Preface

This master thesis is written with guidance from the research group for Artificial Intelligence and Learning (AIL) at the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU). This master thesis is the completion of the master of technology degree in computer science.

The goal of this thesis is to develop a knowledge acquisition method for the knowledge-intensive case-based reasoning system TrollCreek. This includes studying the state-of-the-art knowledge acquisition modelling methods, analysing them and suggesting a method for TrollCreek.

I would like to thank my supervisor Agnar Aamodt for giving me a chance to learn a something completely new and for insisting on that there is no such thing as stupid questions. Without his guidance this thesis would not have been possible.

I would also like to thank my family and friends for supporting me.

Trondheim 14. June 2005

Elise Bakke

Contents

Chapter 1

Introduction

> "Knowledge-acquisition builds a consolidated model of an expert's problem solving behaviour in the terms of knowledge".
>
> — *Walter Van de Velde [dV93]*

This thesis proposes a knowledge-acquisition method for the knowledge-intensive case-based reasoning system TrollCreek [Aam04]. The method exploits the possibilities in the TrollCreek knowledge editor (i.e. the GUI, the top ontology). To get the desired results an extensive study of state of the art knowledge-acquisition methods is needed. The knowledge-acquisition method is based on the knowledge from studying the existing methods and systems. There exist several methods and modelling principles for handling general domain knowledge. However, there are not many methods for handling situation specific knowledge (i.e. cases).

1.1 Background and motivation

The background for this thesis is the focus in the AIL group at IDI NTNU on building models of the real world in order for expert systems to reason (both using model based and case based reasoning). The AIL group has participated in the development of TrollCreek knowledge editor. TrollCreek is a knowledge-intensive case-based reasoning tool. TrollCreek utilizes both a case-base of previous experiences and general domain knowledge. This makes it a challenge to develop a knowledge acquisition-method for TrollCreek (as the existing approaches focuses on general domain knowledge).

In general this field was started in the early 1980s and had a high level of activity in the early 1990s. However, several of the most prominent

researchers in the field have moved their primary research activity onto other fields of study. The field of knowledge acquisition and modelling is also not currently "active" at NTNU. There are no classes taught on the subject. This represents quite a challenge. In order to even start working with the knowledge-acquisition, a great deal of literature studies had to be done. It is a broad, abstract field that one had to get the grasp of before starting to write the thesis. Some of the results are presented in chapter 3.

Ole Martin Winnem wrote a master thesis in 1996 [Win96]. He made the CREEST workbench, based on the KREST [Ste93] a workbench for the components of expertise framework. However, TrollCreek has come a long way since 1996. TrollCreek has been implemented in Java and new features have been added. The graphical user interface is radically different and the top ontology constantly evolving. His work has been an inspiration to this thesis, but very little of the work could actually be used in the thesis.

A larger inspiration was an article by Aamodt [Aam01]. This article started the process of modelling the contents of case based reasoning systems (CBR). However, this was a start and needed to be elaborated upon in order to start working with the knowledge acquisition and modelling for knowledge intensive CBR.

There exists two documents describing modelling in TrollCreek (discussed later in section 4.2.5). However, these are ad hoc solutions with no real focus on the knowledge acquisition perspective.

The development of the Java version of TrollCreek has been done based on previous versions. However, there has been no specific focus on knowledge level(KL)-modelling and the higher level process of knowledge-acquisition. It is a great and inspiring challenge to bring the knowledge-acquisition / knowledge level modelling perspective into TrollCreek again. This became a major motivational factor. It is also great to be able to learn a new side of artificial intelligence and learning.

1.2 Goal

The main goal of this thesis is to suggest a knowledge acquisition modelling method for TrollCreek. This is to be done by looking at already existing frameworks in the knowledge acquisition and modelling research based on a study of state of the art international knowledge acquisition and modelling methods. In the original problem description for this thesis knowledge level-modelling, "Components of Expertise" and Protégé

are mentioned. During the study of the related research I also found it necessary to include the "CommonKADS" framework.

Chapter 2

Research Approach

In this thesis the result will be a knowledge acquisition method and an example of the method used. In addition a large amount of work will go into literature studies on the matter of knowledge acquisition. There will be no implementation, as this thesis is concerned with knowledge level modelling [Ste90].

This thesis uses the proof of technology concept which is a design oriented research.

2.0.1 Thesis Approach

To realise the higher level goal of proposing a knowledge acquisition method, our research approach is to be used:

1. Describe the international related research
   (a) The knowledge level
   (b) KL modelling
   (c) Components of Expertise
   (d) CommonKADS
   (e) Protégé
2. Introduce TrollCreek
3. Compare and analyse methods encountered
4. Define the choices made in this thesis

5. Describe the KA method for TrollCreek

    (a) Utilizing the lessons from related research

    (b) According to the choices made

    (c) Utilizing the features in TrollCreek (GUI, top-ontology++)

6. Give an example of the method in use

    (a) Define an example domain

    (b) Make an actual model in TrollCreek

7. Discussion, evaluation, further work and conclusion

These concrete sub-goals will be addressed in the following chapters.

2.1 Cohen's Framework for Evaluation in AI Research

Paul Cohen [CH88] introduced a framework for evaluating artificial intelligence (AI) research. To some extent it is possible to use some of the principles already when planning the research approach. Cohen emphasises the importance of describing why one has made the choices made in the research. These choices being why we do the research, why our tasks are illustrative, why our methods and views are an improvement and so on [CH88]. In artificial intelligence it is not only the performance of the programme.

Cohen defined a cycle with five different stages in AI research. Cohen also introduced sets of evaluative questions and experiment schemas corresponding to the five stages. The five different stages are [CH88]:

1. Refine the research topic to a "task" and a view of how to accomplish the task
2. Refine the view into a specific method
3. Develop a program that implements the method
4. Design experiments to test the program
5. Run the experiments

In this thesis there will not be an implementation of a whole system. The methodology for knowledge acquisition will be developed for an already existing system TrollCreek. The underlying CREEK framework has already been evaluated using Cohen's framework in Aamodt's doctoral dissertation [Aam91].

Strong guidelines what "task" the research shall attack has already been given through the description of my master thesis ("knowledge-acquisition and modelling for knowledge-intensive CBR"). Within the same description lies also the guidelines to what "view" to take to the task (when specifying which methodologies/existing frameworks (like i.e. knowledge level modelling and components of expertise) to take a look at). A refinement of the general view guideline happens in the choices made in section 6.1.

However, the second stage defined by Cohen is definitely the most relevant for this thesis, thus it is not a perfect match. It is still necessary to use common sense when evaluating. It seems like Cohen has been thinking of a machine internal algorithm when defining the term method. The method developed through this thesis has a human as the user. However, the criteria presented by Cohen in [CH88] for evaluating methods are:

1. How is the method an improvement over existing technologies?

   (a) Does it account for more situations (input)?

   (b) Does it produce a wider variety of desired behaviours (output)?

   (c) Is the method expected to be more efficient (space, solution time, development time, and so on)?

   (d) Does it hold more promise for further development (for example, because of the introduction of a new paradigm)?

2. Does a recognized metric exist for evaluating the performance of your method (for example, is it normative, cognitively valid)?

3. Does it rely on other methods? (Does it require input in a particular form or pre-processed input? Does it require access to a certain type of knowledge base or routines?)

4. What are the underlying assumptions?

5. What is the scope of the method?

   (a) How extendible is it? Will it easily scale up to a larger knowledge base?

   (b) Does it exactly address the task? Portions of the task? A class of tasks?

   (c) Could it or parts of it be applied to other problems?

   (d) Does it transfer to complicated problems (perhaps knowledge-intensive or more or less constrained or with complex interactions)?

6. When it cannot provide a good solution, does it do nothing or does it provide bad solutions or does it provide the best solution given the available resources?

7. How well is the method understood?

   (a) Why does it work?

   (b) Under what circumstances, won't it work?

   (c) Are the limitations of the method inherent or simply not yet addressed?

   (d) Have the design decisions been justified?

8. What is the relationship between the problem and the method? Why does it work for this task?

It is important to have these criteria in the back of the head when proceeding with this thesis. Instead of implementing an own system for the knowledge acquisition method, it is a method made for TrollCreek system. However, several aspects of the evaluation are very relevant for this thesis.

Chapter 3

Related Research

There are several important systems and methodologies in the field of knowledge acquisition and knowledge modelling. Newell introduced the knowledge level [New82] and set the standard for a new way of thinking within knowledge acquisition. This chapter will first introduce Newell's knowledge level (in section 3.1), before discussing a more practical application of the knowledge level in Knowledge Level Modelling (in section 3.2). Several frameworks and systems already use KL modelling. We introduce some of them by discussing the Components of Expertise (section 3.3) and the CommonKADS (section 3.4) frameworks, before introducing the Protégé knowledge acquisition programme (in section 3.5).

The related research described here makes up the state-of-the-art within international knowledge acquisition and modelling.

3.1 Newell's knowledge level

Allen Newell introduced the term "knowledge level" already in 1982 [New82]. Based on the standard practice in the artificial intelligence community, Newell proposed a new level in addition to the standard computer system levels. Traditionally a computer system is described as having different levels, ranging from bottom and up as shown within the bottom five boxes in 3.1. The different levels have different mediums and behaviour laws. The behaviour laws that govern how the level uses the medium. As an example, while the device level consists of transistors and resistors, the circuit level looks at the devices as circuits (the medium) and i.e. Krichoff's laws govern the behaviour of the circuits. In comparison the symbolic level looks at computers as the sys-

tem, symbols and expression as the medium, memories and operations as the components with sequential interpretation as the behaviour laws [New82].

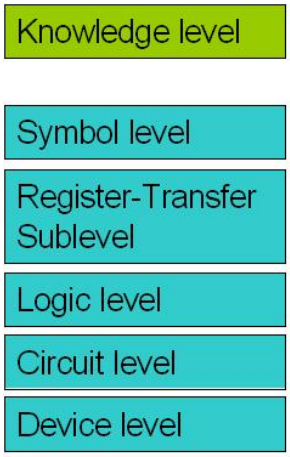

Figure 3.1: The computer's system levels (adapted from [New82]

Newell was the first to introduce the notion of the knowledge level [New82]. The knowledge level enables us to observe the computer system from the outside. An external observer gains an external view on the systems behaviour. The knowledge level is shown in 3.1. At the knowledge level, the system is an agent. The agent consists of both a physical body and a body of knowledge. How knowledge is encoded is a concern for the symbol level. The knowledge level does not need to know this. The agent also has a set of goals and a set of actions. A goal is a certain a state of the world that the agent strives to achieve. The actions are actions the agent can perform to reach a goal. An extremely important feature with Newell's knowledge level is the complete lack of structure in knowledge.

3.1.1 The Principle of Rationality and Knowledge

According to Newell [New82] the agent behaves according to the principle of rationality:

> **Principle of rationality**
> *If an agent has knowledge that one of its actions will lead to one of its goals, and then the agent will select that action.*

Newell defines the term knowledge in close connection with the principle of rationality. The agent processes knowledge in order to decide on what action to take. According to Newell, knowledge is what is ascribed

to an agent so that it acts according to the principle of rationality. This illustrate how the knowledge level distances it self from the symbolic level and views the system subjective from the outside. An important feature with the knowledge level is that the amount of knowledge is not important to determine if a system might be at the knowledge level or not. The important feature is how the system utilizes its knowledge. In many ways, knowledge is the specification of what the symbolic level should be able to do. The symbolic level holds the representation of knowledge (the system at the symbolic level being the computer).

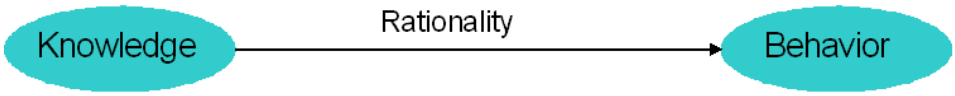

Figure 3.2: Core terms from the principle of rationality (adapted from [dV93])

Newell also introduces what he calls the extended principle of rationality [New82], by adding auxiliary principles. However, even Newell recognizes the "*radical incompleteness*" that characterizes the knowledge level. In many situations, the pure form of the principle of rationality fails to determine behaviour uniquely.

