielinga (1986) Analyse van Expertise. *Ontwikkelingen in Expertsystemen*, A. Nijholt, A. and L. Steels (eds.). Academic Service.
- Brooks, F.P. Jr. (1987) No Silver Bullet: Essence and Accidents of Software Engineering. *Computer* 20(4):10-9.
- Brooks, R.A. (1991a) Intelligence without Representation. *Artificial Intelligence* 47: 139-159.
- Brooks, R.A. (1991b) Intelligence without Reasoning. *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pp. 569-595. San Francisco, CA: Morgan Kaufmann.
- Brown, A.W. and K.C. Wallnau (1996) Engineering of Component-Based Systems, *Component-Based Software Engineering: Selected Papers from the Software Engineering Institute*, pp. 7-15. Los Alamitos, CA: IEEE Computer Society Press.
- Bryant, A. (2000a) Metaphor, myth and mimicry: The bases of software engineering. *Annals of Software Engineering* 10:273-292.
- Bryant, A. (2000b) It's engineering Jim ... but not as we know it: software engineering - solution to the software crisis, or part of the problem? *Proceedings of the 22<sup>nd</sup> international conference on Software engineering*, pp. 78-87.
- Buchanon, B.G. (1989) What Do Expert Systems Offer the Science of AI. *Applications of Expert Systems*, J.R. Quinlan, (ed.), pp. 11-35. Reading, MA: Addison-Wesley.

- Buchanon, B.G., D. Barstow, R. Bechtal, James Benett, W. Clancey, C. Kulikowski, T. Mitchell and D. Waterman (1983) Constructing an Expert System. *Building Experts Systems*, F. Hayes-Roth, D. Waterman and D. Lenat (eds.), pp. 127-167. New York: Addison Wesley.
- Chandrasekaran, B. and T.R. Johnson (1993) Generic tasks and task structures: History, critique and new directions. *Second Generation Expert Systems*, J. M. David, J.P. Krivine and R. Simmons (eds.). Berlin: Springer-Verlag.
- Clancey, W. J. (1999) Situated Cognition: On Human Knowledge and Computer Representation. Cambridge University Press.
- Clancey, W. J. (1992) Model Construction Operators. *Artificial Intelligence* 53(1):1-124.
- Clancey, W.J. (1987) Book review of Winograd & Flores, Understanding computers and cognition: A new foundation for design. *Artificial Intelligence* 31:233–250.
- Clancey, W.J. (1985) Heuristic Classification. *Artificial Intelligence* 27:289-350.
- Clancey, W.J. (1983) The Epistemology of a Rule-Based Expert System – a Framework for Explanation. *Artificial Intelligence* 20:215–251.
- Coenen, F. and T. Bench-Capon (1993) Maintenance of Knowledge Based Systems. London: Academic Press.
- Compton, P., G. Edwards, B. Kang, L. Lazarus, R. Malor, P. Preston, and A. Srinivasan (1992) Ripple down rules: Turning knowledge acquisition into knowledge maintenance. *Artificial Intelligence in Medicine* 4:463-475.
- Compton, P. and R. Jansen (1990) A Philosophical Basis for Knowledge Acquisition. *Knowledge Acquisition* 2:241-257.
- Compton, P, K. Horn, J.R. Quinlan, and L. Lazarus (1989) Maintaining an Expert System. *Applications of Expert Systems*, J.R. Quinlan (ed.), pp. 366-384. Reading, MA: Addison-Wesley.
- Corbridge, C., N. Major, and N. Shadbolt (1995) Models Exposed: An Emperical Study. *Proceedings of the 9<sup>th</sup> AAAI-Sponsored Banff Knowledge Acquisition for Knowledge Based Systems*, pp. 13·1-13·21.
- Crofts, A.E., V.B. Ciesielski, M. Molesworth, T.J. Smith, and R.Y. Lee. (1989) Bridging the Gap Between Prototype and Commercial Expert Systems – A Case Study. *Applications of Expert Systems*, J.R. Quinlan, (ed.), pp. 93-105. Reading, MA: Addison-Wesley.
- Davis, R. (1980) Meta-rules: Reasoning about Control. *Artificial Intelligence* 15(3):179-222.
- Davis, R., H. Shrobe and P. Szolovits (1993) What is a Knowledge Representation. *AI Magazine* 14:17-33.
- Dennet, D.C. (1987) *The Intentional Stance*. Cambridge, MA: MIT Press.
- de Hoog, R. (1998) List of frequently encountered project risks. Technical Report. SWI, University of Amsterdam.
- Dreyfuss, H. (1987) Misrepresenting Human Intelligence. *Artificial Intelligence: The Case Against*, R. Born (ed.), pp. 41-54. London: Croom Helm.
- Dreyfuss, H. (1979) *What Computers Can't Do: A Critique of Artificial Reason*. Freeman.
- Dreyfuss, H. and S. Dreyfuss (1985) From Socrates to Expert Systems: The Limits of Calculative Rationality. *Philosophy and Technology II: Information Technology*

