Knowledge Technology Applications for Knowledge Management

KENT ANDERSSON

UPPSALA UNIVERSITY 2000

Dissertation for the Degree of Doctor of Philosophy in Computer Science presented at Uppsala University in 2000

ABSTRACT

Andersson, K. (2000). Knowledge Technology Applications for Knowledge Management. 35 pp. Uppsala. ISBN 91-506-1437-1.

We investigate how the knowledge in knowledge technology applications for knowledge management can be represented to let the user directly manage the knowledge content of the applications.

In paper I we design a representation of diagnosis knowledge that allows the user to add new components and inspect the function of the device. The representation allows an integration of model based knowledge with compiled and heuristic knowledge so that the device and its function can be represented a suitable level of abstraction and let other parts be represented as non-model based knowledge.

In paper II we use simplified rules for describing the time, resources, activities and amounts required in a tunnelling project and a simulation engine for estimating time and amounts consumed in the tunnelling process. The rules are designed to allow a user to change the facts and computations of the system.

In paper III we present the constraint modelling language CML and show how to model a nurse scheduling problem and a train scheduling problem without programming. The idea is to preserve the problem structure of the domain, allowing constraint formulations that reflect natural language expressions familiar to the users. CML problem specifications are transformed automatically to standard constraint programs.

In paper IV we investigate the use of decision tables for representing requirements on staff scheduling explicitly, providing structure, decision support and overview to the user. The requirements are compiled automatically to a program that use hand-written procedures for efficient scheduling.

It seems possible to let the user modify central parts of the knowledge content in the applications with these representations, by using various illustration techniques. The techniques used are object-based graphics for manipulating device components and connections in diagnosis, simplified rules for simulation of tunnelling activities, text-based query language specification of scheduling problems and finally, decision tables for constraint problems and decision support.

Kent Andersson, Division of Computer and Systems Science, Department of Information Science, Box 513, SE–751 20 Uppsala, Sweden

ISBN 91-506-1437-1

Printed in Sweden by Uppsala University. Tryck & Medier, Uppsala 2000
Distributor: Department of Information Science, Uppsala University

To my parents and my family

In memory of Dan Andersson 1928–2000

This thesis is based on the following papers

Paper I

Andersson, K. (1997). Integrating Heuristic and Model-Based Diagnosis, Licentiate Thesis, *Uppsala Theses in Computing Science* 27, Uppsala.

A short version published as:

Andersson, K. (1998). Components for Integrating Heuristic and Model-Based Diagnosis, in Gaines, B. and Musen, M., Eds., *Proceedings of the Eleventh Workshop on Knowledge Acquisition for Knowledge-Based Systems (KAW'98)*, Banff, Alberta, April 1998.

Paper II

Andersson, K., Hansson, Å. and Hjerpe, T. (1999). Tunnelling Analysis as Planning and Simulation, *The First International Conference on The Practical Application of Constraint Technologies and Logic Programming (PACLP'99)*, London, pp. 51-61.

A first version published as:

Andersson, K. and Hjerpe, T. (1997). Stochastic Simulation of Tunnelling Projects in Logic Programming, *The 10th Exhibition and Symposium on Industrial Applications of Prolog (INAP'97)*, Kobe University, Kobe, Hyogo, Japan, pp. 44-47.

Paper III

Andersson, K. and Hjerpe, T. (1998). Modeling Constraint Problems in CML, *The Third International Conference on the Practical Application of Constraint Technology (PACT'98)*, London, pp. 295–312.

Paper IV

Andersson, K. and Hjerpe, T. (2000). Knowledge Management of Constraint Problems in Decision Tables. T*he Second International Conference on The Practical Application of Constraint Technologies and Logic Programming (PACLP'2000)*, Manchester, pp. 291–307.

Contents

Chapter 1

Introduction

In this thesis we investigate knowledge technology applications for knowledge management. We investigate how the knowledge in these applications can be represented so that it is accessible, modifiable and understandable to ordinary users needing knowledge to do their jobs—not experts in programming. The thesis is based on applications in quite different areas: Diagnosis, simulation and scheduling, although knowledge technology is applicable for knowledge management in many other areas, for example configuration of products or services. The aim is to design open representations of knowledge for the users and knowledge providers of these applications, by creating representations amenable to inspection, modification and automatic processing.

Inspectability is important in two respects: The possibility of understanding the knowledge in the application and the possibility of trusting the result of the computation. Modifiability is important for contributing knowledge to the application. Automatic processing is needed to process the knowledge from a form understandable to the user into a form understandable to the computer.

The next section gives an introductory overview of the knowledge management area to place the thesis into context. Section 1.2 similarly introduces the knowledge technology area and briefly discusses the relation between the two areas. Section 1.3 introduces the papers of the thesis and chapter 2 gives a summary of each paper. Section 2.5 discusses related work for each paper and section 2.6 concludes the thesis.

1.1 KNOWLEDGE MANAGEMENT

The area of Knowledge Management is an interdisciplinary subject that deals with all aspects of managing the most valuable asset in many organizations—knowledge. Important aspects of knowledge management involve organizational, technical and sociological factors. Information technology is not the centre of knowledge management—it is an enabling component for implementing knowledge management systems that can help increase the level of knowledge sharing and reuse.

O'Leary (1998) states that the overriding purpose of enterprise knowledge management is to make knowledge accessible and reusable to the enterprise. It entails formally managing knowledge resources in order to facilitate access and reuse of knowledge. Knowledge resources can be, for example, manuals, letters, news, customer information, process knowledge, competitor intelligence etc. These resources are sometimes formalised in a system with defined categories and terms allowing search of knowledge. Knowledge resources can also be informally represented as documents, for example, in groupware systems, e-mail, intranets and databases.

Information technologies used in knowledge management systems include Lotus Notes, e-mail, expert and knowledge based systems, intelligent agents and data warehouses. Internal and external news can be published on the corporate intranet and previous problems and solutions can be collected in computerized form—for example the "Things gone right/Things gone wrong file" at Ford. Other technologies are search engines and ontologies that define the terminology of the organization to facilitate search of relevant information. However, knowledge technology has further potential in knowledge management as observed by O'Leary (1998): *"We need additional research to expand the use of artificial intelligence and knowledge based systems in KM. We need to know what forms of knowledge representation appears to work best for particular types of knowledge."*

Reasons for managing the knowledge of an organization include the recent trend of downsizing, faster rate of employee turnover, increased pace of change in technologies and innovation and international expansion of companies (O'Leary, 1998). Another trend is that a larger portion of a company's value lies in the application of its knowledge to the value chain, rather than its resources—the company becomes knowledge based, as for example consulting, or knowledge enabled, as for example manufacturing (Harris, 1998). In either case the success of the company depends on the quality of its knowledge. Therefore it needs to be managed and kept within the company.

Downsizing means cutting the workforce to let fewer employees (but key to the process) do the same work. In this process the number of middle-management and staff function people is often reduced. These people often act as coordinators and locators of the right knowledge for different tasks. People that have been a long time in the organization remembers what worked well and what did not in previous projects and who was best for doing a particular type of task.

Increased employee turnover means that the risk of losing critical knowledge increases. A knowledge management system attempts to capture at least some of that knowledge by encoding it in a form accessible to others in the organization.

International expansion of corporations with people spread over the world aggravates the classical problem of communicating and locating the right knowledge for the current task, as well as increases the complexity of the operations. Organizations are realizing the importance of knowing what they know and to create an "organizational memory" (Macintosh, 1999; Kühn and Abecker, 1997; Dieng, 1999).

1.1.1 Knowledge, Data and Information

A distinction between knowledge, information and data is useful as a start for understanding the knowledge management problem. Several definitions of these concepts can be found.

Davenport and Prusak (1998) describe data as *"... a set of discrete, objective facts about events"*. For example, when a customer buys gasoline at a gas station, the time of purchase, the number of gallons and the price are all examples of data. Data becomes information when it is presented as a *message* that makes a difference to the receiver of the message. In other words, data becomes information when some *meaning* is added to it. Methods for adding meaning are for example *contextualizing, categorizing, calculation, correction, condensation*. For example, contextualizing some data can be done by adding the purpose of why the data was gathered.