By taking a knowledge level approach to a system one might focus on knowledge contents instead of worrying about the underlying architecture, implementation or organization [Aam01].One will able to focus on what a system does and why it does what it does. One might be able to predict and understand system behaviour without knowing the implementation of knowledge at the symbolic level.

3.2 Knowledge Level Modelling

However, when dealing with real world applications, the pure form of knowledge level has several weaknesses. The knowledge level uses knowledge only as a resource for accomplishing a goal. Knowledge has no structure at the knowledge level, which makes it hard to analyse knowledge at the knowledge level. The knowledge level also uses an ideal form of rationality. It assumes an ideal world and an agent with no physical or worldly constraints. This is a coarse simplification of the real world, and not applicable in real world scenarios. Even if Newell uses the extended principle of rationality and recognizes the fact that knowledge never is finite, it is impossible to use it efficiently in the real world.

Knowledge Level (KL) modelling uses a modified version of Newell's knowledge level. As shown in figure 3.3 the KL modelling is an activity and a fictious level between the pure knowledge level and the symbolic level.

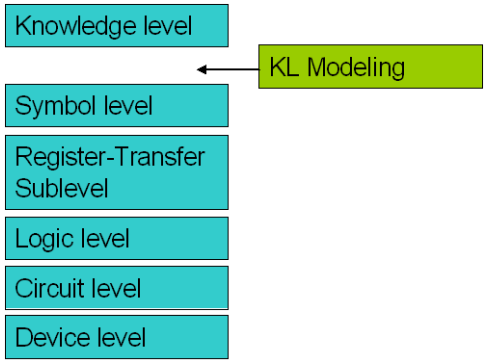

Figure 3.3: KL modelling

In KL modelling there is a minimal structure to knowledge. Because of the structure the KL models can also say something about the "what" and "how" models of behaviour. KL modelling says something about the structure used within knowledge in order to achieve a goal. It is not just the "why" aspect of the systems behaviour [dV93]. This structure is the reason for moving the KL modelling toward the symbolic level in 3.3. To describe the new level there are several possible terms (i.e. "knowledge use level" [Ste90]). The overall knowledge acquisition and modelling community uses this modified version of Newell's knowledge level.

3.2.1 Two-Step Rationality

In a real world setting, one has to make assumptions about the world. Actions are the means the agent has to interact with the world. Therefore, it is through actions the agent gains knowledge, and knowledge gained will not be perfect. The new, modified organization of the knowledge level, uses a two-step (tractable) rationality [dV93].

Newell (as mentioned in 3.1 and shown in figure 3.2) uses the pure form of the principle of rationality. He uses the rationality to get from knowledge to reason about the behaviour. In contrast the two step rationality, as shown in figure 3.4. It reflects that a systems adequate behaviour is both practical and rational [dV93].

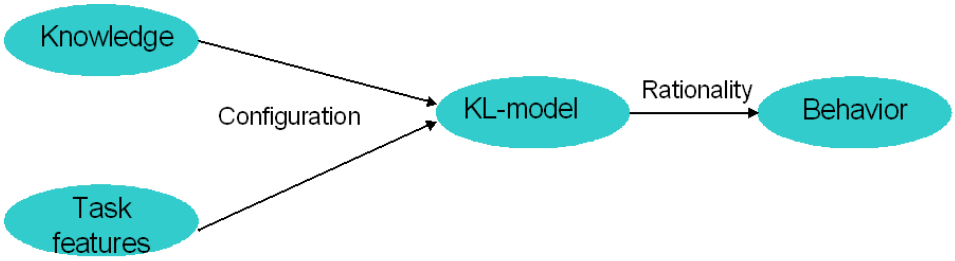

Figure 3.4: The two-step rationality (adapted from [dV93])

The "practical" aspect of the rationality it is the first step. This is where the configuration takes place. This is when the agent imposes structure on knowledge and assigning roles for it to use in the reasoning. As illustrated by fig 3.4, the configuration gives us the KL-model. Therefore, the KL-model is a structured part of knowledge and it contains the way an agent frames a task instance [dV93]. The configuration enables the practical application of the principle of rationality. The "task features" include the agent's environment, how the agent interacts with it and the additional restrictions. These task features is what separates one task from the other tasks.

3.2.2 Structuring Knowledge

As mentioned earlier KL-modelling imposes a minimal form of structure to knowledge (during the configuration). There are three main types of knowledge [Aam91]. Figure 3.5 shows these.

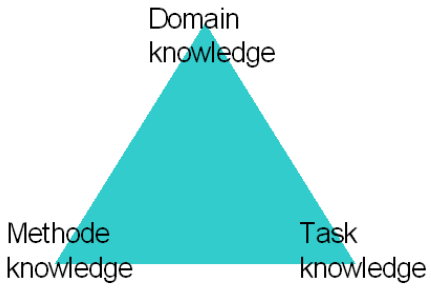

Figure 3.5: The three knowledge types

These three main types represent different perspectives on knowledge. By classifying knowledge according to these perspectives, knowledge attains minimal form of structure. The different perspectives models different aspects of knowledge and results in different forms of KL-models.

Domain Knowledge

Domain knowledge looks upon knowledge as the facts about the world that an agent needs to perform a task.

A domain model is a precise way to express what one assumes the collection of statements about the world to mean [dV93]. The domain model uses "coherence" (i.e. a causal relation between cause and effect). This makes up the model ontology. The ontology gives the model predictive power.

Task Knowledge

The task perspective concentrates on the "what" aspect of the agent's behaviour, based on observations. It defines tasks by the goals the system wants to fulfil and the tasks are decomposed into subtasks.

The task model gives us a way of dealing with the goals in a more precise way. The task model deals with what it means to achieve a goal and how the goals are interrelated. It captures how the goals are interrelated to other goals [dV93].

Method knowledge

As the name suggests, the method knowledge is "how" to accomplish a task. Problem-solving methods are the models using method knowledge [dV93]. In many ways, the problem solving methods relates task and domain models to achieve goals. It describes the roles the models have to play in order to achieve goals. A problem solving method differentiates it self from task models because it can result in several task models and be unspecified for several tasks.

Examples of use can be the instantiation and configuration that are an important part of ProtégéI and ProtégéII (described in more detail in section 3.5). Method knowledge is also used to some extent in the CommonKADS framework.

Examples of use

Several approaches use KL modelling. One example is McDermott (in [McD88]) who introduced the role-limiting problem solving methods as a reasoning strategy. They introduced both problem solvers and knowledge-acquisition (KA) tools. An example of KA tools introduced by them

(all the tools are described in more detail in section 5.1.1) is MOLE [EEMT87] which uses the cover-and-differentiate method for classification. The initial state is a set of one and more symptoms. The first step is to determine the events that possible explains the symptoms (cover) and identify information that will differentiate the candidates, before applying it. They identify explicitly the different ways in which the role-limiting problem solver uses inference from the knowledge base [EST+95]. The role limiting method reduces the development task to identifying what domain knowledge is required in order to fulfil each role. However, the role-limiting method assumes that the behaviour of the knowledge-based system can be defined in domain-independent terms. In order for the role-limiting methods to be reusable, they have to be general. This races a challenge when applying the method to a specific application task. There will then be a gap between the general method and the application task [EST+95]. Section 5.1.1 briefly discusses possible problem solving methods.

Chandrasekaran introduced task-specific architectures [Cha87], [Cha86], [Cha90]. He introduced the notion of generic tasks (GT). A generic task also includes information about the domain structures needed to execute the task. Chandrasekaran felt that classification, data retrieval, plan selection and refinement, state abstraction and adductive assembly were in some sense re-usable subtasks [CJ93]. The GTs are components in other, more complex problem-solving tasks.

This thesis looks at the KL-modelling frameworks Components of Expertise and CommonKADS. It has already been done initial research on adapting Components of Expertise to knowledge-intensive case-based reasoning [Aam01]. CommonKADS is the dominating KA methodology in Europe. It is natural to have a look at both these. The thesis also looks at Protégé the meta level KA program. Protégé is a very much used system and has almost become the standard editor for ontology modelling.

3.3 Components of Expertise

The CoE framework [Ste90] gives us a way to select expert system solution based on the task characteristics at hand. CoE enables us to make a mapping between the conceptual feature, pragmatic constraints and available knowledge to the components (i.e. problem solving methods, domain models and task structures). A componential framework decomposes the knowledge level description into many different components. Compared to CommonKADS, the Components of Expertise (CoE) frame-

work is more informal. The CoE framework is used for example in the KREST workbench (another name for the same workbench is COMMET [Ste93]).

3.3.1 Conceptual vs. pragmatic viewpoint

Each task and subtask can be viewed from both a conceptual and a pragmatic viewpoint. From the conceptual viewpoint, task is characterized as a problem that needs to be solved (i.e. diagnosis, interpretation, planning, and design). In contrast, the pragmatic viewpoint concentrates on the constraints in the tasks. These constraints are a product of the environment where the system operates or from the epistemological limitations. Example of such limitations can be time and space, observation and formation. From these epistemological limitations follows pragmatic constraints in the system.

3.3.2 Perspectives On Knowledge

CoE expertise also uses the three different perspectives on knowledge (as seen in figure 3.5). Usually the three perspectives are used in a spiral movement (as shown in figure 3.6). By doing it this way the descriptions of each perspective will be progressively refined [Ste93].

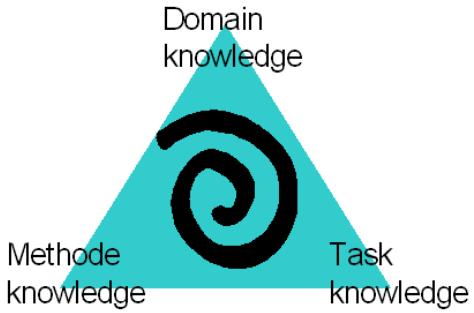

Figure 3.6: The progressively refining description of each perspective

In short, one might say that the task perspective specifies WHAT needs to be done by the problem solver (the major tasks and subtasks in the application), the model perspective specifies WHAT knowledge is available and the methods describe HOW and WHEN knowledge is going to be applied.

A more detailed description of the three perspectives is given in the next sections.

3.3.3 The Tasks Perspective

A task still is something that needs to be accomplished. The CoE framework assumes that there is a detailed analysis of the task [Ste90]. In a real world situation there is most likely a conglomerate of mutual dependent tasks to e dealt with.

There is a distinction between the domain acquisition tasks (acquiring domain models) and the application tasks (developing case models). There is also a clear distinction between the solution tasks (have no further subtasks) and the decomposition tasks (possible to split/decompose into further sub-tasks).

Task Structure

All the tasks can be decomposed into subtasks with input-output relations between them (task decomposition). This makes up a treelike structure (as shown in figure 3.7), a part-subpart hierarchy showing how tasks relates to subtasks. However, the task decomposition does not say anything about the interdependencies between tasks (i.e. horizontally in the hierarchy). This aspect is handled by the control methods.

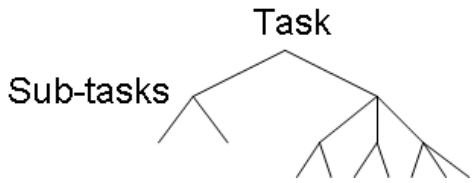

Figure 3.7: CoE task decomposition

The task decomposition approaches the decomposition in a top-down to achieve the first subtask. Sometimes not all tasks are executed for each situation (case). A task structure might be an AND/OR tree. Human users (as opposed to the system executing every task) might even execute some of the tasks in the task structure [Ste91]. The control flow diagram (will be introduced in section 3.3.5) governs the control structure and decides what tasks to be executed in what sequence.

When looking at i.e. medical diagnosis, car problem diagnosis or computer diagnosis, there are several similar features. It is then possible to construct a generic diagnosis task based on the observations [Ste91].