- and Computers in Theory and Practice*, Mitcham, C. and Huning, A. (eds.). Boston Studies in the Philosophy of Science Series. Reidel.
- Duda, R., J. Gaschnig and P. Hart (1979) Model design in the Prospector consultant system for mineral exploration. *Expert Systems in the Microelectronic Age*, D. Michie (ed.), pp. 153–167. Edinburgh: Edinburgh University Press.
- Edelman, G. M. (1992) *Bright Air, Brilliant Fire: On the matter of the mind*. New York: Basic Books.
- Eisenstadt, M., J. Domingue, T. Rajan and E. Motta (1990) Visual Knowledge Engineering. *Transactions in Software Engineering* 16(10):1164-1177.
- Endres, A. (1996) A Synopsis of Software Engineering History: The Industrial Perspective. *History of Software Engineering*, Dagstuhl Seminar 9635, Schloß Dagstuhl, August 26-30.
- Feigenbaum, E. (1982) Knowledge Engineering in the 1980's, Dept. of Computer Science, Stanford University, Stanford, CA.
- Feigenbaum, E., B. Buchanon and J. Lederberg (1971) On Generality and Problem Solving: A Case Study Using the DENRAL Program. *Machine Intelligence* 6:165-190. Edinburgh: Edinburgh University Press.
- Feigenbaum, E.A. (1977) The art of artificial intelligence: Themes and case studies in knowledge engineering, *Proceedings of the 5<sup>th</sup> International Joint Conference on Artificial Intelligence (IJCAI-77)*, pp. 1014-1029.
- Fikes, R. and T. Kehler (1985) The role of frame-based representation in reasoning. *Communications of the ACM* 28(9):904-920.
- Finin, T., D. McKay and R. Fritson (1992) An Overview of KQML: A Knowledge Query and Manipulation Language. Technical report. Department of Computer Science, University of Maryland.
- Fuchs, N. E. (1992) Specifications are (preferably) executable. *Software Engineering Journal* 7(5):323-334.
- Fowler, M. (2001) The New Methodology. <http://www.martinfowler.com/articles/newMethodology.html>.
- Fowler, M. (2000) Put Your Process on a Diet. *Software Development Magazine*, December.
- Gamma, E., R. Helm, R. Johnson and J. Vlissides (1994) *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley.
- Gaspari, M., E. Motta and A. Stutt (1993) Inferring in Lego-land: an Architecture for the Integration of Heterogeneous Inference Modules. *Languages, Architectures and Tools for AI*, pp. 142-153.
- Genesereth, M. and R. Fikes (eds.) (1992) Knowledge Interchange Format, Version 3.0. Technical Report Logic-91-1. Computer Science Department, Stanford University.
- Genesereth, M. and S.P. Ketchpel (1994) Software Agents. *Communications of the ACM* 37(7):48-53.
- Giarratano, J. and G. Riley (1989) *Experts Systems: principles and programming*. Boston, MA: PWS-Kent.
- Ginsberg, M.L. (1991) Knowledge Interchange Format: The KIF of Death. *AI Magazine* 12(3):57-63.