Knowledge is said to be transformed from information by humans through methods of *comparison, consequences, connections, and conversation.* When a person e.g. makes a comparison of some information with another situation previously encountered, knowledge is derived (Davenport and Prusak, 1998).

Knowledge is an elusive concept to define as can be seen from the working definition that Davenport and Prusak offers:

> *"Knowledge is a fluid mix of framed experience, values, contextual information, and expert insights that provides a framework for evaluating and incorporating new experiences and information. It originates and is embedded in the minds of knowers. In organizations, it often becomes embedded not only in documents or repositories but also in organizational routines, processes, practices, and norms."*

Brooking (1999) distinguishes data, information and knowledge somewhat differently: "Data*: Sequences of numbers, letters, pictures, etc. presented without a context.* Information*: Organized data, tables, sales statistics, a well presented talk when presented in context.* Knowledge*: Organized information together with understanding of what it means."*

Both definitions build on a hierarchical view of the relationship between the concepts, where some value is added to the lower level concept to get the higher

level concept. The added value concerns understanding something *about* the concept—a meta level is added. For example, when we understand the purpose of collecting the data, we get information rather than just data.

The Concise Oxford Dictionary (1982) offers the following definition of knowledge: *"1. knowing, familiarity gained by experience. 2. person's range of information. 3. theoretical or practical understanding; the sum of what is known."* The definition of information reads: *"informing, telling, thing told, knowledge, (desired) items of knowledge, news"*. From these definitions we can also see that some value is added when we move from information to knowledge, in particular a person's perspective on the information. However, information is also defined as knowledge, so the concepts are clearly close.

1.1.2 Knowledge Management Processes

Once we have understood what knowledge concerns and why it is important to manage it, we want to understand the process for managing it. The knowledge management process can be described in several ways. In this section we review the knowledge management process as presented by Davenport and Prusak (1998), Nonaka (1991) and the Gartner Group (1998).

Davenport and Prusak

Davenport and Prusak divide the process of Knowledge Management into the activities of generation, codification/coordination and transfer of knowledge.

Knowledge Generation. Davenport and Prusak describe five modes of Knowledge Generation: acquisition, dedicated resources, fusion, adaptation and knowledge networking.

Acquisition is described as knowledge that is imported into the organization from outside sources, for instance by stealing or buying knowledge, employing individuals with knowledge, acquiring organizations with knowledge. Other means include support to research organization for first right to research results, and knowledge transfer from leased consultants with special knowledge. *Dedicated resources* involves putting up special groups, such as research and development groups, for the purpose of generating knowledge. *Fusion* is a way of creating new knowledge in the organization by bringing people with different viewpoints and experience together to spark new thinking. *Adaptation* means to adapt to new circumstances by being open to new innovations and actively being on the lookout for new knowledge and skills. *Knowledge networking*, finally, involves informal, self-organizing networks consisting of people with common interests.

Knowledge Codification and Coordination. Davenport and Prusak describe the aim of knowledge codification as to put organizational knowledge into a form that makes it accessible to those that need it. It means to transform knowledge into a code that can be understood by people or machines, such as texts or computer programs that can illustrate the knowledge for people. We can categorize knowledge, describe it, map and model it, simulate it and embed it in rules and

recipes. Codification can be taken to very different lengths depending on the character of the knowledge. For example, codification of tacit, complex knowledge, internalized by the knower over a long period of time, is very difficult and may be limited to listing someone in the organization that has the tacit knowledge. At the other end of the scale, knowledge can be codified as a computer program that simulates a process using the knowledge.

Davenport and Prusak state that codifying knowledge is an essential step in leveraging its value in the organization, However, they also feel that the challenge lies in codifying knowledge in structures that allow rapid and flexible change of the knowledge, because knowledge itself changes rapidly and flexibly.

Knowledge Transfer. Knowledge transfer often involves human interaction. The most efficient way of transferring knowledge, according to Davenpor and Prusak, is giving people time to meet and talk to each other. Examples are chance meetings at the water cooler, formal knowledge fairs with much time for people to talk informally, corporate picnics, assignees working at the research facility for a limited time and mentoring programs.

Davenport and Prusak observe that knowledge that is more or less explicit can be represented in documents, databases and transferred with reasonable accuracy. Tacit knowledge on the other hand usually needs personal interaction.

The term tacit knowledge was introduced by Polanyi (1957; 1958) who wanted to include a personal dimension in the process of scientific discovery and validation, as opposed to the objective ideal of scientific detachment. He argued that a scientist's work involved not only explicit, objective knowledge, but that it had a tacit component—a personal coefficient, which influences the knower in making choices in the scientific work. This personal background knowledge guides the scientist forward with a sense of what feels like the right theory or explanation of phenomena. The tacit coefficient is generally not articulated but remains in the background influencing the scientific process. The tacit component is also involved in skills, for example in swimming or finding the best place for drilling an oil well.

In this thesis, we use the term tacit knowledge in the limited sense of unstated knowledge, part of an expert's performance, such as heuristics, preferences and strategies. Many parts of tacit knowledge may be impossible to articulate, but we believe that some limited parts can be articulated.

If an organization is distributed and the people in it have little time to meet, we think that personal interaction should be complemented with more structured and formal ways of transferring knowledge. The key word here is representation—if knowledge can be represented explicitly, then it can also be transferred through this representation, provided that the representation is understandable to people.

The Knowledge Spiral

Nonaka (1991) suggests that knowledge needs to managed differently than quantifiable data in a knowledge-creating company. Softer methods are needed for tapping the tacit insights, intuitions and ideals of employees, such as making use of slogans, analogy, metaphors and constantly challenging people to re-examine what they take for granted. The methods should be applied in an organization with

redundancy in personnel and free access to information to be the basis for the knowledge management process.

Nonaka suggests four distinct patterns of knowledge creation in any organization, based on the distinction between tacit and explicit knowledge:

1. From Tacit to Tacit: *socialization*, for example learning baking bread by observing and trying.
2. From Explicit to Explicit: *combination*, for example collecting data to a financial report.
3. From Tacit to Explicit: *articulation*, for example developing a new approach to budgetary control based on long experience.
4. From Explicit to Tacit: *internalization*, for example a new budgetary control approach causes new behaviours.

Nonaka believes that these four patters of knowledge creation interact dynamically as a *Knowledge Spiral* where an organization builds on tacit knowledge to create new explicit knowledge, that in turn creates new tacit knowledge at a higher level.

Nonaka exemplifies the knowledge creation process with the development of the home bread-making machine. The developers at Matsushita were hard at work developing a home bread-making machine. However, they were having difficulties with getting the machine to knead dough correctly. The outside of the bread was always overcooked but the inside was not ready at all. Finally Ikunu Tanaka suggested going to the Osaka International hotel to learn from the master baker. After a year of learning and testing new techniques they came up with the "twist-dough" method for the machine to replicate the stretching technique used by the master baker. The knowledge spiral in this case is described as:

1. First, the tacit knowledge of bread-making is learned in interaction with the baker (socialization).
2. Next, the tacit knowledge is translated into an explicit form that can be communicated to other members of the team (articulation).
3. The explicit knowledge is standardized into a manual or workbook together with previous product development knowledge, that is used to produce a really good bread-making product (combination).
4. The members of the team creating the better product using the new knowledge increase their own tacit knowledge of product development with the knowledge that products like bread-making machines can provide genuine quality.

Gartner Group

The Gartner Group categorizes Knowledge Management into *Knowledge Creation*, *Knowledge Sharing* and *Knowledge Use*. Knowledge Sharing is further categorized as *Capture*, *Organize*, *Display* and *Access*. In Fig. 1–1 these categories describe the knowledge management process in the view of the Gartner Group.

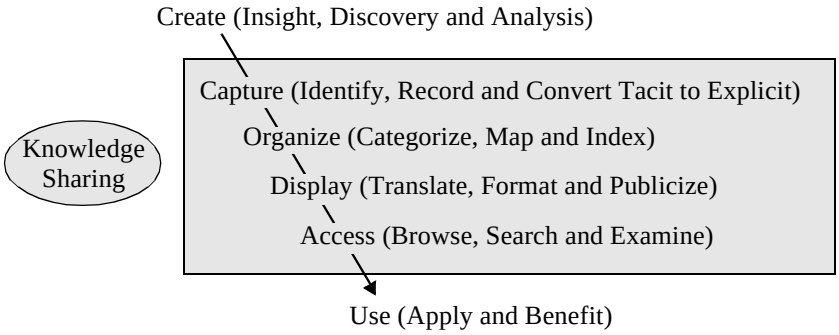

Fig. 1-1 *Gartner Group Knowledge Management Process Framework,* Adapted from Harris (1998).