Model Dependency Diagram

In the task perspective the model dependency diagram uses both target models (models that tasks have an impact on), source models (models that a task consults) and the interface between them. The model dependency diagram illustrates the data flow relationship between the models and the tasks. Input interfaces and source models have pointers going to a task. In contrast, the target models and output interfaces have pointers coming from a task.

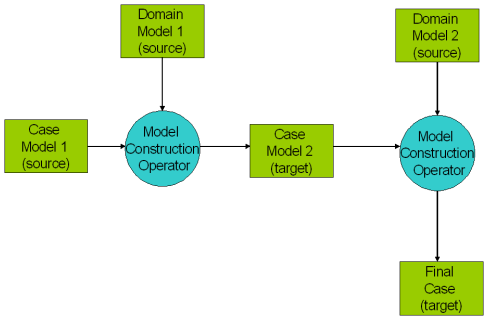

Figure 3.8: A model dependency diagram (adapted from [Ste91])

The model construction operators (shown in figure 3.8) have a one-to-one connection to tasks. Moreover, in the implemented system, when clicking on the model construction operator, it shows the problem solving method. The connection is also a one-to-one relation between tasks and methods.

3.3.4 The Model Perspective

Domain knowledge in CoE is referred to as models. A model makes an abstraction from certain aspects of the reality [Ste93]. In a classificatory model, a model relates the features to the classes and enables the classification. However, there are several ways of making general distinctions between models:

Domain vs. Case Models

It is possible to make a distinction between two main types of models [Aam01][Ste90]. These are the case and domain models:

- **Case models**

  - models of the CURRENT problem-solving situation
  - problem solving state descriptions
  - part of the working memory

  Please note that the case model in CoE is different from the case in case-based reasoning (as discussed in section 5.2.1).

- **Domain models**

  - makes up the contents of knowledge base
  - consists of all knowledge that is part of the long-term memory
  - more abstract than the case models
  - expands the case model by interference or data gathering [Ste90]
  - fixed on a particular application
  - valid for several case models

The ontology models are a special class of domain models. The ontology constraints the vocabulary used in other models.

Type of Case Models

The case models are a set of facts about the situation. The different models can be discriminated between based on what kind of facts they are modelling [Ste91]. For instance, facts about temporal properties are grouped into temporal models and so on. The distinction between the different case models will not be visible in the symbolic level, according to Steel. Steels also mention a set of different models:

- Component model (describe component and its subcomponents)
- Descriptive model (models the system in terms of its features)
- Classificatory model (specifies what class a system belongs to)
- Connection model (connections between the parts of the system)
- Temporal model (identifies ordering of action in the system)
- Spatial model (physical location of object in space)
- State model (parts of the system being in a certain state)
- Causal model (identifies causal relationships between states)
- Behavioural model (describes process)

- Functional model (functionality of system)

In addition to these it is also possible to make a deviation model (describes how the system derives from what is expected). The elements of a model might be shown in a hierarchical structure (for instance a class hierarchy, a components organized in modules and so on).

The type of model built can be decided by the types of questions asked to build the model and the type of task used (classification, description, selection, configuration and so forth).

Types of Domain Models

Domain models contain domain-specific knowledge. The problem solving method uses domain specific knowledge to construct the case models (at the knowledge level the PSM are regarded as domain independent) [Ste91].

In the KREST workbench there are not clearly expressed a predefined set of models. However, there are two main categories (closely connected to the methods discussed later 5.1.3).

Expansion Model The expansion model contains knowledge that enables an expansion of the model [Ste91]. The resulting model has at least the same number of elements as the one before [Win96].

Examples of expansion domain models are [Win96]:

- Descriptive domain model
- Functional domain model (contains functional hierarchy as well as functions)
- Component domain model (constraints on the components)
- Connectivity domain model (constraints on the connections between the components)
- Descriptive domain model (constraints on the features the client may have)

Mapping Model As the name suggest, the mapping model maps from one model to another. They are used to construct or modify a set of target models (case models), based on mapping of elements from other source models.

Steels identify three different models:

- Description-to-class model (maps a set of description into a class)
- Function-to-component model (maps a functional model to a componential model)
- Symptom-to-malfunction model (maps symptoms to malfunctions)

Form vs. Contents

It is also possible to make a distinction between the form and the content of a model [Ste93].

- The form handles the structure of the model.
- The content refers to the actual element within the model.

Model Dependency Diagram

A model dependency diagram (shown in figure 3.8) shows what is needed to construct a new case model. It illustrates the dependencies between the various domain model types. The model dependency diagrams are organized in abstraction hierarchies. This implies that it is possible to decompose a model construction activity into a more detailed level.

3.3.5 The Method Perspective

From a knowledge level, perspective methods organize and execute the model construction activities. As mentioned, the method perspective tries to answer HOW knowledge is used and WHEN it is used [Ste91]. As mentioned earlier methods are algorithms that specify how a task gets accomplished. The method contains several activities (some of them might be tasks themselves) and a control flow over activities. The activities have an impact on or consult models. Each model and interface plays a specific role in the method. A method uses models to fill a set of roles.

A method needs to do two things [Ste91]:

1. Map task onto the model construction activities represented in the model dependency diagrams
2. Impose a control structure on various tasks

Methods

The control diagram is a description of the methods that impose control over tasks. Aamodt [Aam01] defines four different types of methods :

- **Task decomposition**
  decomposes the task it is applied to. They divide tasks into subtasks and regulate the flow between them.
- **Task execution**
  executes the task directly. It describes how a task is solved without further decomposition, without further sub-tasks getting accomplished.
- **Control methods**
  controls the sequence in which the subtasks are executed. The sequence is controlled by the interdependencies between the subtasks.
- **Mapping**
  maps tasks to model decompositions.

According to Steels [Ste93] the methods can be decomposed into even more categories. These can be shown as a hierarchical tree. This is shown in figure 3.9.

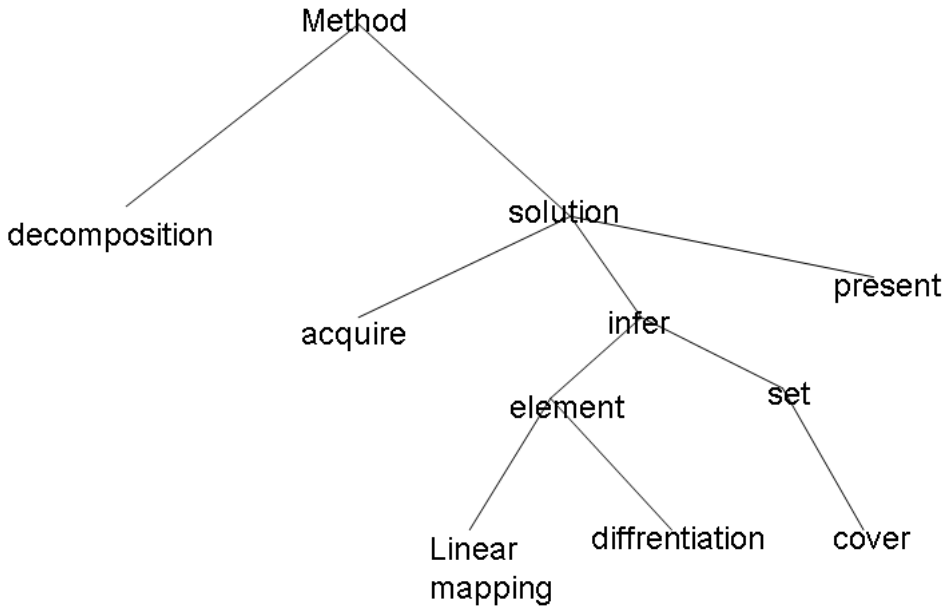

Figure 3.9: Method hierarchy in CoE

Steels differentiate between the acquisition methods that acquire information through user interactions. The solution methods are decomposed into acquire, infer and present methods. The inference methods use other models to infer new models. These can be splitted into elements and set methods.

However, as stated by the differences between Steels and Aamodt, there are several ways of doing this. It is not a goal to have every method represented in the workbench. It would be an extremely large number of methods. An ideal situation would be to use the hierarchy of methods that the user needs. This would be more manageable for the user and possible for the developer of the workbench.

The control Flow Diagram

The method depicting the control structure to the tasks is the control (flow) diagram. As shown in figure 3.10, the control diagrams are finite state automata [Ste93]. The states correspond to the activities (subtasks) in a 1-1 fashion. The transitions corresponds to the control flow (also 1-1 relations), where the conditions regulate when control flows from one state to the other. The control diagram also contains a start, succeed and fail finite states.

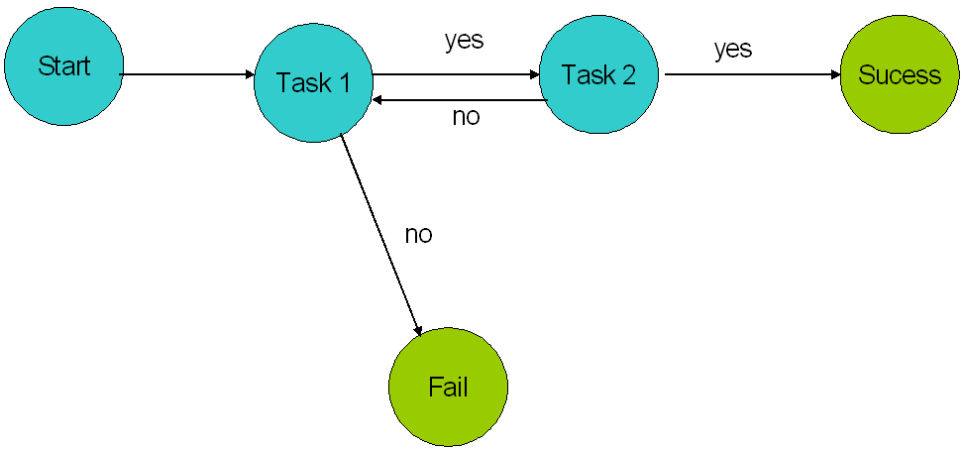

Figure 3.10: The CoE control diagram

A transition between tasks occurs when the activity implementing one subtask has finished [Ste91].

3.3.6 Problem Solving Methods

According to Steels [Ste90] the problem solving methods applies the domain knowledge to the tasks. In general, they perform two distinct functions. The problem solving methods divide a task into subtasks or solve the task directly.

Example of Problem Solving Methods

There is not one unique problem solving method for every task. In addition, when a task is decomposed, each new task has to be analysed in turn. There are several possible problem-solving methods. However, some examples might be:

- linear search
- top-down refinement
- association (between feature and class)
- differentiation
- weighted evidence combination
- distance computing

It is important to notice that each of these methods needs its own type of domain knowledge. An example might be the distance metric needed for distance computing, the hierarchical structure needed by the top-down refinement and so on.

In general the way the tasks are decomposed (the task structure), is essential to the whole analysis.

3.3.7 Relating task feature to solutions

As mentioned earlier, CoE enables the mapping from conceptual features, pragmatic constraints of a task and the available knowledge to components.

When selecting the most appropriate problem solving method, both the conceptual (specifies the input-output relation) and the pragmatic aspect (distinguishing between the different methods) plays an important role. Both aspects are still important when selecting and acquiring the domain model. However, an additional choice when it comes to the depth of how knowledge is represented. The problem-solving method

has a set of roles that need to be filled with specific domain knowledge. One might use domain models as the viewpoint for domain theories. The domain theories are more underlying and sought after in deep (knowledge) expert systems. A problem solving method only uses a part of these theories and might need additional heuristics [Ste90]. However, both the domain model viewpoint and the role viewpoint are complementary and are used according to what knowledge one has access to.

All the aspects discussed in this section have been conceptual. However, there are pragmatic aspects of domain knowledge as well, addressing the constraints in the domain. Like the conceptual aspects, the pragmatic aspects might also bee viewed from two different viewpoints. The aspects can be used in problem solving or as needed for driving the problem solving.

3.4 CommonKADS

CommonKADS is a framework aimed at constructing knowledge level models. CommonKADS is a synthesis of the KADS 4-layer model of expertise combined with aspects from Components of Expertise [WdVSA93]. CommonKADS is a very detailed, formal framework.