- Goldberg, A. (1990) Information models, views, and controllers. *Dr.Dobbs Journal*, July 54-61.
- Goldberg, D. (1989) Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, MA: Addison-Wesley.
- Guha, R.V. and D.B. Lenat (1990) CYC: A Midterm Report. *AI Magazine* 11(3):32-59.
- Hammond, K. (1989) Case-based Planning: Viewing Planning as a Memory Task. Boston, MA: Academic Press.
- Harmon, P. (1995) Object-Oriented AI: A Commercial Perspective. *Communications of the ACM* 38(11):80-86.
- Hayes, P. J. (1985) The second naïve physics manifesto. *Formal theories of the commonsense world*, J. Hobbs and R. Moore (eds.), pp. 1-20. Norwood, NJ: Ablex Publishing Corporation.
- Hayes, P. J. (1979) The Logic of Frames. *Frame Conceptions and Text Understanding*, D. Metzger (ed.). Walter de Gruyter and Co: Berlin.
- Hayes, I.J. and Jones, C.B. (1989) Specifications are not (necessarily) executable. *IEEE/BCS Software Engineering Journal* 4(6):330-338.
- Hayes-Roth, F., D. Waterman and D. Lenat (eds.) (1983) Building Expert Systems. New York: Addison-Wesley.
- Hees, R., L. Pel and B. Lubelli (2000) Towards Compatible Repair Mortars for Masonry in Monuments. *5<sup>th</sup> International Symposium on the Conservation of Monuments in the Mediterranean Basin*, E. Galán (ed.), Seville.
- van den Herik, H.J. (1988) Informatica en het menselijke blikveld . Inaugural Address, Maastricht, University of Limburg. Faculty of General Sciences, The Netherlands. (in Dutch)
- van den Herik, H.J. (1986) Het gebruik van kennis in expertsystemen. *Ontwikkelingen in Expertsystemen*, A. Nijholt, and L. Steels (eds). Academic Service.
- Holland, J.H. (1986) Escaping brittleness. *Proceedings of the Second International Workshop on Machine Learning*, J. G. Carbonell, R. S. Michalski, and T. M. Mitchell (eds.), pp. 92-95
- Jacobson, I., G. Booch and J. Rumbaugh (1998) The Unified Software Development Process. Reading, MA: Addison-Wesley.
- Jeffries, R., A. Anderson and C. Hendrickson (2001) Extreme Programming Installed. Reading, MA: Addison-Wesley.
- Janse, E. (2000) Implementatie van BRAS in het IO Concept. Technical Report 2000-CVB-R00701. TNO Bouw, Rijswijk, The Netherlands. (in Dutch)
- Jennings, N.R. and M. Wooldridge (1997) Applications of intelligent agents. *Agent Technology: Foundations, Applications, Markets*. N.R. Jennings and M.R. Wooldridge (eds.). Berlin: Springer-Verlag.
- Johnson, R. (1992) Documenting Frameworks using Patterns, *Proceedings of OOPSLA '92*, ACM SIGPLAN, pp. 63-76.
- Jongeneel, C. (1996) The informatical world view. Ph.D. Thesis, Delft University of Technology, Delft, The Netherlands.
- Kaindl, H. (1994) Object-oriented approaches in software engineering and artificial intelligence. *Journal of Object-oriented Programming*, January, pp. 38-45.