1.1.3 Why is Knowledge Management Difficult?

In the previous section we see that the knowledge management process can be described in somewhat varying terms, but three main activities can be identified in each account: Creating new knowledge, representing knowledge, and making it available to other people. With this unified view of the knowledge management process we can see that the difficulty with knowledge management does not lie in understanding the fundamentals of the process. The problem seems to lie in making the process work—getting the details of the process right. For instance, making people willing to participate in the process, giving people time and opportunity to participate, and representing knowledge.

Davenport and Prusak (1998) report several cultural and social factors that inhibit knowledge transfer:

"- Lack of trust

- Different cultures, vocabularies, frames of reference

-Lack of time and meeting places

- Status and rewards go to knowledge owners

- Lack of absorptive capacity in recipients

- Belief that knowledge is prerogative of particular groups, not-invented-here syndrome

- Intolerance for mistakes or need for help"

The possible solutions suggested central on cultural and social factors:

"- Build relationships and trust through face-to-face meetings

- Create common ground through education, publications, teaming, job rotation

- *Establish time and places for knowledge transfers: fairs, talk rooms, conference reports*
- *Evaluate performance and provide incentives based on sharing*
- *Educate employees for flexibility; provide time for learning; hire for openness to ideas*
- *Encourage non-hierarchical approach to knowledge; quality of ideas more important than status of source*
- *Accept and reward creative errors and collaboration; no loss of status from not knowing everything."*

In our view, by building systems that represent knowledge that can be inspected, these cultural and social solutions can be supported. A system containing knowledge that can be inspected and evaluated will be more trusted than a system whose knowledge is opaque. A system will save time in collecting knowledge—time that can be used for personal interaction, discussing the knowledge. A system that people can contribute knowledge to can be a way to promote sharing. A system does not need to reveal the source of the knowledge in the system. The fact that it has been allowed in the system would be the voucher for its correctness or it could advertise the contributor to gain authority. A system that contains knowledge for helping people will convey the message that it is natural not to know everything.

Davenport and Prusak describe the goal of knowledge transfer as to improve an organization's ability to do things, and increase its value. This goal is not reached if the knowledge being transferred is not put to use. They illustrate this with the following expression:

Transfer = Transmission + Absorption (and Use)

For transfer to take place, knowledge needs not only be sent to a recipient but also be absorbed and put to use. If we design representations of knowledge that can be understood, we improve the chances that the knowledge will be absorbed. Moreover, if these representations can be used by the computer for solving a task, we increase the likelihood that the knowledge also will be put to use.

1.2 KNOWLEDGE TECHNOLOGY

Knowledge Technology has dealt with the difficulties of capturing knowledge, especially expert knowledge, for a long time. The areas of Expert Systems, Knowledge Acquisition, Knowledge Based Systems and Artificial Intelligence in general have gathered experience in the difficult tasks of acquiring, representing and structuring knowledge and reasoning with it.

Difficulties in knowledge acquisition include: It can be difficult to capture knowledge because people may be reluctant to contribute knowledge or because they are unable to formulate their knowledge. Knowledge acquisition is generally time consuming and therefore expensive. Ideally people should themselves be able

to contribute knowledge, as part of their work process. It is difficult to formulate expert knowledge because there may be gaps in the knowledge, inconsistencies and people may know more than is apparent.

Knowledge Acquisition has developed methods for collecting and structuring knowledge that have been reported useful also for knowledge management (Milton et al., 1999):

– Laddering: Creating hierarchies of knowledge elements such as concepts, attributes, processes and requirements.

– Card sorting: Grouping of objects or concepts into classes, using the metaphor of sorting cards into piles

– Repertory grids: Using personal construct theory to identify attributes in a domain and grouping objects or concepts according to how similar they are with respect to the attributes.

It is generally difficult to communicate informal text-based knowledge, because it is ambiguous—open to interpretation, and often lack overview and structure, so inconsistencies can go undetected. Moreover, it can be difficult to put to use, because there is a lack of control structure and it needs to be internalized by a person before it can be put to use. Knowledge acquisition tries to deal with this problem by structuring knowledge with the goal of representing knowledge formally.

Knowledge representation concerns the problem of designing formal representations of knowledge suitable for inference such that we in a mechanical way can arrive at new assertions based on inference rules and factual knowledge– automatic inference. Many knowledge representation languages and inference techniques have been suggested: rules, frames, semantic networks, decision tables, meta-representations, forward chaining, backward chaining, constraint reasoning, non-monotonic reasoning, probabilistic reasoning, etc.

In our view, knowledge technology can offer knowledge management methods and techniques for acquiring, representing and using knowledge. Knowledge technology enables intelligent applications of the knowledge that an organization wants to manage. Knowledge should not only be managed for use by people, but also for use by intelligent applications.

Knowledge technology can also offer knowledge management an incentive for people in the organization to contribute knowledge. If people know that their knowledge actually will be used in an application they may be more willing to share their knowledge.

1.3 APPLICATIONS

In this section we briefly present a number of example applications where we have tried to represent knowledge in such a way that it can be managed and understood by people, but also used for inference by a machine. The examples range from diagnosis and simulation to scheduling. The main goal has been to design representations that are suitable for automatic inference but also understandable for

people to make management of the knowledge in the applications easier—an explicit representation of knowledge that will give an overview and structure of the knowledge.

In paper I we design a representation of diagnosis knowledge that represents a device in an explicit and general way that allows the user to add new components and inspect the function of the device. The representation allows an integration of model based knowledge with compiled and heuristic knowledge so that the device and its function can be represented at a suitable level of abstraction and let other parts be represented as non-model based knowledge. Heuristics are represented as rules. An explicit reasoning strategy is designed that uses the model to find possible diagnoses and the rules to direct the search for diagnoses. A metalevel representation for the rules and the strategy allows an integration with the model based parts.

The suggested representation is a foundation to allow a user access and modify the diagnosis knowledge. This is based on a visual illustration to manipulate the knowledge. For example, components and connections could be added by drag-and-drop from a set of general components, where general definitions of the components would be added to the model. The user can also interact with the visual illustration when using the diagnosis system.

In paper II we use simplified rules for describing the time, resources, activities and amounts required in a tunnelling project and a simulation engine for estimating time and amounts consumed in the tunnelling process. The rules are designed to allow a user to change the facts and computations of the system. A graphical view illustrates the progress of the tunnelling simulation.

In paper III we present the constraint modelling language CML and show how to model a nurse scheduling problem and a train scheduling problem without programming. The idea is to preserve the problem structure of the domain, allowing constraint formulations that reflect natural language expressions familiar to the users. CML problem specifications are transformed automatically to standard constraint programs, providing the potential for efficient problem solving. By representing the constraint problem in a modelling language the user can manage the knowledge of the problem and produce solutions for different sets of constraints.

In paper IV we investigate the use of decision tables for representing requirements on staff scheduling explicitly, providing structure and overview to the user. The requirements are compiled automatically to a program that use hand-written procedures for efficient scheduling. The format of the decision table gives overview of the knowledge and sub-tables provide structure to the knowledge. Inconsistencies and incorrectness can be detected by the user. A computer can run the table on previous explicit knowledge, available in databases, to produce a schedule. The user consults the tables to state problem data for each new schedule and can receive support in the tables on deciding the values.

Chapter 2

Summary of the Papers

2.1 INTEGRATING HEURISTIC AND MODEL-BASED DIAGNOSIS

A model-based diagnosis system has a model of the diagnosis object, which describes the construction and function of the object. It then applies a general diagnosis algorithm to its model to find a diagnosis. The model can be a model of correct function, as in (de Kleer and Williams, 1987), or a fault model describing possible ways of incorrect function, as in (Console et al., 1989). An other type of diagnosis systems, such as MYCIN in (Davis et al., 1977), uses heuristic knowledge for making diagnoses. The heuristics mainly consist of empirical associations from symptoms to causes. We want to take advantage of knowledge from both of these kinds of systems, but also keep them separate with the aim of letting the user manage (at least parts of) the knowledge of the application.