3.4.1 Principles

There are several important underlying principles in CommonKADS framework. One of them is the knowledge application principle. This principle states that [WdVSA93]:

> **Knowledge Application Principle** *Effective real-world problem solving is viewed as the rational (or at least, rationable) application by an agent of appropriate domain- and task specific knowledge.*

Another important principle is the modelling principle. The modelling principle recognizes the modelling activity as the heart of the knowledge engineering. The central model in CommonKADS is the expertise model. The expertise model models what knowledge is being applied to carry out a task. This illustrates the problem solving behaviour of the agent.

However, there is a whole suite of models in the CommonKADS methodology. These are [WdVSA93]:

- Organization Model (analyse an organization)

- Task Model (captures the global task within the organization)
- Agent Models (describes agent tasks)
- Expertise Model (agent competence involved in realizing the overall task
- Communication Model (communicating amongst agents)
- Design Model (structure and mechanism of the things involved in the task)

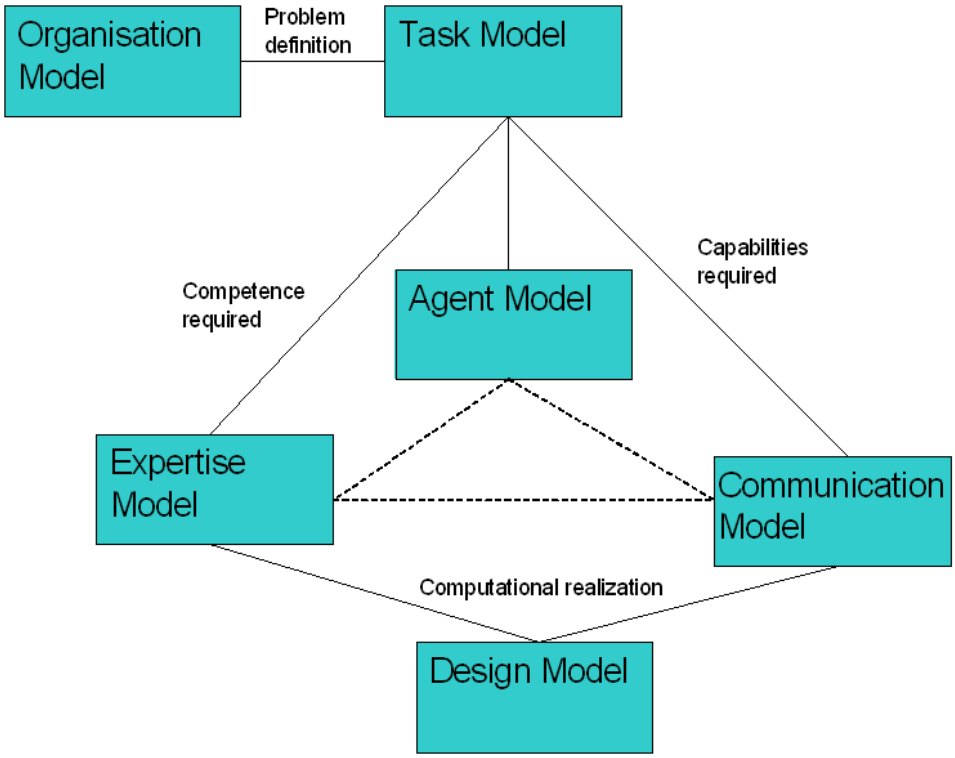

Figure 3.11: The different models in CommonKADS (adapted from [WdVSA93])

All the models have different purposes, different moments in time. They are related to each other as shown in figure 3.11. The expertise model will be used as a guide later in this master thesis.

CommonKADS also uses the knowledge level principle (see section 3.1, 3.2), the role limiting principle (ascribing particular role to a component), the principle of differentiated (two-step) rationality 3.2 and categorizes knowledge into domain, task and inference knowledge (as introduced in section 3.2).

CommonKADS have to be able to handle different strength couplings between the different categories of knowledge (i.e. task-domain coupling). This implies that CommonKADS has to be able to model multiple ontologies and

3.4.2 Problem solving

A case model is the product of the problem solving in CommonKADS. The case model summarizes what the agent understands of the reality. According to Newell's (section 3.1) goals are defined as the desired state. The case model contains enough information for the system to conclude if a problem has been solved / a goal has been reached.

3.4.3 Knowledge Categories

The primary epistemological categories in CommonKADS are domain, task and inference knowledge. These categories are vital to describing the knowledge level.

Domain Knowledge

As mentioned in subsection 3.2.2, domain knowledge expresses relevant knowledge about the systems. Domain knowledge refers to both the system and its class of systems. In KADS-I framework, domain knowledge was not handled properly. KADS-I used primitive representational terms and meta classes. However, CommonKADS here uses the ideas from the CoE who focuses on the domain ontology and domain models.

There are two important features that the ontology should have (according to Lenat and Guha)[LG90].The ontology should be able to make the distinctions needed in the task environment (epistemological adequacy). The ontology should also be able to handle efficient problem solving (pragmatic adequacy). However, these two features might be conflicting. One solution is to use the simplest notions on the domain, and put a number of pragmatic abstractions on top of this simple ontology. This will enable a more effective reasoning. Such a coherent collection of statements is a certain viewpoint on the domain and is called a domain model [WdVSA93].

Task Knowledge

Task knowledge still specifies tasks and goals of the task. A goal of a case is a specification of the case model. In CommonKADS the task definition and task body contain the goal and the different aspects of the task. The task body contains *how* a task is achieved. A task definition contains *what* needs to be achieved. A task is described by an abstract input-output description.

Inference Knowledge

Interference knowledge specifies the possible basic interferences in domain knowledge. They are quite similar to the interference rules in logic where one can derive new information from axioms (domain knowledge).

3.4.4 The Knowledge Structure in CommonKADS

There are several ways of structuring knowledge in CommonKADS. The structure within the agent plays different roles during the problem solving.

Intra-category Structure

The expertise model in CommonKADS divides knowledge into domain, task and inference knowledge. It follows naturally from these three categories that there is also three different internal structures. Domain models and the interrelations between them structures the domain knowledge. The interferences the domain models take a part of govern these. This makes up the domain structure. The interference structure is the structure of the interference knowledge. The dependencies among the basic inference make up the interference. This is the most primitive view on the application [WdVSA93]. The task knowledge is structured as a task structure. This task structure is all contained in the top-level task body. This task body contains the decomposition into several sub-tasks. The basic tasks are linked either to the basic inference or to basic axioms.

Inter-category Structure

In the expertise model the different categories of knowledge are also related to each other. This is called the inter-category structure. The relations between the different elements of different knowledge categories are called knowledge roles. In short a knowledge role is the knowledge item in combination with the vocabulary to talk about the knowledge item [WdVSA93]. One is also able to differentiate between knowledge roles by differentiate between static and dynamic knowledge items. The dynamic ones are characterised by the fact that they change during the problem solving process. One is able to manipulate the dynamic knowledge items. The static ones are not affected by the problem solving process, even if they are being used.

Problem Solving Knowledge

CommonKADS also supports problem solving knowledge. This is knowledge about a how to model an application, describing a way to solve a class of problems. Since this is knowledge about a specific application (problem solving), the problem solving knowledge is not a fourth knowledge category. It is just meta knowledge about how it is possible to build a good model. It is purely epistemological. The problem solving knowledge can be divided into to sub-categories [WdVSA93]:

- Problem solving method (possible to organization of the different types of knowledge). in general a problem solving method consists of a number of possible actions and some way of determining how these actions are ordered in time.
- Strategic knowledge (what kind of model fits a given task environment)

There are different notions of what problem solving knowledge really is. Examples of "problems" that differentiate the notions are the general performance task (what grain-size tasks the methods are applied to) and the way the methods are described. For the general performance task KADS-I composes it into several generic tasks, while CoE can apply the problem solving methods to any task. When it comes to the way the methods are described KADS-I uses Interpretation Models to encapsulate the methods. KADS-I also uses knowledge differentiation and problem space hypothesis [WdVSA93].

In CommonKADS a problem solving method is a prescription of the way a certain task definitions can be satisfied [WdVSA93]. The problem

solving method in CommonKADS specifies/rationalize a relationship between the task definition and the task body. It also opens for recursive problem solving, as the method might lead to new problems.

3.4.5 The CommonKADS Expertise Model

As shown in figure 3.12 the expertise model in CommonKADS consists of several different components.

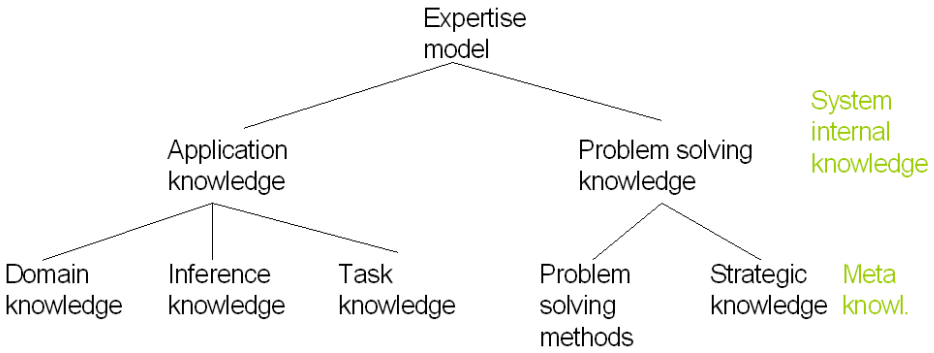

Figure 3.12: The different components of the expertise model (adapated from [WdVSA93])

The expertise model contains two types of knowledge. The application knowledge is how the system is able to reason about the domain knowledge, while the problem solving knowledge is how the system internal knowledge.

Domain Knowledge

In the expertise model the domain knowledge contains a wide range of components. These are shown in figure 3.13. One might make a distinction between the static (case independent) and dynamic (case dependent) knowledge. Static knowledge is the domain models, while the case model is dynamic knowledge. In CommonKADS there are two levels in the modelling language. There are the basic domain language (basic model ontology and the basic model schemata) and the statements within a single domain (domain model schemata, ontology and axioms (meta data)).

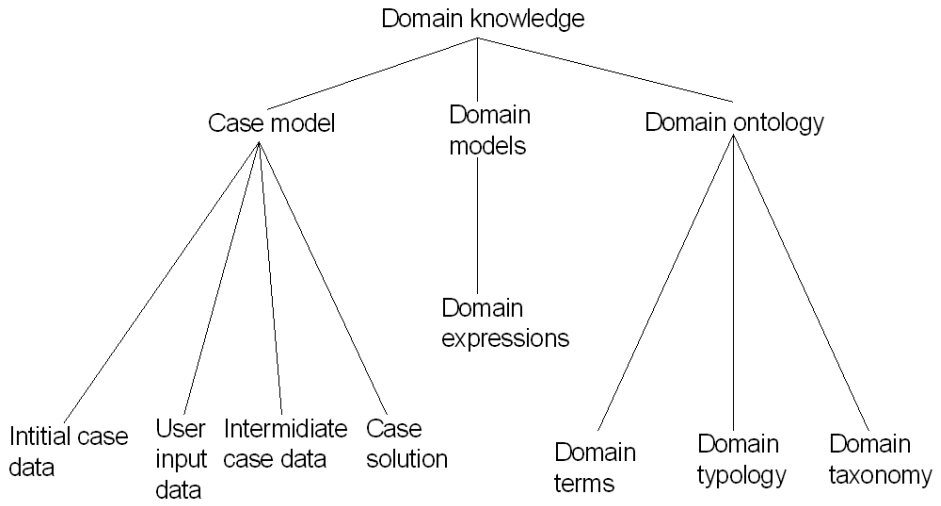

Figure 3.13: The components in the domain knowledge (adapted from [WdVSA93]

The model ontology is the meta description of the elements in the domain model. The ontology describes the vocabulary that represents the domain (however, it does not say anything about the structure). A modelling schema describes the structure of the entities in the domain model. The relations between different domain knowledge components (in an expertise model) are shown in figure 3.14.