- Killin, J. (1993) Managing and Maintaining an Operational KADS System. *KADS: A Principled Approach to Knowledge System Development*, G. Schreiber, B. Wielinga, and J. Breuker (eds.), pp. 289-311. Academic Press.
- Kolodner, J. (1993) Case-based reasoning. San Mateo, CA: Morgan Kaufmann.
- Koppelaar, H. (2000) Personal Communication.
- Krasner, G.E. and S.T. Pope (1988) A Cookbook for Using the Model-View-Controller User Interface paradigm in Smalltalk-80. *Journal of Object-oriented Programming*, August, pp.26-49.
- Kremer, R. (1997) Constraint Graphs: A Concept Map Meta-Language, Ph.D. Thesis, University of Calgary, Canada.
- Kuechler, W.L. Jr., V. K. Vaishnavi and D. Turk (1995) Implementation of a Smart Object Interpreter: An Object Engine Use Case. *Proceedings of Object-Oriented Programming Systems, Languages and Applications (OOPSLA '95)*.
- Kuhn, T. S. (1970) The Structure of Scientific Revolutions (Second Edition, Enlarged). *Foundations of the Unity of Science*, pp. 55-273. Chicago, IL: University of Chicago Press.
- Kuijpers, P. (1998) XML: Een praktische aanpak. Technical Report TNO-98-CON-R0975. TNO Bouw, Rijswijk, The Netherlands. (in Dutch)
- Kurzweil, R. (1999) The Age of Spiritual Machines: When Computers Exceed Human Intelligence. Penguin Books.
- LaFrance, M. (1997) Metaphors for Expertise: How Knowledge Engineers Picture Human Expertise. *Expertise in Context: Human and Machine*, P.J. Feltovich, K.M. Ford and R.R. Hoffmann (eds.), pp. 163-180. Menlo Park, CA: AAAI Press.
- Langley, P. and H.A. Simon (1995) Applications of Machine Learning and Rule Induction. *Communications of the ACM* 38(11):54-64.
- Laurie, S. (1992) Experience on a disk. *The Banker* 142(795), pp. 56.
- Lenat, D.B. (1998) The Dimensions of Context Space. Technical report, Cycorp. <http://www.cyc.com/context-space.doc>
- Lenat, D.B. and E.A. Feigenbaum (1989) On the Thresholds of Knowledge. *Applications of Expert Systems*, J.R. Quinlan, (ed.), pp. 11-35. Reading, MA: Addison-Wesley.
- Lethbridge, T.C. (2000) Evaluating a Domain-Specialist Oriented Knowledge Management System. *International Journal of Human-Computer Studies* (to appear)
- Lethbridge, T. C. (1998) Metrics for Concept-Oriented Knowledge Bases. *International Journal of Software Engineering and Knowledge Engineering* 8(2):161-188.
- Lethbridge, T.C. (1994) Practical Techniques for Organizing and Measuring Knowledge. Ph.D. Thesis, University of Ottawa, Canada.
- Levesque, H. (1986) Making believers out of computers. *Artificial Intelligence* 30(1):81-108.
- Lindsay, R., B. Buchanan, E. Feigenbaum and J. Lederberg (1980) *Applications of Artificial Intelligence for Organic Chemistry: The DENDRAL Project*. New York: McGraw-Hill.

- Linster, M. (1994) Problem statement for Sisyphus: models of problem solving. *International Journal of Human-Computer Studies* 40:187-192.
- López, B. and E. Plaza (1993) Case-Based Planning for Medical Diagnosis. *Proceedings of the 7th International Symposium on Methodologies for Intelligent Systems (ISMIS'93)*, pp. 96-105.
- Lucardie, L. (1994) Functional Object-Types as a Foundation of Complex Knowledge-Based Systems. Ph.D. Thesis, Eindhoven Technical University, Eindhoven. Rijswijk: TNO Bouw.
- Lucardie, L. (1992) Integratie van kennis- en database technologie voor complexe kennisgebaseerde systemen. *Proceedings of Kennistechnologie '92*, pp. 87-101. (in Dutch)
- Maes, P. (1994) Agents that reduce work and information overload. *Communications of the ACM* 37(7):31-40.
- Maes, P. (1989) Computational Reflection. Ph.D. Thesis, AI Lab, Free University of Brussels, Belgium.
- Marques, D., G. Deallemagne, G. Klinker, J. McDermott and D. Tung (1992) Easy Programming: Empowering People to Build Their Own Applications. *IEEE Expert*, June, pp. 16-29.
- McCabe, T. J. (1976) A Complexity Measure. *IEEE Transactions on Software Engineering* 2(6):308-320.
- McCarthy, J. (1968) Programs with common sense. *Semantic Information Processing*, Minsky, M. (ed.). Cambridge, MA: MIT Press.
- McConnell, S. (1993) Code Complete: A Practical Handbook of Software Construction. Redmond, WA: Microsoft Press.
- McDermott, D. (1987) A Critique of Pure Reason. *Computational Intelligence* 3:151-160.
- McDermott, J. (1982) R1: A Rule-Based Configurer of Computer Systems. *Artificial Intelligence* 19:39-88.
- McDermott, J. (1980) R1: The formative years. *AI Magazine* 2(2):21-29.
- Menzies, T.J. and J. Debenham (2000) Expert Systems Maintenance, *Encyclopedia of Computer Science and Technology*, A. Kent and J.G. Williams (eds.), pp. 35-54. Marcell Dekker Inc.
- Menzies, T.J. (1998) Towards Situated Knowledge Acquisition. Special Issue: The Challenge of Situated Cognition for Symbolic Knowledge Based Systems *International Journal of Human-Computer Studies* 49:867-893.
- Menzies, T.J. (1997) Object-Oriented Patterns: Lessons from Expert Systems. *Software – Practice and Experience* 27(12):1457-1478.
- Menzies, T.J. (1996a) Assessing responses to Situated Cognition. *Proceedings of the 10th Knowledge Acquisition Workshop for Knowledge-Based Systems*, Banff, Canada.
- Menzies, T.J. (1996b) Visual Programming, Knowledge Engineering and Software Engineering. *Proceedings of the Eighth International Conference on Software Engineering and Knowledge Engineering*. Knowledge Systems Institute, Skokie, Illinois.