This paper studies the problem of integrating these different kinds of knowledge into a single diagnosis system. The purpose is to represent problem solving knowledge such that its logical structure is preserved, it is represented in a modular fashion allowing parts to be replaced or modified without affecting each other, the representation is transparent enough for a knowledge provider to distinguish different kinds—with the aid of a suitable interface—and the representation allows them to be used together.

A modular and transparent representation is a foundation for knowledge management of the diagnosis system. In the paper we propose a visual interface for manipulating the system, for example by adding components or connections. The interface can be further developed to let the user modify the rules and the definitions of diagnosis knowledge. This is a topic for further research.

We propose a metalogic representation structure in which we separate the different kinds of knowledge to preserve its logical structure and to make it possible to revise and edit. Sentences in an object theory represent the domain knowledge for a device to be diagnosed, while sentences in a metatheory represent the diagnosis strategy, heuristic knowledge and control knowledge. The communication of computations of the object theory is performed by a predicate *demo* that relates the object theory and names for sentences provable in the object theory.

Our approach to diagnosis of a device is based on a comparison of an actual (incorrect) device and a representation of an ideal (correct) device. The structure and function of a correct device is formalized as an object theory, OT, itself an object of discourse in a metatheory, MT. Knowledge on forming hypotheses and conclusions are formalized in MT. In this theory heuristic hypotheses and conclusions about the system are defined in terms of the theorems of OT and knowledge about the device. The metatheory also formalizes a diagnostic strategy which studies the hypotheses and conclusions of MT in the diagnosis of the device. Knowledge about the real world, as perceived by the end user, is represented in MT.

The kind of diagnosis domain that the theory is designed to describe is a system with a set of components connected with connectors, through which some directed flow is transported. The components display various observable behaviours and measurable flows. The system is controlled through various settings of the components. We have implemented an example system for diagnosing a simple device—a stereo system—where we have integrated model based knowledge with heuristic knowledge.

The object theory consists of two parts: (1) definitions of the structure of a system and (2) definitions of the function of a system. The structure definitions fall into two categories: (a) definitions of the objects of the system and their subparts, and (b) definitions of how the objects are connected to each other.

The objects of a system can be connected in different ways—different system configurations. For example, the loudspeakers in a stereo system may be connected in different ways to the amplifier. We represent this in `configuration` as a relation between a system `Sy` and a configuration `C` of components, connectors and external objects.

The main definition of function, `output`, builds on two definitions: `input` and `transformation`. The first characterizes the inputs to the objects and the second characterizes the measurable output signals and the observable behaviours of the objects in the domain. An output signal could be a voltage signal of an output port on an amplifier and a behaviour could be that the on-light of the amplifier is on.

The input signals to an object depend on the configuration of the system and the settings of the objects. For example, the cable connections of a stereo system determine what inputs the stereo components receive; the settings of an amplifier

determine what inputs a loudspeaker receives. We represent this as a relation `input` between an object `X`, a system use `U` and a list `I` of inputs. We state a clause for each type of object in the definition.

The inputs are *transformed* into outputs. We state transformation rules for the components in order to represent the function of a system correctly. Transformation of input signals to output signals and behaviours are represented as a relation `transformation` between a list `I` of inputs, an object `X`, a system use `U`, and a list `Out` of output signals and behaviours. We state a clause for each type of object that has its own kind of transformation rules for its input signals. The `out_signal` relation states transformation rules for subparts of objects. For example, the transformation rules for an amplifier are stated over its output speaker ports.

In the diagnosis system, we deduce consequences of the theory to diagnose malfunctions. We represent only correct function and behaviour in the object theory, whereas the metatheory contains knowledge about faulty function and behaviour. Therefore, all consequences of the object theory should be observable in the real world and any that is not, would indicate a fault. We call observations that correspond to the central theorems of the object theory for *normal* observations, and, consequently, those that are not normal for *abnormal* observations. We give logic programs for *normal* and *abnormal* as relations between a theory and an observation term.

We take advantage of the object theory when we generate hypotheses for a fault (an abnormal observation) reported by the user. The formalization tells us which objects influence the output observed by the user, so by studying the object theory hypotheses for the fault can be generated. We analyse the object theory to see what could have caused the fault. To aid the analysis we use the concept of signal path. This can be understood as a sequence of objects that a signal traverses in order to construct an output from an object in the diagnosis domain, such as a sound from a loudspeaker. The signal graph for an output is constructed by a metalevel analysis of the object theory. We have implemented a meta-interpreter *solve_construct* for the analysis of an object theory. The hypotheses are dynamically generated in relation to an object theory, so if the object theory is modified in restricted aspects, for example, in what objects there are, or what their connections are, there would be no need to alter the program for the generation of hypotheses.

We distinguish two methods for investigating hypotheses, based on whether or not they rely on the model of the diagnosis domain. The first uses the model of the diagnosis domain to investigate the hypotheses. This method is called *object level refutation*. It compares statements of the object theory regarding structure and behaviour of the diagnosis domain with the user's observations of the real world. If the observations confirm the statements, then the hypothesis is refuted. The relation between a hypothesis and the relevant statements is formalized as a metatheory program, *refutation*, between an object theory *T*, a list *L* of names of statements of *T* and a hypothesis, such that the statements derivable from *T* is a refutation of the hypothesis. The last step of the refutation method is to check if the statements of the object theory can be observed in the user's malfunctioning diagnosis domain. If the

statements are confirmed by the user, then the hypothesis is refutable, i.e. it does not identify the fault and can thus be removed from consideration.

The second method is called *metalevel refutation* because hypotheses are eliminated on the grounds of knowledge formalized in the metatheory. In its basic form it does not use the model of the diagnosis domain to investigate the hypotheses. The metatheory is based on diagnosis experience in the diagnosis domain, knowledge that is not formalized in the object theory either because it is heuristic or because it would require a more detailed formalization than is desired in the object theory.

Heuristics compile knowledge of structural, functional and experiential character into a compact form. We want to use domain specific heuristics because they can be very powerful for problem solving, as a guidance for the use of the model-based knowledge in the object theory.

The second kind of compiled knowledge has a complexity that surpasses the level of abstraction of the domain theory. Such knowledge is inconvenient to represent explicitly in the object theory. The reason is that an explicit representation would force the level of abstraction lower than that chosen by the designer. In the design of a domain theory one must choose a level of detail that is not too difficult and expensive to represent and has an acceptable computational complexity. The mechanisms behind the compiled knowledge may also be difficult or not completely understood. Therefore, we represent it directly as compiled knowledge in the metatheory.

With a metalogic approach to the representation of diagnosis knowledge it is also possible to represent generalizations of compiled knowledge. Our approach is to generalize compiled knowledge by stating it relative an object theory so that the knowledge is general over some class of object theories for different systems.

For example, we can draw a set of conclusions in the stereo domain from the fact that we have a normal observation. We can draw the conclusion that the cable between that component and the power supply, as well as the connections between the component and the cable, are OK, because a component with a normal signal or behaviour requires electricity. This is represented as rule in the meta theory.

A more advanced example of how complex compiled knowledge is formalized as metalevel reasoning involves different object theories. This makes it possible to write programs that reason hypothetically over distinct object theories. If we have a relation *modify_theory* between a theory *T1*, a set *Obs* of observations and a theory *T2* where *T2* is the result of modifying *T1* such that the observations in *Obs* are statements of *T2*, we can reason with different object theories. For instance, if two observations *O1* and *O2* are abnormal for a theory *T1*, but *O2* is normal for *T1* modified for *O1*, then we can draw the conclusion that *O2* is a result of *O1* (i.e., *O1* explains *O2*), and therefore exclude the possibility of an internal fault in the component that *O2* mentions.

Metalevel reasoning where an object theory is regarded as a term in the language gives us an approach for writing declarative and powerful reasoning programs. Moreover, we find in the *modify_theory* program a modular representation. It is modular because the program is general over different theories,

so it works even if we replace the theory *T* for another theory *T'* with a different set of stereo components or components of some similar technical system. So, despite the fact that we represent complicated compiled knowledge we have the means to preserve a certain generality of the knowledge, to allow a knowledge provider to manage the knowledge.