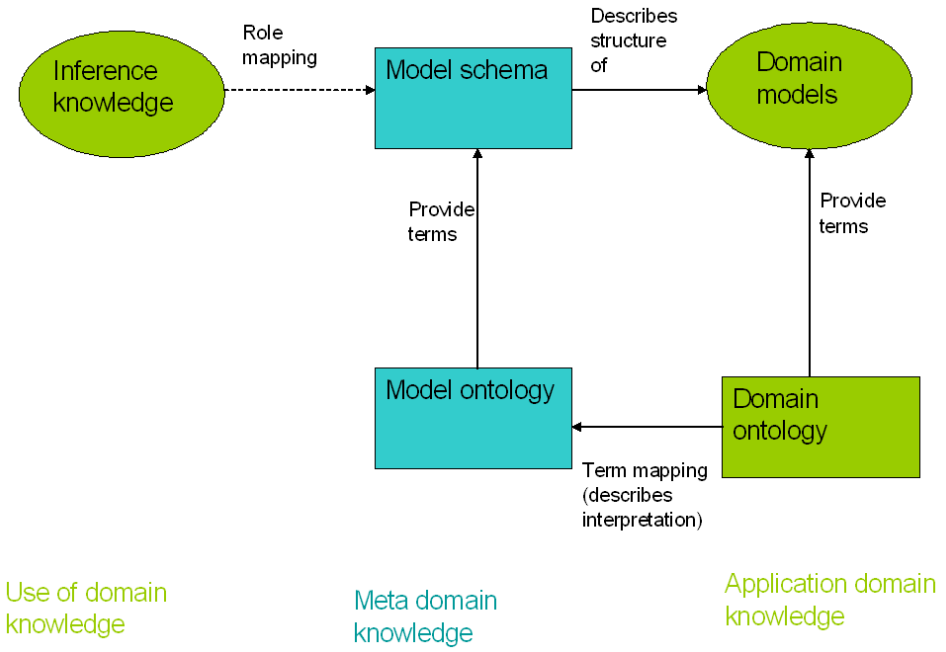

Figure 3.14: The relations between the various domain knowledge components in an expertise model (adapted from [WdVSA93])

Inference Knowledge

The inference knowledge in the expertise model follows the same principle as in 3.4.3. Generally, the reasoning process in a knowledge-based system usually consists of a number of inference steps. The inference uses static domain knowledge to relate items to the case model [WdVSA93]. In CommonKADS the interference relation creates relations between different dynamic domain knowledge roles. The interference relations are non-directional. The interference structure is in many ways a model of the reasoning process.

Task Knowledge

In the expertise model task knowledge still follows the same principles as mentioned in 3.4.3. However, the task definition in the expertise model consists of goal, case roles, task specification. The task body contains the sub-goals, the sub-tasks and the task expression.

In the expertise model one distinguishes between the composite tasks (further decomposable), transfer tasks (tasks of interaction with world) and the primitive tasks (directly linked to interferences)[WdVSA93].

Problem Solving Knowledge

The problem solving knowledge in the expertise model still follows the same principles as described in section 3.4.4. In the expertise model the problem solving method contains a goal, a task characterization, control roles, sub-goals, definition and a task expression schema [WdVSA93]. In the expertise model it is possible to utilize a combination of several criteria, use sub-goals, test criteria and list several assumptions that the method uses. Only under these assumptions is the method acceptable.

However, it is the task features that determine how adequate a problem solving method and a model in general are. CommonKADS solves this by using a library of models and re-usable model components. The library uses task feature as index.

3.4.6 Various Modelling Methods

CommonKADS gives support to various modelling methods by using libraries of generic components (various level of granularity) and providing modelling tools.

The generic components (the basic components for defining a model) offered by CommonKADS are:

- Problem solving methods
- Instantiated problem solving methods
- Generic tasks
- Inference structures
- Generic domain models

The modelling methods provided by CommonKADS are:

- Bottom-up assembly from expertise data
- Model assembly around a problem solving method
- Model assembly from generic components
- Top-down task decomposition
- Knowledge differentiation
- Structure mapping

3.4.7 Further Development

The overall goal for the CommonKADS project still is to create a unified framework for knowledge modelling in knowledge-based system approach [WdVSA93]. The further development of CommonKADS moves in two directions. One branch uses libraries of re-usable models (the CommonKADS library), while the other branch uses formal models and a formal modelling language (FML for CommonKADS expertise models). There are two levels of the formal modelling language. One level to use when implementing models, while there is another level for conceptual design.

IBROW

The IBROW EU (information society technologies) project is a spin-off from CommonKADS. The IBROW project is an intelligent broker for handling web requests. The customers buy classes of knowledge by using libraries of problem-solving methods. The problem-solving methods were accessible on the Internet. The customer could select, adapt and configure the methods to fit the customer's domain better. The goal was to make knowledge-system technology available in larger scale by using library.

Because of the IBROW project, the Unified Problem-solving Method Development Language (UPML) was developed. The goal of UPML is to be able to make it possible to semi-automatic reuse and adaptation when describing and implementing architectures and components. UPML is a framework for developing knowledge-intensive reasoning systems (based on libraries and generic problem-solving components. UPML provides an editor for writing specifications based on Protégé-2000 [CM01].

In IBROW the library of software components are extremely important. IBROW defines the library by using the UPML architecture [WvAAJ03]. The library consists of the three component types task, problem solving method (PSM) and domain model. The library defines two types of PSMs: problem decomposers (decompose task into subtask) and reasoning resources (primitive operators). The ontology acts as glue in the UPML architecture, used to tie the pieces together. Thus, there are a Task Ontology (conceptualization of a particular of a goal specification), a PSM ontology (adding terminology needed to describe the PSMs) and the Domain ontology (domain specific terminology). In UPML mapping between the different ontologies are done by using bridges. IBROW also

has a library for heterogeneous PSMs (heterogeneous problem decomposition methods and reasoning resources that share a common ontology) and a library for configurable PSMs (one or more generic problem solving problem solving components).

3.5 Protégé

Protégé is a knowledge acquisition program (as shown in screen shot in figure 3.15). Protégé is a knowledge acquisition Meta tool for building conceptual models, a knowledge engineering workbench. Protégé is not a tool for building experts systems directly. It builds other tools that are tailor made to assist the knowledge acquisition for expert systems [Mus89].

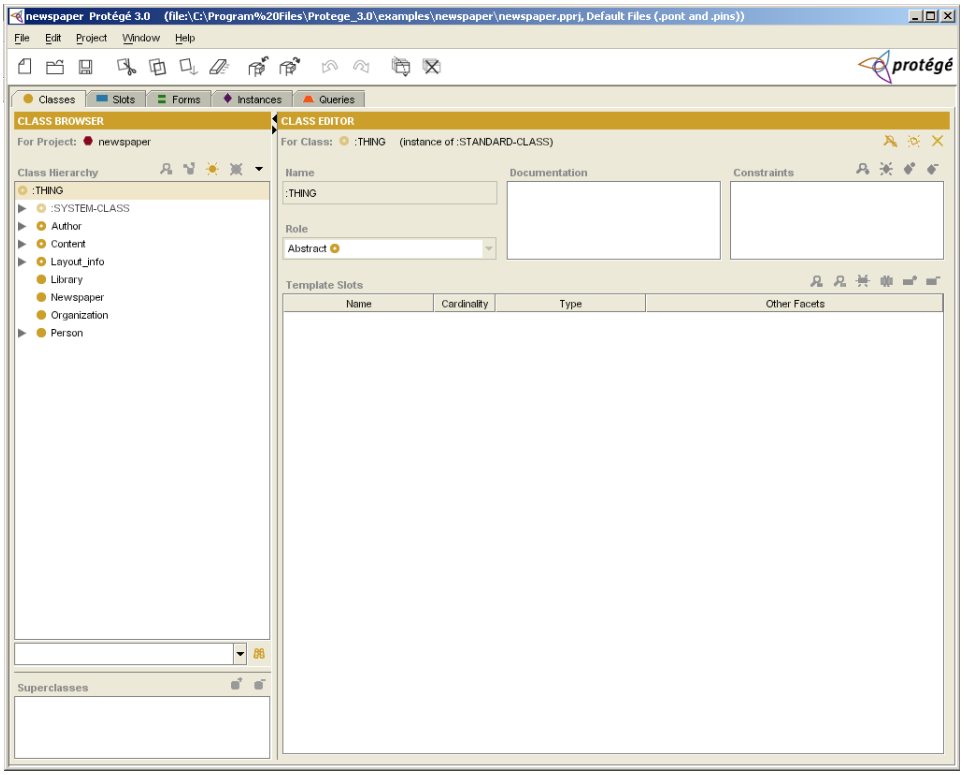

Figure 3.15: Administrating classes in the current version of Protégé

3.5.1 Protégé: fundamental ideas

The Protégé system has undergone major changes over a period of 16 years. However, some features and ideas remain unchanged. These fundamental ideas about knowledge-based system have made up the base of the Protégé system.

- The end user of a knowledge base is a domain expert, not necessarily a knowledge engineer.
- The domain-specific knowledge acquisition tools generated from underlying domain model or ontology
- The knowledge engineer does the structural domain modelling and tool design. The domain expert fills in detailed domain knowledge.
- One should capture domain knowledge declaratively, without and references to a specific problem solving method or inference
- The inference methods can be isolated as problem solving methods or as plug in applications

However, Protégé is based on class-subclass connection. Protégé contains little causal relations.

3.5.2 Method to Task

The central concepts in Protégé is tasks and methods. Musen et al [PTM93] defines tasks as an activity/an abstraction of an activity in the real world. A task accepts a type of input and produces an output, decided by what domain the task is applied to. However, the task in it self does not give any knowledge requirements.The task does not say anything about how the output is produced. This is not until a method solves a task the knowledge requirements are set.

A method imposes a task decomposition to tasks. The task decomposition results in subtasks. The subtasks are then either solved by mechanisms (methods that does not decompose tasks any further) or further decomposed by methods. in ProtégéII a mechanism is a black box, that given the inputs of subtasks produces the output of that subtask.

What methods becomes a mechanism or not is decided by if decomposing the method does not provide corresponding mechanisms for each subtask that can easily be reused. Then the method is not decomposed and becomes a mechanism. There is no restriction on using the mechanisms directly on the tasks that has not been decomposed [PTM93]. The methods doing the task decomposition only enumerates the tasks.

It does not specify the execution ordering or any other control specification. Mechanisms are method independent, reusable and limited to solving resulting subtasks. The control details can be presented only at the method level[PTM93]. There is a clean distinction between the input/output specification of the task and the control specification for a method. The division enables the two method manipulation operations: method configuration (electing appropriate solve each subtask, verifying the relationships between the method-level and mechanism-level) and method assembly (viritual creation of a new method). These two method manipulation operations are the building blocks of ProtégéII [PTM93].

3.5.3 The Developing Protégé

Protégé has undergone a continuous development since Mark Musen first introduced the meta-tool in 1987. Figure 3.16 shows the development of protégé.

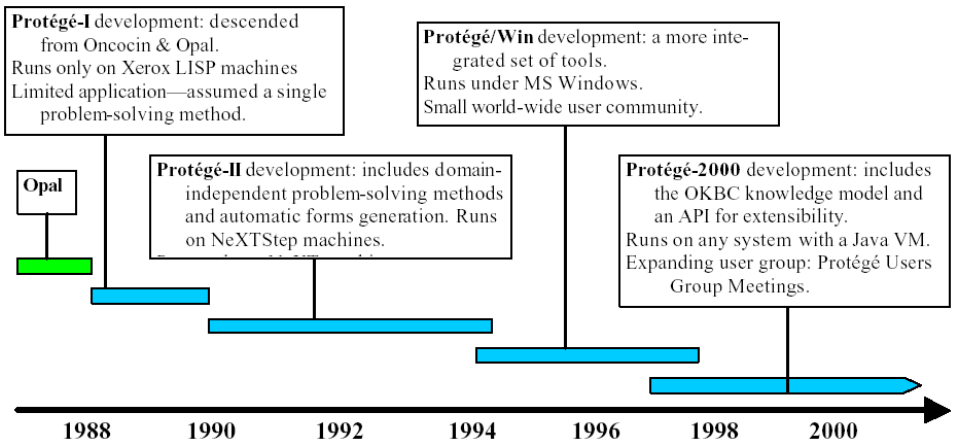

Figure 3.16: The history of Protégé as illustrated by Musen[GMF+02]

As one can see from figure 3.16, there has been three major developments in the Protégé program. In short the major differences are:

ProtégéI

The initial version of Protégé introduced the notion of generating knowledge acquisition tools from structured meta-knowledge. Protégé used a specific problem-solving method called episodic skeletal-plan refinement (ESPR). ESPR assumes that a top-down modelling of the task

hierarchy of plan components. ESPR was used to make method specific knowledge bases (for use with skeleton planners only). Protégé lacked a formal semantic that would enable the use in different settings [GMF+02]. Due to ESPR ProtégéI used the semantics of the target application area [Mus93].