- Menzies, T.J. (1995) Limits to Knowledge Level-B Modeling (and KADS). *Eighth Australian Joint Conference on Artificial Intelligence (AI '95)*, Canberra, Australia. World-Scientific.
- Ministerie van VROM (1998) Bouwstoffenbesluit, parts 1, 2 and 3. (in Dutch)
- Minsky, M. (1990) Logical vs. Analogical or Symbolic vs. Connectionist or Neat vs. Scruffy. *Artificial Intelligence at MIT, Expanding Frontiers*, Winston P. (ed.), pp. 218-243. Cambridge: MIT Press.
- Minsky, M. (1988) *The Society of Mind*. New York: Simon and Schuster, Inc.
- Minsky, M. (1975) A framework for representing knowledge. *The Psychology of Computer Vision*, P.H. Winston (ed.), pp. 211-277. New York: McGraw-Hill.
- Montelbano, M. (1973) *Decision Tables*. Palo Alto, CA: Science Research Associates, Inc.
- Mors, N.P.M. (1993) *Beslissingstabellen*. Leidschendam: Lansa Publishing BV. (in Dutch)
- Motta, E. (1999) Reusable Components for Knowledge Modelling: Case Studies in Parametric Design Problem Solving. *Frontiers in Artificial Intelligence and Applications*, Vol. 53. Amsterdam: IOS Press.
- Motta, E., D. Fensel, M. Gaspari and V.R. Benjamins (1999) Specifications of Knowledge Component Reuse. *11th International Conference on Software Engineering and Knowledge Engineering (SEKE '99)*.
- Newell, A. (1982) The Knowledge Level. *Artificial Intelligence* 18(1):87-127.
- Newell, A. and H.A. Simon (1963) GPS, A Program That Simulates Human Thought *Computers and Thought*, E. Feigenbaum, E. and J. Feldman (eds.), pp. 279-293. New York: McGraw-Hill.
- Newell, A. and H.A. Simon (1981) Computer science as empirical inquiry: Symbols and search. *Mind Design*, J. Haugeland (ed.), pp. 35-66. Cambridge, MA: MIT Press.
- Newell, A. and H.A. Simon (1976) Computer science as empirical enquiry: symbols and search. *Communications of the ACM* 19(3):113-126.
- Nilsson, N. J. (1998) *Artificial Intelligence: A New Synthesis*. San Francisco, CA: Morgan Kaufmann.
- Preston, P., G. Edwards and P. Compton (1994) A 2000 Rule Expert System Without a Knowledge Engineer. *Proceedings of the 8th AAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada.
- Preston, P., G. Edwards and P. Compton (1993) A 1600 Rule Expert System Without Knowledge Engineers. *Second World Congress on Expert Systems*, J. Leibowitz (ed.), pp. 220-228.
- Redmond, M. (1990) Distributed cases for case-based reasoning: Facilitating use of multiple cases. *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pp. 304-309.
- Riesbeck, C.K. and R.C. Schank (1989) *Inside Case-based Reasoning*. Hillsdale, NJ: Lawrence Erlbaum.
- Ringland, G.A. and D.A. Duce (eds.) (1988) *Approaches to Knowledge Representation: An Introduction*. New York: John Wiley & Sons.