Heuristic compiled knowledge is not allowed to give us conclusions—it only guides the diagnosis process. The methods of object refutation and meta refutation give conclusions regarding the hypotheses of the diagnosis and this is correct because they refer to knowledge that is correct in the domain. For meta refutation, complex compiled knowledge is represented as a set of clauses in the relation *meta_refute*. We represent heuristic compiled knowledge in a similar fashion because the kind of knowledge to be expressed is rather similar to complex compiled knowledge, the difference being that heuristics can only be taken as recommendations. We represent heuristic compiled knowledge as a separate relation, *probably_refutable*. With this we get a recommendation of which hypotheses to test. Each hypothesis in the list of hypotheses generated by the program could be tested for the property of *probably_refutable* as an indication of its potential for refutation.

The basic idea in the diagnosis strategy is to ask the user for an object theory and an observation of some faulty behaviour (or signal), for example that a loudspeaker does not produce any sound and then generate the hypotheses. The strategy then tries to refute (eliminate) each hypothesis, first trying heuristic knowledge and then falling back on model based knowledge, until a hypothesis is found that cannot be refuted. This hypothesis will then be presented as the answer of the diagnosis, together with the user's observations and the object theory.

We have a two-dimensional graphical display as a basis for a user interface to provide the user with a visualization of the domain, in which the user can manipulate different objects in the domain. In this display the diagnosis system can express questions to the user by displaying a box, containing a question, pointing to the object of the stereo system the question refers to. The user can answer the question positively by clicking on a "yes-button" and negatively by clicking on a "no-button".

This graphical display could be the basis for managing the knowledge of the system. For example, new components could be added by drag-and-drop from a tool-box of components. The representation of components as general objects would facilitate this process. Furthermore, the separation of model based knowledge, compiled knowledge, heuristic knowledge and strategy also make this easier. A graphical management of the compiled and heuristic knowledge requires a more advanced interface. However, the fact that this knowledge is explicitly represented in a separate theory, instead of being interwoven with the model knowledge, facilitates its knowledge management.

2.2 TUNNELLING ANALYSIS AS PLANNING AND SIMULATION

A tool for analyzing the process of tunnelling has been developed. The tool takes a representation of the tunnelling parameters such as geography, geology, type of tunnels, methods of constructing tunnels and simulates the process using a constraint based simulator developed for this representation. The planning and analysis of a tunnelling project is a very complex task, for which there is no standard procedure. The process is not only time consuming, but also requires expert knowledge in all aspects of tunnelling.

We represent this knowledge as a set of activities, resources, time rules, amount rules and stochastic time data separately from the simulation engine. This representation allows a knowledge provider to modify the rules and the data in order to model different methods of tunnelling and different sets of time data.

The goal of a tunnelling project is to construct tunnels in one or more directions—building a subway is one example. Typical activities are drilling, charging, blasting, etc. The duration of an activity, such as drilling, is not known in advance; it varies stochastically. The quality of the rock is of crucial importance and is often researched for different parts of the project, but the duration of drilling a number of holes may still vary. These variations are significant for the total duration, since a project may span a long time period, perhaps a year with 50000 activities taking place.

The tunnelling problem seems first like an ordinary scheduling problem involving disjunctive constraints, such as the well-known Bridge problem (Dincbas et al., 1990). However, tunnelling is neither an ordinary scheduling problem nor an optimization problem. Optimization is not desirable, since an optimal plan would not reflect how the project is carried out in reality.

Activities are stated as constraint logic programming (CLP) facts with four arguments. The first is its name, the second is the front where it occurs, the third is the rock class and the last argument is a list of machines. Precedence constraints are also stated as CLP facts, ordering the activities on each front. The last kind of constraint states that two activities are interfering and therefore must be separate in time. Typically, one should not have an explosion on one front too close to people working on another front. There are many more facts to be stated, such as rock classes for different parts of the tunnel fronts, the number of holes to drill, transfer speeds of machines, etc.

The duration of each activity depends on several subactivities captured in time rules. In addition, the durations of these subactivities vary stochastically. The drilling activity, for example, consists of mobilising (preparing) the drilling equipment, initiating the drilling of a hole, drilling the hole, moving the drill to the next hole, completing all the holes and demobilising (removing) the drilling equipment. The duration of an activity is calculated by a time rule that takes all its subactivities into account.

Since the activities of a tunnelling project are repeated so many times, variations in their durations must be taken into account. In the tunnelling domain, data from previous projects are known by the domain expert. By collecting

observations of time spent on, for instance, the different phases of drilling a hole we construct distribution diagrams that tell us how the duration of an activity is distributed over an interval of time. For instance, initiating the drilling of a hole may be distributed over 10–55 seconds, with 24 seconds being the nominal (normal) case.

We use a random number generator and a distribution function to calculate a random duration from the random value. The duration is adjusted by the nominal duration that the expert wants to apply to this particular case to compensate for difficult conditions. For instance, the conditions of the rock may necessitate a compensation of 10% to the normal duration given by the observation data.

We also need to compute plausible amounts of material and manpower. The amount rules use facts about the numbers of trucks and persons at the current front and rock class to compute the amounts spent during the activity. These amounts, and many others, will be accumulated for the whole project.

With all the relevant facts stated, planning begins. The system will choose a front to schedule and then perform a full planning cycle on this front. That is, all activities to be performed in the current rock class are collected and scheduled, following the precedence constraints. If an activity cannot overlap with an activity of another front, because they share a machine or are stated as interfering, a disjunctive constraint is set up using the standard formulation. The planning continues on the rest of the fronts with a full planning cycle. The planning stops when the required distance of tunnelling has been covered.

A single tunnelling project plan represents one possible outcome in terms of time and material spent, totally and divided into parts of the tunnel fronts, when knowledge is combined with chance. By repeating the planning, say thirty runs or more, we obtain a worst case, a best case, a mean value and other statistical measures.

The system has been validated by a tunnelling expert at Atlas Copco on the Stockholm South Link, a project in which tunnels are built for road traffic in the southern part of Stockholm. The time estimations and analysis results have been judged realistic in tests with different configurations of equipment. Our model could very well be used for a standard scheduling problem and, more interestingly, it is scalable to very big problem instances, due to a linear growth of the execution time with respect to the problem size.

The end goal with a complete system is to distribute expertise to the sales offices throughout the organisation, giving them the capability to perform expert configuration of tunnelling projects interactively with the customers. A graphical interface to the knowledge needs to be implemented. The business process of tunnelling analysis starts with layout and configuration, followed by simulation and evaluation of the results. This iterative process is completed when the construction company and the supplier eventually agrees on a suitable configuration for the project. By giving the sales engineer access to a simulation tool, the company uses its collective knowledge to improve the sales process. The simulation tool would be a central resource in managing the company’s knowledge of tunnelling.

2.3 MODELING CONSTRAINT PROBLEMS IN CML

A constraint modelling language CML is presented allowing a non-programmer to specify a constraint problem with problem specific constraints as simple expressions in a declarative language. The specification of the constraint problem is compiled into a constraint logic program that is able to compute a solution to the problem. CML is similar in spirit to SQL (ISO, 1999) where declarative queries over a set of relations can be stated without regard to the physical implementation of the data.

Constraint programming languages such as CHIP V5 (Cosytec, 1996) or ILOG (ILOG, 1996) provide programmers with the means for writing declarative constraint programs. Users of constraint programs may be allowed to change some pre-defined parts of the programs, for example, to add facts or to modify constraints. Major changes to the program, such as defining new constraints, are normally left to the programmer. This inevitably results in a gap between the problem description and the program. CML is an attempt to reduce this gap by providing the user with a more general interface to constraint programming. The goal is to develop tools that allow skilled people, for example, hospital administrators, to define and modify classes of constraint problems at a higher level of abstraction, with constraint programs as the result.

The paper presents two examples of using CML to specify constraint problems. The first example is a nurse scheduling example where the work schedule of nurses is constructed taking into account constraints for working regulations and preferences of nurses. The second example is a train scheduling example, where a number of trains are scheduled on a single line track, and the constraints are that the same line between stations can not be used by trains travelling in opposite directions and that trains travelling in the same direction must respect a certain margin.