ProtégéII

The limitations, due to the assumptions in ProtégéI, called for a generalization of the system. ProtégéII does not presume the use of ESPR or any other particular problem solving method. The developer might choose what method to use from a library (reuse of problem-solving methods as components). This enables the possibility of using the method most suitable for each domain and the application requirements.

In ProtégéII, the ontology plays an important role and the ontology uses a more formal representation language (frame based). It is able to generate knowledge acquisition tools from any ontology (in contrast to just being able to use instantiations of ESPR). ProtégéII operates under the assumption that the ontology is more durable than the knowledge bases [GMF+02], a "downhill flow" assumption.

Protégé/Win

The Protégé/Win version was an attempt to broaden the user community and make the system easier to use. Protégé/Win had some new features (such as ontology inclusion), however it was mostly the same features as in ProtegeII [GMF+02].

Protégé2000

Protégé2000 is the current version of the Protégé system. In Protégé2000, there is a strong focus on scalability. The system is able to handle the development of larger knowledge bases than its predecessors are. It is a single tool with more flexible knowledge-model. Protégé2000 is also more extensible, both the system in general and the user interface [GMF+02]. UMPL uses a version of Protégé2000 is for to create specifications in UMPL (as mentioned in section 3.4.7).

3.5.4 Ontology Modelling in Protégé

There is actually a guide on how to build an ontology in Protégé [NM00] written by Noy and McGuinness. The guide is a hands-on explanation of how to model in Protégé with no particular focus on knowledge acquisition modelling. The guide builds on experiences from using Ontololingua and Chimeara in addition to Protégé. The guide defines five reasons for making an ontology [NM00] (from the Protégé perspective):

- Sharing common understanding of the structure of information among people or software agents
- Enabling reuse of domain knowledge
- Making explicit domain assumptions
- Separating the domain knowledge from operational knowledge
- Analysing domain knowledge

The main elements in the modelling approach in Protégé are to [NM00]:

1. defining classes in the ontology
2. arranging the classes in a taxonomic (subclass-super class) hierarchy
3. defining slots and describing allowed values for these slots
4. filling in the values for slots and instances

The knowledge-engineering methodology in the guide consists of seven steps:

1. Determine the domain and scope of the ontology (using competency questions)
2. Consider reusing existing ontology
3. Enumerate important terms in the ontology
4. Define the classes and the class hierarchy
5. Define the properties of classes-slots
6. Define the facets of the slots
7. Creating instances

3.5.5 Recent extensions

Protégé is still very alive and used. It is more or less the leading modelling tool in Europe. And much of its popularity is for instance conected to the fact that it is quite simple to add plug ins. One example of recent extensions is the implementation of the PROMPT framework for Protégé [NM03]. PROMPT is a uniform framework for working with different ontologies. PROMPT aligns, compares and merges different ontologies. It also maintains different versions of the ontologies and translates between different formalisms. PROMPT is an extension to the Protégé ontology-editing environment.

Protégé evolves through the involvement of international users that adds capabilities and extends the program.

Chapter 4

The ki-CBR system TrollCreek

This chapter is an introduction to the knowledge intensive case-based reasoning (ki-CBR) system TrollCreek. The chapter starts with a short introduction to ki-CBR, before going into the TrollCreek system. Important features in the TrollCreek system are used later when introducing the knowledge acquisition and modelling method for TrollCreek. The tools/different views, a short introduction to the CBR process and similarity measurements, the existing modelling approaches and the predefined top-ontology will give the reader an understanding of the system.

4.1 Knowledge Intensive Case-Based Reasoning

Knowledge intensive case-based reasoning (ki-CBR) is a machine learning technique. What makes ki-CBR unique is the combination of a case base and general domain knowledge. The combination enables ki-CBR systems to use both semantic and pragmatic criteria when reasoning. Several other pure CBR systems are only able to perform reasoning with syntactical criteria. Ki-CBR systems are able to match for example two cases with different names, but the same meaning (based on the deep domain knowledge in the domain ontology).

As shown in figure 4.1, ki-CBR is in between the pure CBR and the pure MBR (model based reasoning). Ki-CBR uses elements from both MBR and CBR.

Figure 4.1: KI-CBR compared to CBR and MBR

4.1.1 The CREEK ki-CBR Framework

The CREEK framework is an example of such a ki-CBR framework. CREEK is a collection of modules all integrated in a conceptual basis in the General Domain Model [Aam04]. Each module represents a sub-model of knowledge. The main modules are:

- object-level domain knowledge model (real world relations and entities)
- strategy level model
- meta level modules (one for combining CBR and MBR, one for combining different learning methods)

Situation specific experiences are stored in the frame-based case base. The concepts are submerged in an interconnected knowledge model.

4.2 TrollCreek

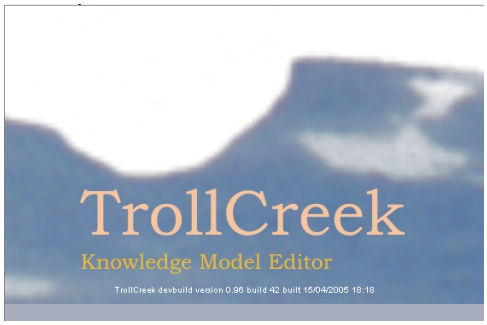

Figure 4.2: TrollCreek

TrollCreek is a java based realisation of parts of the CREEK architecture. TrollCreek is developed by Trollhetta. TrollCreek consists of a knowledge editor application (as shown in figure 4.3) and a case matching module.

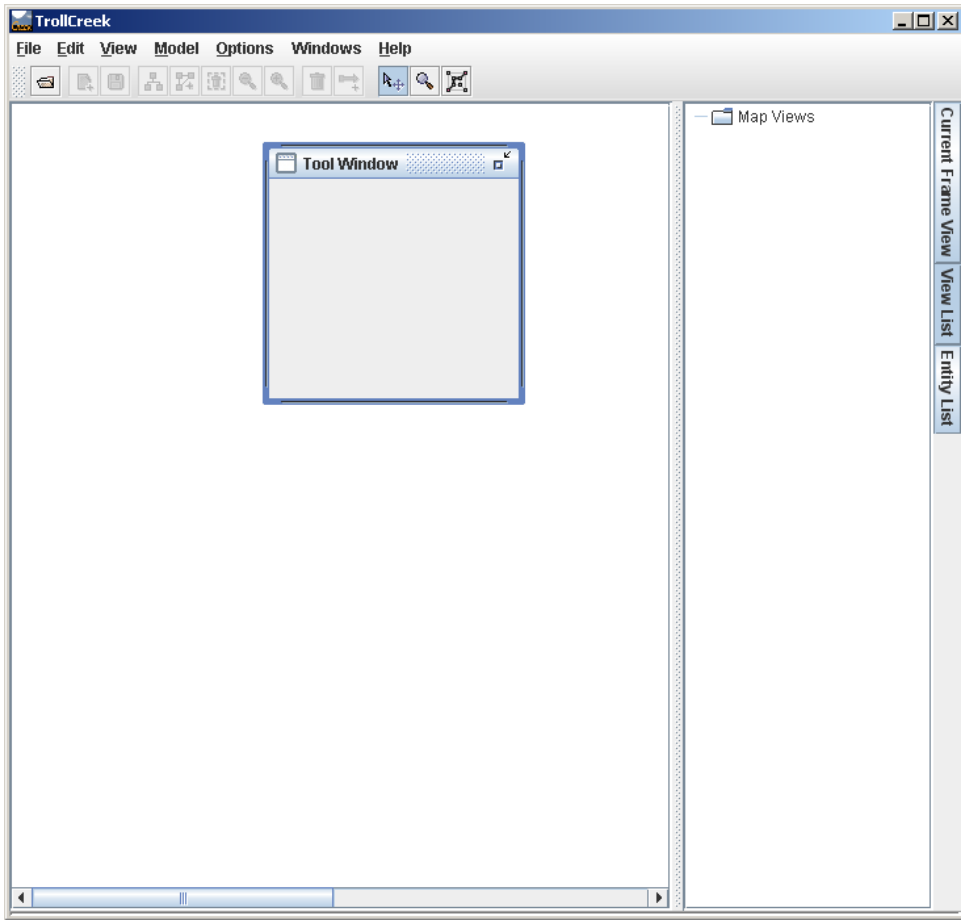

Figure 4.3: The TrollCreek knowledge editor

There are several different ways of looking at the TrollCreek knowledge base. There are different "views" visualizing different aspects of the knowledge base. The "views" also supplies the user with different ways of manipulating the knowledge model. However there are differences between the views:

4.2.1 Map View

The map view (as shown in figure 4.4) visualizes the relationships between entities (as nodes with links connecting them), and the entities in it self. The user might manipulate the knowledge model by adding entities from the "entity list" and deleting entities from the visualisation.

When using the "build tool" the user might manipulate the actual knowl-

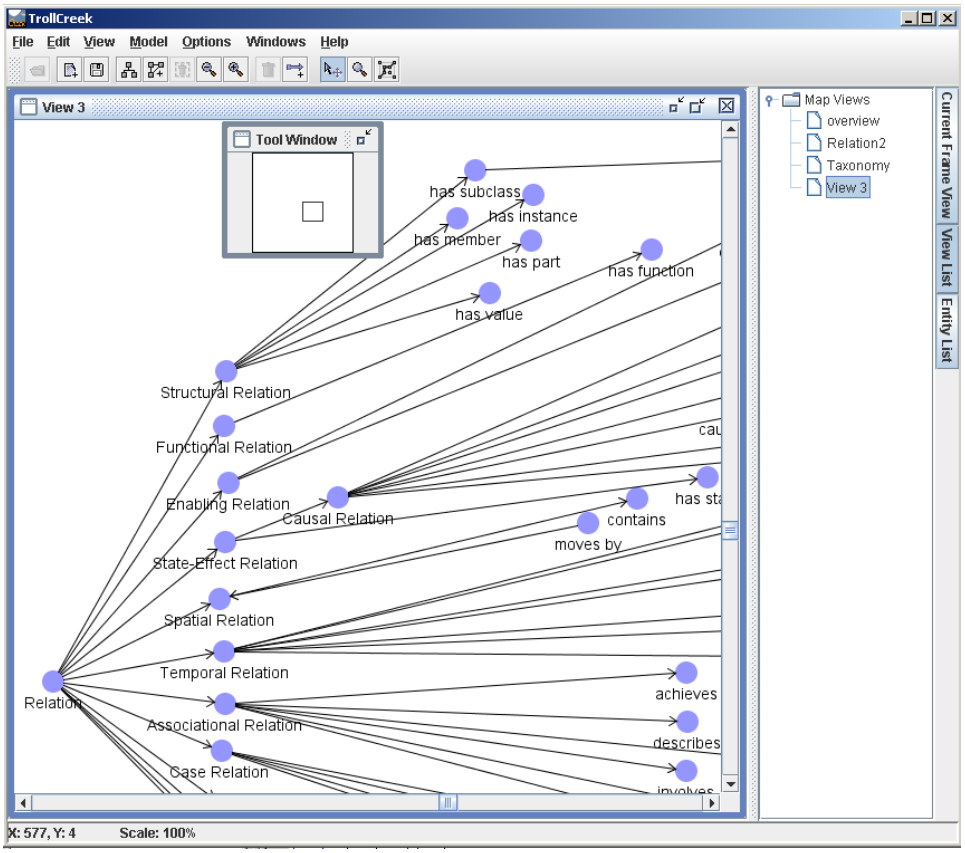

Figure 4.4: Example of a TrollCreek map view

edge model (adding/deleting relations).