- Rumbaugh, J., I. Jacobson and G. Booch (1998) *The Unified Modeling Language Reference Manual*. Reading, MA: Addison-Wesley.
- Russel, S. and P. Norvig (1995) *Artificial Intelligence: A Modern Approach*. New York: Prentice Hall.
- Scandura, J.M. (1994) Converting Legacy Code into ADA: A Cognitive Approach. *Computer* 27(4):55-61.
- Schank, R.C. and R.P. Abelson (1977) *Scripts, Plans, Goals and Understanding*. Hillsdale, NJ: Lawrence Erlbaum.
- Schilstra, K. and P. Spronck (2000) Towards Continuous Knowledge Engineering. *The 20<sup>th</sup> SGES International Conference on Knowledge Based Systems and Applied Artificial Intelligence (ES2000)*, A. MacIntosh, M. Moulton and F. Coenen (eds.), Cambridge, UK, pp. 49-62.
- Scholten, N.P.M. (2001) Technische en juridische grondslagen van de technische bouwregelgeving Woningwet en Bouwbesluit. Ph.D. Thesis, Delft University of Technology, Delft, The Netherlands. (in Dutch)
- Schreiber, G., H. Akkermans, A. Anjewierden, R. de Hoog, N. Shadbolt, W. van der Velde and Bob Wielinga (1999) *Knowledge Engineering and Management*. Cambridge, MA: MIT Press.
- Searle, J. (1980) Mind, Brains and Programs. *The Behavioural and Brain Sciences* 3:417-457.
- Shapiro, S. (1987) *Encyclopedia of Artificial Intelligence*, Vols. 1 and 2 (Second edition). New York: John Wiley & Sons.
- Shortliffe, E. (1976) *Computer Based Medical Consultations: MYCIN*. New York: Elsevier.
- Simon, H.A. (1982) *The Sciences of the Artificial (Second Edition)*. Cambridge, MA: MIT Press.
- Smith, R.G. and J.D. Baker (1983) The Dipmeter Advisor System: A Case Study in Commercial Expert System Development. *Proceedings of the 8th International Joint Conference on Artificial Intelligence (IJCAI-83)*, Karlsruhe, Germany, pp. 122-129.
- Smyth, B. and E. McKenna (2000) Incremental Footprint-based Retrieval. *The 20<sup>th</sup> SGES International Conference on Knowledge Based Systems and Applied Artificial Intelligence (ES2000)*, Cambridge, A. MacIntosh, M. Moulton and F. Coenen (eds.), UK, pp. 49-62.
- Soloway, E., J. Bachant and K. Jensen (1987) Assessing the maintainability of XCON-IN-RIME: Coping with the problems of a very large rule-base. *Proceedings of Sixth National Conference on Artificial Intelligence (AAAI-87)*, pp. 824-829.
- Spronck, P.H.M., and K.O. Schilstra (2000) BOKS: A Knowledge Based System in Support of the Dutch Building Materials Regulations. *Proceedings of the Symposium on the Application of Artificial Intelligence in Industry (PRICAI 2000)*, D. Lukose and G.J. Williams (eds.), pp. 121-131.
- Spronck, P. (1996) *Elegance: Genetic Algorithms in Reinforcement Control*. M.Sc. Thesis, Delft University of Technology, Delft, The Netherlands.
- Stanchfield, (1997) Applying the Model/View/Controller Design Paradigm in VisualAge for Java. IBM, <http://www.ibm.com/>.