The same basic structure and form of constraints are used for both examples despite that they are quite different, indicating that the CML formalism is general over different domains. In compiling the specifications to constraint programs no extra knowledge about the domain is used—only syntactical transformations are done. The compiled program consists of primitive constraints that defines the problem.

CML is based on a general assignment theory, AT (Hjerpe, 1995) that is used in compiling the CML expressions into a constraint logic program. The idea behind this theory is that a constraint problem can be specified in terms of demands on a solution structure. Specific problems are represented by instantiating meta variables in schemas, thereby stating the resources, the solution structure and the constraints. The CML constraints are simplifications of first-order formulas using logic connectives and universal and existential quantification of variables. The constraints correspond to the demand in AT that a solution, must satisfy all constraints.

We use AT to decide, model-theoretically, which parts of a solution structure (that is, which domain variables) to use in constraint programs and the fact that the domains are finite to rewrite the AT constraint formulations as conjunctions

and disjunctions. The constraint programs use primitive constraint formulations with #=, ##, #>, #>=, #< and #<=. We have implemented a compiler that automatically transforms specifications into standard constraint programs. Naturally, one wants to use the power of global constraint formulations, such as the cumulative constraint. Further work remains to construct an automatic transformation of CML problem specifications to constraint programs which use global constraint formulations.

The idea of CML is to preserve the problem structure of the domain, allowing constraint formulations that reflect natural language expressions familiar to the users. The main potential lies in large-scale use of constraint technology through tools. Based on CML and database technology, domain specific tools can be developed to increase the productivity of constraint technology.

All CML constraint specifications are independent of each other, so that any constraint can be modified without reference to other constraints. Only the solution structure that sets up the basic problem specification needs to be modified if, for example, more nurses need to be scheduled. If a new legal regulation needs to be added, a new constraint is simply added. This ability provides the foundation for a knowledge provider to understand and validate the constraint specification and to modify it with new knowledge, giving the organization the ability to manage the knowledge that is actually used for producing schedules.

2.4 KNOWLEDGE MANAGEMENT OF CONSTRAINT PROBLEMS IN DECISION TABLES

We present a method, DT-CML, for integrating decision tables with constraint logic programming, as a knowledge management and software engineering tool, making the power of constraints available to non-technical people. By managing the constraint knowledge in decision tables integrated with constraint logic programs the knowledge becomes visible and modifiable by people in an organization.

Standard decision tables are used as a fundamental representation for constraints and extended with a solution structure that represents the problem data. The basis for specifying constraints is the same structure as in CML, discussed in the previous paper, but the user does not need to know the details of the structure. The user specifies the constraints in decision tables and a programmer writes a constraint procedure over the solution structure corresponding to the specification of each constraint in the decision tables. The procedure will have open parameters that the end user states when using the decision tables to solve a specific instance of the problem. The decision table formalism is used as a problem specification and knowledge management tool.

As an example, staff requirements for scheduling can be specified in DT-CML at an abstract level in decision tables, together with knowledge in the decision tables that can support the user in selecting the correct requirements. For example, we may have a hospital policy stating that the number of doctors and nurses to be on duty is dependent on the number of patients expected and quality of service factors, such as maximum waiting time. We may also have legal or union requirements such

as the maximum number of work days in a row, perhaps with exceptions. Another class of requirements is desirable requirements, such that no one should work a single day (a working day preceded and followed by a free day). These requirements may be used more frequently in some units than others. When they are specified in decision tables, the user can choose which to use and which not to. The user can also access the factors behind the requirements since they are specified in the decision tables.

The tables have a hierarchical structure, where a decision variable can have a sub table that can be accessed for further decision support. The user sets up the constraint problem by consulting the tables, starting with the main table. Consulting the tables means that each row of the tables is given as a question to the user by the system, in order to assign a value to the decision variable of that row. The user can choose one of the suggested values or consult a sub table for decision support on the value. When the user has set up the problem, a program is compiled that reflects the constraints and the parameters to the constraint chosen by the user. We have implemented this compiler.

The compilation of DT-CML takes as its start the resources available for scheduling together with the solution structure. The resources are coded as integers by the compiler and a finite-domain constraint program is generated. Integer codes of the resources in the domain, basic pre-stored constraints, the solution structure representing the problem data (for example the staff schema), the decision table specification of user constraints and hand-written CLP-code are combined to generate a CLP program that can solve the constraint problem.

We connect the informal specification in the decision table with its implementation as a constraint procedure to transfer the values of the decision table variables to the constraint procedure using a *template*. Each decision table having a goal variable with a constraint attribute specifies a constraint. The goal variables of constraint tables have a template for a constraint call to the constraint procedure. When a table with a constraint attribute has been consulted by the user, a constraint call is generated for the CLP program. In the template the decision variables are replaced by the values chosen by the user for these variables. In addition, terms representing resources are replaced by their numeric codes in accordance with the coding.

A central concern of Knowledge Management is knowledge codification and coordination. Davenport and Prusak (1998) describe the aim of knowledge codification and coordination as to put organizational knowledge into a form that makes it accessible to those that need it. It means to transform knowledge into a code that can be understood by people or machines, such as texts or computer programs that can illustrate the knowledge for people. Nonaka (1991) also point to the need for explicit representation of knowledge. A knowledge spiral of a knowledge creating company is described where a central step is going from tacit to explicit knowledge: *articulation*.

Decision tables can be seen as a tool for knowledge management by providing a representation that can be understood by individuals in the organization. The decision table format helps in providing overview and structure of the knowledge with a graphical illustration. Moreover, knowledge management is very

dependent on people really contributing knowledge. By using decision tables as a tool both for collecting knowledge and actually using it when creating schedules, an incentive for sharing knowledge is created. If you know that your knowledge for scheduling will be used when you contribute it, then you may be more motivated to participate.

2.5 RELATED WORK

In the Second Generation Expert Systems manifesto, Steels (1985) discussed the idea of exploiting more than one type of knowledge in knowledge based systems, for example by integrating heuristic knowledge with some form of deep knowledge. The focus of his approach is on the integration and cooperation at the conceptual level of different types of knowledge. Our approach to diagnosis emphasizes integration and cooperation between the heuristic and the model-based level, but also uses the representation to formulate heuristic knowledge in terms of the model-based knowledge. The meta-object architecture allows us to represent semi-general heuristics at the metalevel in terms of model-based knowledge at the object-level.

The CHECK system (Console et al., 1992) consists of a heuristic level cooperating with a causal level. The heuristic level is invoked first and generates a set of hypotheses to be discriminated (or confirmed) by the causal level. The causal level tries to find a part of the causal network that covers the observations. The authors discuss the problem of inconsistency between the two types of knowledge. In particular, it is difficult to verify that they are consistent if they are acquired in different knowledge acquisition processes. In the AID system (Console et al., 1992) compiled knowledge is used as a focusing component for the causal reasoner. So called “rule-out” conditions are compiled from the causal model that allows the system to safely prune the search space for the abductive solver. By considering necessary conditions associated with states (having a cause) in the causal model, it is possible to immediately exclude a state when its necessary condition is inconsistent with the data. To compare, we generate hypotheses in the metatheory. The generator defines hypotheses in terms of model-based knowledge in the object theory, but could also use heuristic knowledge. The diagnosis strategy first tries to eliminate hypotheses based on compiled knowledge before resorting to model-based knowledge in the object theory. The metatheory also contains heuristic knowledge used to rank the hypotheses for discrimination.

Simmons and Davis (Simmons and Davis, 1987) combine causal and associational reasoning in the GORDIUS system in a generate-test-debug paradigm. The system uses associational knowledge to generate an initial hypothesis which is tested and, if incorrect, debugged into a solution using a causal explanation for the failure. Several iterations can be made until a satisfactory solution is found, when the hypothesis passes the test.

Koton’s system CASEY (Koton, 1988) integrates a model-based component (causal model) with case-based reasoning in a system that can make model-based modifications to previous solutions (cases). The case-based component uses the causal model to determine what characteristics are important in an old case when it

evaluates how well the case matches the new current case—essentially those that played a role in the causal explanation of old case. If the case-based component cannot find an old case sufficiently similar to the current so that it can be adapted using the causal model, it falls back on the causal model to solve the case from scratch. The use of model-based modification of old cases have similarities to our use of semi-general heuristics defined in terms of model-based knowledge. They both involve modification of associational knowledge with the help of model-based knowledge integrating different types of knowledge. They both also attempt to reduce the number of inferences needed to find a solution. In Koton's case by modifying an old case supported by model-based knowledge, and in our case by eliminating hypotheses or focusing on some hypotheses with associational knowledge supported by model-based knowledge.