Event though this might sound/look like domain knowledge purely the graph also includes cases and states as nodes. It is

4.2.2 Frame View

The frame view (shown in figure 4.5) enables the user to display (using select tool) and change the properties of (build tool) of entities. It is also possible to change/add/delete relationships.

The case view gives the same options for modifying the case as the frame view. However, it pops up as an independent window (shown in figure 4.6).

In the CREEK framework and in TrollCreek there are strong couplings between the cases and the domain ontology. TrollCreek assumes utilizing general domain knowledge to enrich the cases. The cases are com-

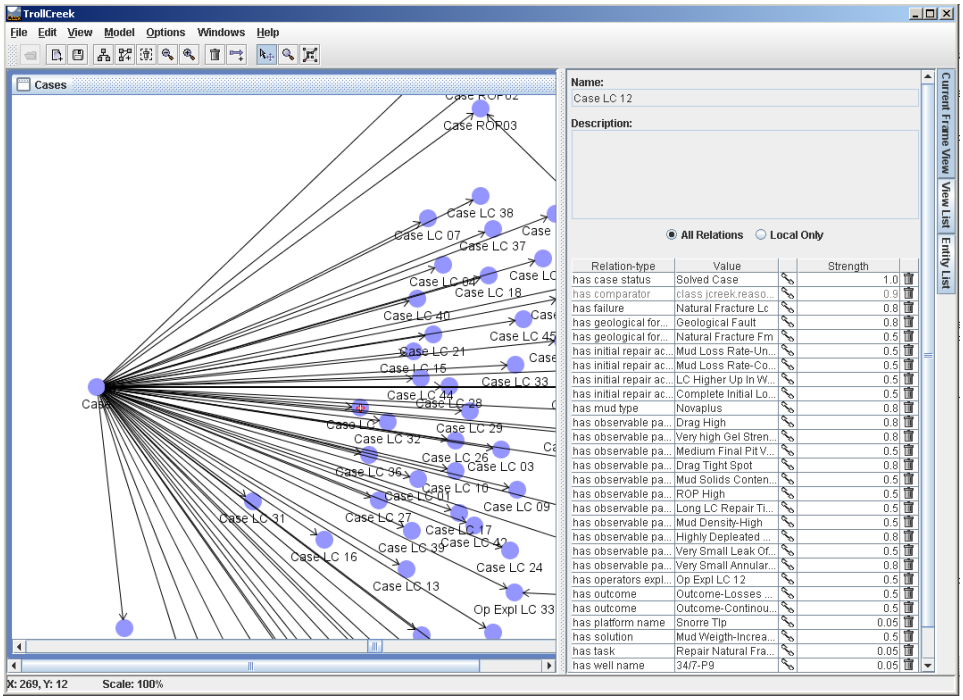

Figure 4.5: Screen shot from TrollCreek showing the frame view

pletely submerged into the ontology [Aam04]. However, TrollCreek is also able to do pure case matchings in addition to the ki-CBR matching involving general domain knowledge.

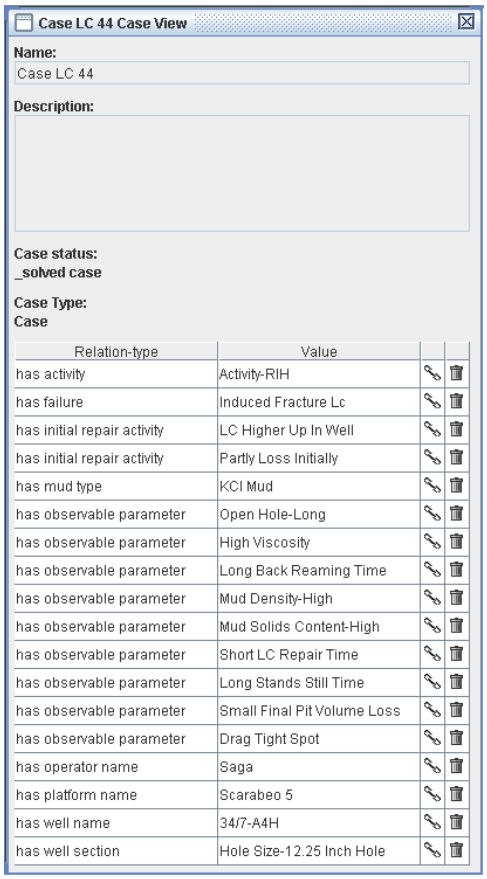

Figure 4.6: A screen shot of a case view from TrollCreek

4.2.3 The CBR Process and Explanation Engine

In CBR the current situation is compared with previously experienced cases (stored in a case-base). CBR adapts the previous cases to fit the situation and proposes a solution. Aamodt explains in his article [Aam04] the explanation engine in the CBR process. It is a three step process that

1. activates relevant parts of the knowledge model
2. generating and explaining the derived information (within the activated knowledge structure)
3. narrowing down and selecting a conclusion that matches the goal

The three step explanatory engine gets repeated for retrieve, reuse and retain. The explanation engine is specialized for each of the four system tasks. (The Revise task is not defined in TrollCreek. It still has to be preformed by i.e. a domain expert.)

Figure 4.7: The CBR process (from [Aam04])

4.2.4 Similarity Measurements

As TrollCreek is today, the attributes need to be identical in the initial attribute matching process (syntactical). As described in [Aam04] the equation for $sim(f_1, f_2)$ is just given by:

$$sim(f_1, f_2) = \begin{cases} 1 & \text{if } f_1 = f_2 \\ 0 & \text{otherwise} \end{cases}$$

An important feature with this similarity metric is that it only treats symbolic matching.

There is also a simple numerical measurement for comparing two numerical values. It just takes the differences between the numbers and normalizes an updateActivationStrength.

The explanatory matching is more complex and in the second stage of the matching process.

4.2.5 Existing Modelling Approaches In TrollCreek

Today, there exist two documents aiding the user when modelling in TrollCreek.

The TrollCreek Tutorial

The TrollCreek tutorial [BSAB04] is a more technical description of how to use TrollCreek. This tutorial contains detailed information about the different views. It also contains how to use the different tools in TrollCreek.

However, there is a section on "How to model in TrollCreek". The tutorial defines a knowledge model as the resulting structure after given the system the knowledge needed to reason about cases. The tutorial goes defining a simplified methodology, with three steps:

- **Making taxonomy**
  extract terms about the domain they are modelling and entering it into the system
- **Building a causal model**
  building a model describing the cases and effects.
- **Adding cases**
  adding the concrete previous experiences.

When making the taxonomy the tutorial also includes a step where one should also make other hierarchies (i.e. compositional hierarchies). A taxonomy usually only includes has-subclass and has-instance relations. The compositional hierarchies would include i.e. has-part and subclass of. It will be more correct to call them ontologies. The making of the ontology defines the vocabulary for later use. It is some overlap between the different steps.

The tutorial uses examples from the car domain throughout the whole tutorial. It also illustrates the modelling approach in the car domain. However, the modelling approach is highly superficial.

Modelling in the petroleum Engineering Domain

This document is also written as a practical tool for modelling in TrollCreek. It is an ad hoc solution in order to complete the practical side of modelling. The document is a tool for the engineer not interested in knowledge acquisition, just interested in making a model that works.

The document starts with explaining how to download the tool, explaining what a TrollCreek model is and how to build ontology and cases. It continues with how to compare the cases and how to test the model.

This document defines a TrollCreek model as a bi-directional graph of entities and the relations between them. A core model consists of a structural model that shows the subclasses, the part and the instances of the entities. The relations can be divided into subclasses (structural, implicational, associative, temporal or role (not yet implemented)). The different relations have different strengths. Since the graph is bi-directional i.e. the "has subclass" relation automatically has a "is subclass of" relation and so forth.

In TrollCreek there already is a default "top level-ontology". This has to be expanded in each case by adding new levels downwards. The document goes on describing the building an ontology process click by click. It goes on explaining the structure of cases for the oil-drilling domain (containing characteristics, observed parameters, solution path, outcome and the operators experience or lessons learned).

4.2.6 The Predefined Ontology Model in TrollCreek

As already mentioned, there is a predefined ontology in TrollCreek. This ontology is the basis of every model made in TrollCreek, and it is shown in figure 4.8. All the relations in the predefined top-ontology is "has-subclass" relations. (The screen shot shows the model called Taxonomy. However, the model might contain other relations that has-subclass and has-instance. It might there for be more correct to call it an ontology.)

As can be seen from the TrollCreek screenshot in figure 4.8 the root node "thing" is decomposed into "relation", "entity" and "descriptive thing". The ontology describes how the knowledge is structured in TrollCreek, and it is therefore important to have a look at the three different branches:

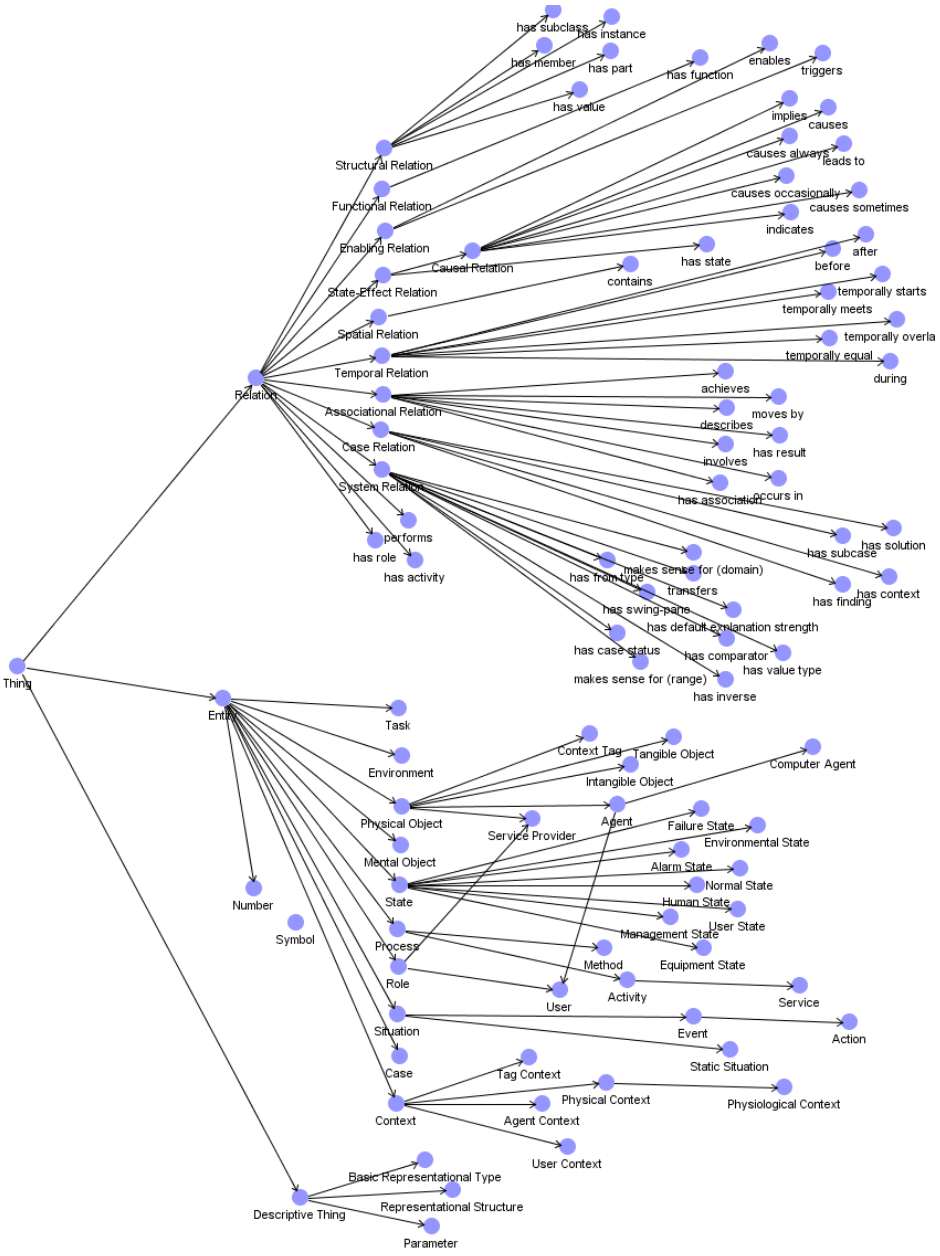

Figure 4.8: The top-ontology in TrollCreek

Relation

Figure 4.9 shows the definition of different types of relations in this part of the ontology. The relations govern the relationships between entities when modelling the real world domain.