- Stapleton, J. (1997) DSDM: Dynamic Systems Development Method. Reading, MA: Addison-Wesley.
- Steels, L. (1992) Kennissystemen. Reading, MA: Addison-Wesley. (in Dutch)
- Steels, L. (1990) Components of Expertise. *AI Magazine* 11(2):28-49.
- Steels, L. (1986) Collectieve Geheugens. *Ontwikkelingen in Expertsystemen*, A. Nijholt and L. Steels (eds.) Academic Service. (in Dutch)
- Stefik, M. (1995) Introduction to Knowledge Systems. San Mateo, CA: Morgan Kaufmann.
- Stefik, M. (1986) The Next Knowledge Medium. *AI Magazine* 7(1):34:46.
- Stroustrup, B. (1991) The C++ Programming Language (Second edition). Reading, MA: Addison-Wesley.
- Studer, R., S. Decker, D. Fensel and S. Staab (2000) Situation and Perspective of Knowledge Engineering. *Knowledge Engineering and Agent Technology*, J. Cuenca, Y. Demazeau, A. Garcia and J. Treur (eds.). Amsterdam: IOS Press.
- Studer, R., V.R. Benjamins and D. Fensel (1998) Knowledge Engineering: Principles and methods. *Data & Knowledge Engineering* 25:161-198.
- Sussman, G. (1975) A Computer Model of Skill Acquisition. Amsterdam: Elsevier/North-Holland.
- Turban, E. (1990) Decision Support and Expert Systems: Management Support Systems (Second Edition). New York: MacMillan Publishing Company.
- van de Velde, W. (1986) Leren in Tweede Generatie Kennissystemen, *Ontwikkelingen in Expertsystemen*, A. Nijholt and L. Steels (eds.). Academic Service. (in Dutch)
- Verhelst, M. (1980) De praktijk van beslissingstabellen. Deventer: Kluwer. (in Dutch)
- Visser, P., T. Bench-Capon and J. van den Herik (1997) A Method for Conceptualising Legal Domains: An Example from the Dutch Unemployment Benefits Act. *Artificial Intelligence and Law* 5:207-242.
- Warmer, J.B., and A.G. Kleppe (1999) The Object Constraint Language: Precise Modeling with UML. Reading, MA: Addison-Wesley.
- Waters, J.R. and N.R. Nielsen (1988) Crafting Knowledge Based Systems: Expert Systems Made Easy/Realistic. New York: Wiley.
- Way, E. C. (1991) Knowledge Representation and Metaphor. Dordrecht, The Netherlands: Kluwer Academic Publishers.
- WordNet (1997) WordNet 1.6. Princeton University, <http://www.cogsci.princeton.edu/~wn/>.
- Wilson, E.O. (1998) Consilience. New York: Alfred A. Knopff.
- Winograd, T. (1975) Frame representations and the declarative/procedural controversy. *Representation and understanding: Studies and cognitive science*, D.G. Bobrow and A. Collins (eds.), pp. 185-210. New York: Academic Press.
- Woolridge, M. and N.R. Jennings (1995) Intelligent agents: theory and practice. *The knowledge engineering review* 10(2):115-152.
- Zadeh, L. (1975) Fuzzy Logic and Approximate Reasoning. *Synthese* 30:407-428



## Curriculum Vitae

---

The author Klaas Schilstra was born in Oudega, Hemelumer Oldeferd, Friesland on the 23<sup>rd</sup> of May 1970. He studied Technical Informatics at Delft University of Technology on the subject of “An Intelligent Prefetching Cache using a Case-based Temporal Pattern Recognition Engine”. This was the result of a yearlong visit at the Concurrent Engineering Research Centre in Morgantown, WV, USA.

After a short excursion to the Artificial Intelligence Group at the University of York, York, UK, he started work at the Knowledge-based Systems Department at TNO in Rijswijk. Associated with a team, involved in the construction of knowledge systems and tools to support that construction, proved fertile ground for new ideas. Sponsored by the Ministry of Economic Affairs and by TNO, this also signalled the beginning of a Ph.D. study at the TU Delft on that subject. Eventually, this led to this thesis.

He currently works at VIS Entertainment in Scotland, where he develops computer games.