In (ten Teije, van Harmelen, Schreiber and Wielinga, 1996; ten Teije 1997) parametric design is used for automated configuration of problem solving methods using a meta architecture, to build a system with flexible reasoning. The architecture consists of three logic theories: an object theory APPL containing the domain knowledge and problem data for an application such as a causal model, a metatheory METH containing problem solving methods together with definitions of method components, and a meta-metatheory FLEX containing strategy knowledge and a general schema for the problem solving methods in METH as well as relations between components and methods. ten Teije's architecture has similarities to our architecture, in that the model-based knowledge is represented in an object theory and the problem solving methods in a metatheory and they communicate via a reflection mechanism using naming of terms. One difference is that our metatheory contains compiled and heuristic knowledge integrated with the model-based knowledge in the object theory to support diagnosis. Another difference is that we represent the diagnosis strategy in the metatheory and intend a meta-metatheory only to control the instantiating of schemata for building a diagnosis system. A major difference between the architectures is the scope of the systems where ten Teije’s system can capture a number of different notions of diagnosis and furthermore configure problem solving methods automatically for different problem. In contrast our architecture does not currently encompass different notions of diagnosis, although it could be extended in that direction.

Nejdl and colleagues (1995) discuss explicit representation of diagnosis strategies for model-based diagnosis systems. They introduce a metalanguage to express strategic knowledge in an explicit way. In contrast to our approach, which uses standard first-order predicate logic, they employ modal logic. In a related paper (Damásio, Nejdl, Pereira and Schroeder, 1995) a logic meta-programming approach is used to represent a model-based diagnosis process, expressing preferences and strategies, based on extended logic programs.

Libraries of problem-solving methods and ontologies have been discussed by several authors, for example (Benjamins, 1993). The object- and metatheory in our diagnosis approach could be reused by generalizing the components of the theories to a schematic representation, in the form of a library. A library could be designed as a meta-metatheory of schemata, or templates, for the formulas of the

object- and metatheory. The process of instantiating these schemata would be controlled by the meta-metatheory, formalizing the process.

Simulation is normally seen as a numerical technique, without reasoning and symbolic processing. In (Haddock, 1987) it is pointed out that a relatively high level of training is necessary to perform useful simulation studies. In (Lee et al., 1996) a Hybrid Expert and Simulation System, HESS, is described where expert system techniques are used both for input of simulation parameters and for interpreting and analyzing the results. The idea to combine simulation tools with expert systems was proposed by O'Keefe (1986). We have not really investigated the suitability of the HESS approach for our tunnelling domain. So far, the integration of simulation directly in our Prolog scheduling program has worked very well and we see no need for a special simulation system. The Prolog simulation approach has the great advantage of being transparent to the end user—we can explain how it works without an expert system working as an intermediate interface.

In a special issue of the Constraints journal, on strategic directions in constraint programming, several researchers point to the need for constraint modeling languages. Pascal van Hentenryck (1997) claims that "Constraint programming is still far from the description of the problem and the gap between the description and the program is non-trivial." He puts forward the concept of global modeling as a particularly challenging problem, of which the traditional combinatorial problem, as stated by a constraint program, is only a small part. Eugene C. Freuder (1997) emphasizes the problem of knowledge acquisition: "We want to extract the relevant constraints, and automate the translation of constraints expressed in the user's language into a form appropriate for problem solving". Pascal van Hentenryck and Vijay Saraswat (1997) discuss the problem of efficient modeling: "We need to automate the process of moving from problem descriptions natural to the problem domain to problem descriptions designed for efficient solutions." Our modelling language CML is not a language for modeling mathematical problems, cf. the Numerica modeling language (Hentenryck, Michel, and DeVille, 1997). Instead, it attempts to model practical real world problems as naturally as possible, by preserving the problem structure of the domain and allowing constraint formulations that reflect natural language expressions. For instance, when modeling a staff planning problem, the CML structure reflects a working schema and its assignments and the finite domain variables are extracted from the structure when transforming the CML problem statement to a constraint program.

In (Paltrinieri, 1995) a visual constraint programming environment *Gianna*, based on an object-oriented design methodology, is discussed. A graphical model editor is used to introduce the classes of e.g. tasks and their constraints as nodes and edges. The constraints are then specified in text form and the resulting program is executable by traditional constraint programming language. In contrast to DT-CML, Gianna does not provide any decision support mechanism for the user in setting up the constraint problem. The factors underlying the specification of the constraints are not made visible and sharable by people in the organization. It is stated that appropriate heuristics for constraint propagation are specified, but this is

not illustrated. Compiling into efficient constructs, such as global constraints, is not discussed.

Vanthienen and Dries (1994) present a decision table tool for graphically constructing decision tables and generating, for example, Pascal code implementing the tables. The intended application area is knowledge based systems. The tool supports checking the tables for completeness, correctness, contradictions and simplifying the tables. The system does not address generation of constraints–it is an example of a graphical decision tool for generating knowledge rules specified in decision tables.

Farfarakis and Hope (1997) introduces a software engineering method, $TOCCA_n$, for analysis and design of constraint problems, based on object-orientation. A problem is analyzed in a class diagram and a state diagram, both based on the object-oriented theory. A constraint diagram is added that classifies each constraint and specifies it in a first-order logic format. The resulting model is more accessible for a user than a text-only specification. It also holds a potential for automatic translation into an executable form. The $TOCCA_n$ approach does not provide decision support for the user when consulting a finished system.

Renschler (1998) presents a 'Configuration Spreadsheet' where a user can interactively set mutually dependent parameters of a configuration problem. The system modifies the available parameters depending on the choices of the user, to obtain a consistent solution. An idea is to let the user choose parameters in steps (in a 'wizard' style of interaction) so that groups of parameters can be addressed by the user in the order of importance. The way that alternatives for the user reduce as the user chooses alternatives in the cells is similar to the way the alternatives reduce in the decision tables of our approach. The hierarchical structure of the tables, where the user can seek decision support for a particular parameter, does not seem to be part of Renschler's spreadsheet. Moreover, the structure of the decision tables facilitate an overview by the designer of the tables, where all the alternatives are visible at the same time in a table.

Milton and colleagues (1999) discuss the use of knowledge engineering methods for knowledge management. The current techniques in knowledge management are classified as information technologies, which primarily deal with storing and communicating information. They develop a framework for identifying opportunities to use knowledge technology methods in knowledge management, and report lessons and insights from the use of these methods for a large intranet.

2.6 CONCLUSIONS

We have investigated knowledge technology applications for knowledge management. In particular, we have studied knowledge representations for the application areas diagnosis, simulation and scheduling, with the aim to let the user directly manage the knowledge content of the applications. To achieve this we connect knowledge in two directions. First, to the user by illustrating the knowledge content in a form that makes the knowledge modifiable by the user; second, back to the computer by transforming it into a form suitable for inference and problem

solving. In each of the application areas we have used a representation form designed specifically for the domain and type of application, because each puts different requirements on the representation. O'Leary (1998) asks for research on what representation forms work best for different types of knowledge. It still remains to develop a guide to this, but the domain and type of application are certainly major factors.

It seems possible to let the user modify central parts of the knowledge content in the applications, by using special illustration techniques. The techniques used are object-based graphics for manipulating device components and connections in diagnosis, simplified rules for simulation of tunnelling activities, text-based query language specification of scheduling problems and finally, decision tables for constraint problems and decision support. Some informal user evaluation of the illustration techniques has been conducted with positive indications. Rigorous and formal user evaluation of the illustration forms remains to be done.

Davenport and Prusak argue that the challenge of knowledge management lies in codifying knowledge in structures that allow rapid and flexible change of the knowledge, because knowledge itself changes rapidly and flexibly. To this can be added that it is important to provide user interfaces that allow the user to understand the knowledge and manipulate it.

To speculate, the new e-economy will benefit from knowledge intensive applications that make it possible to design intelligent products and services. The pace of change and the range of products will make knowledge management of the knowledge applications a very interesting prospect. Visual modelling languages and illustration techniques seem to be important tools for effective implementation of these applications.

ACKNOWLEDGEMENTS

Many thanks to all of you who have helped and supported me during my work on the thesis. I am greatly indebted and most grateful to the following people.

Professor Åke Hansson for useful advice, many stimulating discussions, never failing interest and constant encouragement. He also designed and implemented a very nice graphical user interface to the simulation system presented in paper II.

Dr. Torkel Hjerpe, my co-author on papers II–IV, for most stimulating and creative collaborations. We have worked closely on these projects and credit for them goes to him in equal part. Furthermore, his Ph.D. thesis is the foundation for CML and DT-CML, presented in papers III and IV.

Professor Keith Clark for valuable comments and suggestions on paper I.

The people at the department over the years for making it a pleasant, stimulating and interesting place to work.

The people at Intologic for making it possible for me to divert my attention from business while finishing this work.

John Roche for kind help and advice on printing the dissertation and for designing a beautiful cover.

My family and parents for constant support and encouragement. Malin, my wife and number one supporter, for always being there. My parents Kerstin and Dan for teaching me important things about life. My children Agnes and Henrik for inspiration. Kristina and Bo for always helping out when needed.

Uppsala University provided generous funding that allowed me to complete this research.

Volvo Research Foundation, Volvo Educational Foundation and Dr. Pehr G. Gyllenhammars Research Foundation supported the CML project financially with grant 97:20.

Uppsala, October 2000
Kent Andersson

REFERENCES

Benjamins, R. (1993). Problem Solving Methods for Diagnosis, Ph.D. Thesis, University of Amsterdam.

Brooking, A. (1999). *Corporate Memory: Strategies for Knowledge Management.* International Thomson Business Press, London.

Concise Oxford Dictionary (1982). University Press, Oxford.

Console, L., Dupré, D. and Torasso, P. (1989). A Theory of Diagnosis for Incomplete Causal Models, *Proc. Eleventh International Joint Conf. on Artificial Intelligence*, Detroit pp. 1311–1317.

Console, L., Portinale, L., Dupré, D.T. and Torasso, P. (1992). Combining Heuristic Reasoning with Causal Reasoning in Diagnostic Problem Solving. *Second Generation Expert Systems*, Springer-Verlag, Berlin, pp. 46-68.

Cosytec (1996). *CHIP User's Guide*, COSY/USER/001 Ver. 5.0, Cosytec S.A., France.

Davenport, T.H. and Prusak, L. (1998). *Working knowledge: how organizations manage what they know.* Harvard Business School Press, Boston, Mass.

Davis, R., Buchanan, B. and Shortliffe, E. (1977). Production Rules as a Representation for a Knowledge-Based Consultation Program, *Artificial Intelligence* 8, pp. 15–45.

de Kleer, J. and Williams, B.C. (1987). Diagnosing Multiple Faults, *Artificial Intelligence* 32, pp. 97–130.

Dincbas, M., Simonis, H. and Hentenryck, P. van (1990). Solving Large Combinatorial Problems in Logic Programming. *Journal of Logic Programming*, 1990:8, pp. 75-93.

Dieng, R., Corby, O., Giboin, A. and Ribière, M. (1999). Methods and tools for corporate knowledge management. *Int. J. Human-Computer Studies*, **51**, pp. 567–598.

Farfakis, R. and Hope, S. (1997). The TOOCA$_n$ Approach, *The Second International Conference on the Practical Application of Constraint Technology (PACT'97)*, London.

Freuder, E.C. (1997). In Pursuit of the Holy Grail, *Constraints: An International Journal*, 2(1), pp. 57–61.

Haddock, J. (1987). An expert system framework based on a simulation generator. *Simulation*, 48, 46, 1987.

Harris, K. (1998). *The Value Proposition of KM*, Gartner Group Symposium ITExpo98 presentation, 2-5 November, Cannes.

Hentenryck, P. van (1989). Constraint Satisfaction in Logic Programming, MIT Press, 1989.

Hentenryck, P. van (1997). Constraint Programming for Combinatorial Search Problems, *Constraints: An International Journal*, 2(1), pp. 99–101.

Hentenryck, P. van and Saraswat V. (1997). Constraint Programming: Strategic Directions, *Constraints: An International Journal*, 2(1), pp. 7–33.

Hentenryck, P. van, Michel, L. and DeVille, Y. (1997). *Numerica: A Modeling Language for Global Optimization*, The MIT Press.

Hjerpe, T. (1995). High-level Specification and Efficient Solving of Constraint Satisfaction Problems, PhD Thesis, *Uppsala Theses in Computing Science* 22, Uppsala University.

ILOG (1996). *Ilog Solver Reference Manual*, Version 3.1, ILOG S.A., France.

ISO (1999). *Database Language SQL*. International Organization for Standardization, Document ISO/IEC 9075:1999.

O'Keefe, R. (1986). Simulation and expert systems—A taxonomy and some examples. *Simulation*, 47, 10, 1986.

Koton, P. (1993). Combining Causal Models and Case-Based Reasoning. S*econd Generation Expert Systems*, Springer-Verlag, Berlin, pp. 69-78.

Kühn, O. and Abecker, A. (1997). Corporate Memories for Knowledge Management in Industrial Practice: Prospects and Challenges, *J. Universal Computer Science*, **3**, pp. 929-954.

O'Leary, D.E. (1998). Enterprise Knowledge Management. *Computer*, **31**, pp. 54–61.

Lee, H.F., Chou, H.J. and Klepper, R.W. (1996). A HESS for Resource Planning in Service and Manufacturing Industries. *Expert Systems with Applications*, vol. 10, No 1, pp. 147-156.

Macintosh, K. (1999). Knowledge management techniques: teaching and dissemination concepts, *Int. J. Human-Computer Studies*, **51**, pp. 549-566.

Milton, N., Shadbolt, N., Cottam, H. and Hammersley, M. (1999). Toward a knowledge technology for knowledge management. *Int. J. Human-Computer Studies*, **51**, pp. 615–641.

Nejdl, W., Frölich, P. and Schroeder, M. (1995). A Formal Framework for Representing Diagnosis Strategies in Model-Based Diagnosis Systems. *Proc. 14th Int. Joint Conference on Artificial Intelligence*, Montréal, pp. 1721–1727.

Nonaka, I. (1991). The Knowledge-Creating Company, *Harvard Business Review*, Nov-Dec, Harvard Business School Publishing, Boston, Mass.

Paltrinieri, M. (1995). A Visual Constraint-Programming Environment, *Proceedings of CP'95—First International Conference on Principles and Practices of Constraint Programming, LNCS 976*, pp. 499–514.

Polanyi, M. (1957). *The Tacit Dimension*. Doubleday, New York.

Polanyi, M. (1958). *Personal Knowledge—Towards a Post Critical Philosophy*. University of Chicago Press, Chicago, Routledge & Kegan Paul, London.

Renschler, M. (1998). Configuration Spreadsheet for Interactive Constraint Problem Solving. (patent pending). Ericsson Germany. *The Third International Conference on the Practical Application of Constraint Technology (PACT'98)*, London, pp. 427–440.

Simmons, R. and Davis, R. (1987). Generate, Test, and Debug: Combining Associational Rules and Causal Models. *Proc. Tenth Int. Joint Conference on Artificial Intelligence*, pp. 1071-1078.

Steels, L. (1990). Components of Expertise, *AI Magazine*, Summer 1990, pp. 29–49.

ten Teije, A. (1997). Automated Configuration of Problem Solving Methods in Diagnosis. Ph.D. Thesis, University of Amsterdam.

ten Teije, A., van Harmelen, F., Schreiber, G. and Wielinga, B. (1996). Construction of Problem-Solving Methods as Parametric Design. In *Proceedings of the Tenth Workshop on Knowledge Acquisition for Knowledge-Based Systems (KAW'96)*, Banff, Alberta.

Vanthienen J. and Dries E. (1994). Illustration of a Decision Table Tool for specifying and implementing Knowledge Based Systems, *Int. J. Artificial Intelligence Tools*, (3)2, pp. 267–288.