Figure 4.9: The "relation" branch of the TrollCreek ontology

Entity

The entity branch shown in figure 4.10 contains both task knowledge and domain knowledge (following the principles in section 3.2). Besides tasks the entity branch contains environment, physical and metal objects, state, process, role and context (domain objects). In this version of the taxonomy there is discrimination between the case and its contents. There is a separation between the framework of the actual contents is

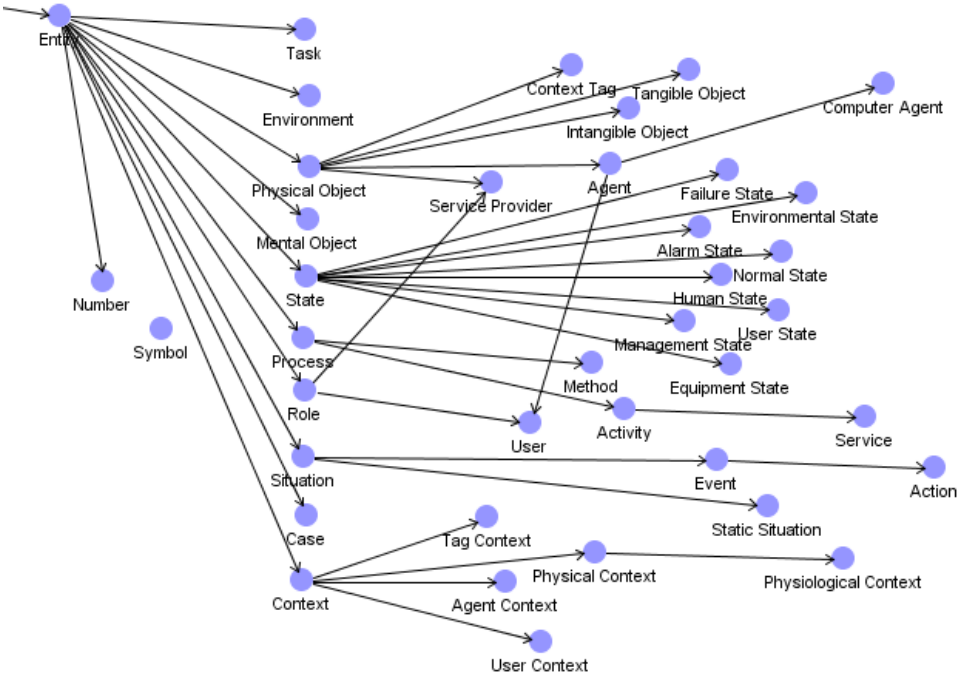

Figure 4.10: The "entity" branch of the TrollCreek ontology

put into and the actual contents (the situation). Section 7.3 presents a possible solution.

Descriptive Thing

Both "entity" and "relation" is a description of the real world, of external things. However, "descriptive things" (as shown in figure 4.11) describes the systems internal things. The problem solving uses these. One example is the case structure.

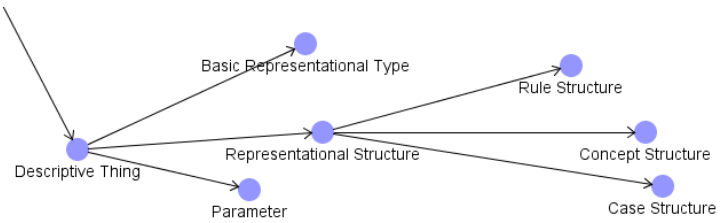

Figure 4.11: The "descriptive thing" branch of the TrollCreek ontology

Chapter 5

Comparison and Analysis

This chapter contains the possible solutions to the knowledge acquisition and modelling task in TrollCreek, a knowledge intensive CBR system. These possible solution are based on related research. The last part of this chapter handles the possibilities of combining modelling approaches and ideas from the KA-modelling frameworks Components of Expertise and CommonKADS.

5.1 Problem Solving Methods

The problem solving method within the KA system is the interference engine. The problem solving method decides on identification, selection and implementation of a sequence of actions in order to fulfil a goal within a domain[McD88]. There are at least two different categories of problem solving methods. There are the weak methods and the role-limiting methods. The role-limiting methods are not so broad as the weak methods. However, the role-limiting methods have little task-specific control knowledge. This makes the role limiting methods a good foundation for KA tools. It enables the developer to know in advance the control knowledge a method will use. It is also possible to know what kind of task-specific knowledge will be needed [McD88].

5.1.1 Different Problem Solving Methods

John McDermott compares different problem solving methods used in KA tools in [McD88]. All the problem-solving methods are role-limiting methods. McDermott introduces 6 KA tools with their respective problem solving methods. McDermott lists the different characteristics with

the tasks that make that particular KA tool and problem solving method possible to use. He also mentions what types of task-specific knowledge the experts have to provide to each KA tool / problem solving method [McD88]. The following tables summarize McDermott's findings:

KA tool	MOLE
Builds	Diagnostic expert systems
Role-limiting method	*cover-and-differentiate*
Task characteristics	1. Identifiable set of problem states/events (explained and accounted for) 2. Exhaustive set of candidate explanations 3. Discriminating information for candidate explanations 4. Only one candidate explanation applicable at any time per event
Experts must provide	1. Complaints/abnormalities 2. Explanations 3. Differentiation knowledge

Table 5.1: MOLE KA tool with problem solving method

KA tool	YAKA
Builds	Diagnostic expert systems
Role-limiting method	*cover-and-differentiate and qualitative reasoning*
Task characteristics	1. Set of problem states/events (explained and accounted for) 2. Model of normal functioning system 3. Qualitative equations between state variables 4. External candidate explanations affecting equations 5. Discriminating information for candidate explanations 6. Only one fault is applicable at any time per event
Experts must provide	1. Library of generic equations and faults 2. Structural model of system 3. Refinements of functional model 4. Description of faults 5. Differentiation knowledge

Table 5.2: YAKA KA tool with problem solving method

KA tool	SALT
Builds	Constructive expert systems
Role-limiting method	*cover-and-differentiate*
Task characteristics	1. Procedures specifiable to determine starting point 2. Constraint and remedies specified by changes to design 3. Not high level of conflict between design extensions
Experts must provide	1. Procedures for obtaining initial value 2. Procedures for obtaining constraints 3. Local remedies for violated constraints

Table 5.3: SALT KA tool with problem solving method

KA tool	KNACK
Builds	Expert systems (WRINGERs)
Role-limiting method	*acquire-and-present*
Task characteristics	1. Report used to document task 2. Small set of concepts cover all reports for a task 3. Report is essential mean for documenting task
Experts must provide	1. Domain model 2. Sample report 3. Sample information gathering strategies

Table 5.4: KNACK KA tool with problem solving method

KA tool	SEAR
Builds	RIME methodology systems
Role-limiting method	*developers define a set of methods*
Task characteristics	1. No role-limiting method suitable 2. Possible to define a set of methods for performing tasks 3. Challenging to decide what to do next based on task details
Experts must provide	1. Collection of problem-solving methods

Table 5.5: SEAR KA tool with problem solving method

KA tool	SIZZLE
Builds	Expert systems
Role-limiting method	*extrapolate-from-similar-case*
Task characteristics	1. Large collection of validated cases available 2. Notion of overall similarity between different cases 3. Knows how to adjust case solution to match case changes in the case problem 4. The nature of the problem is to determine needed quantities of resources for some process (precise nature of process not well understood) 5. A large set of factors to be considered 6. Quality of solution can be classified as better or worse
Experts must provide	1. Sized cases 2. Case indexing knowledge 3. Extrapolation knowledge

Table 5.6: SIZZLE KA tool with problem solving method

A SIZZLE built system will use its task-specific knowledge to guide it through an iteration of steps [McD88]:

1. Ask for enough information about a specific sizing problem in order to identify other, similar solved cases in the knowledge base.
2. Use the differences between the solved and the unsolved case to extrapolate a new solution from a known solution.

The expert builds a sizer by building a indexing discriminating tree of case features when specifying the sizing cases and user models. The rule generator provides this capability (translates source files of case features, cases and user resource demand. There is also an included feature in the sizer that permits the user to define their own sizing cases and test their performance on the system.

5.1.2 Task Decomposition Methods

There are also other general problem solving methods (PSM) than the ones mentioned by McDermott [McD88]. Some PSMs might even apply to non-construction tasks. A definite example of such PSM is the methods used for task decomposition. This section introduces a couple of task decomposition methods. They are all possible solutions to the task decomposition part of the TrollCreek KA method. These methods were introduced as a part of the CommonKADS library in [ABB+93], in appendix E.

Divide-and-Conquer

> "The control of a large force is the same principle as the control of a few men: it is merely a question of dividing up their numbers."
>
> *— Sun Zi, The Art of War (c. 400 C.E.), translated by Lionel Giles (1910)*

Divide-and-conquer decompose a task into subtasks. These subtasks are handled independently. The result from each subtask makes up the total solution of the task. The control between the subtasks is sequential. One way of using divide-and-conquer is to divide the functions up into groups. Divide-and-conquer is fits a wide variety of tasks (i.e. construction tasks, diagnosis, generic and so forth).

The divide-and-conquer algorithm [CLRS01]:

1. **Divide** the problem into subproblems

2. **Conquer** the subproblems by solving them recursively

3. **Combine** solutions to subproblems into the solution for the original paradigm

Divide-and-conquer helps coping with the complexity of the task. It is easier to handle small, independent parts than handling the whole problem. The method is simple and widely used. However, divide-and-conquer is less appropriate when the tasks interfere a lot with each other.

Progressive-Refinement

As the name suggests the progressive-refinement method is a continuing process of gradually refinement of the task. Iteration makes the granularity of the task description gradually finer. First one makes the abstract task. Then one presents the solution. Then one refines the task by adding more details, which produced a more detailed solution and so on. The control is sequential. However, iterative loops refining the abstract description might be necessary.

This method differentiates it self from i.e. divide-and-conquer by not decomposing the task into subtasks (that then constructs the final solution). The Progressive-refinement works on the total solution each time (when refining the solution). It does not decompose the task into partial tasks, but each step produces a more detailed description.

Example of the progressive-refine method used is the hierarchical-design (when applied to design tasks) and hierarchical-planning (when applied to planning tasks) [ABB+93].

Propose-and-Revise

Propose-and-revise decomposes a task into three subtasks:

1. proposing a in initial solution

2. assessing the proposed solution

3. revising the solution (based on the assessment)

The control is sequential, except if it is impossible to revise the proposed solution because it is to fare off. Then there will be a loop in the control flow.

This method is a perfect match with CBR (explained in section 4.2.3). However, it only handles the reasoning part of the CBR cycle. It is not concerned with the learning part of the CBR process.

5.1.3 Task Execution Methods

Another important breed of PSM are the task execution methods. They execute the leaf nodes of the subtask hierarchy (the non-decomposable tasks). Aamodt mentioned these methods in [ABB+93].

When executing a task the task has to be either further decomposed, or if that is not possible the task has to be executed. For the actual execution, there are two possible types of methods for doing this: the mapping methods and the expansion methods.

Mapping methods

Mapping methods use one or more case models to define case models (new single or a new set), illustrated by figure 5.1.

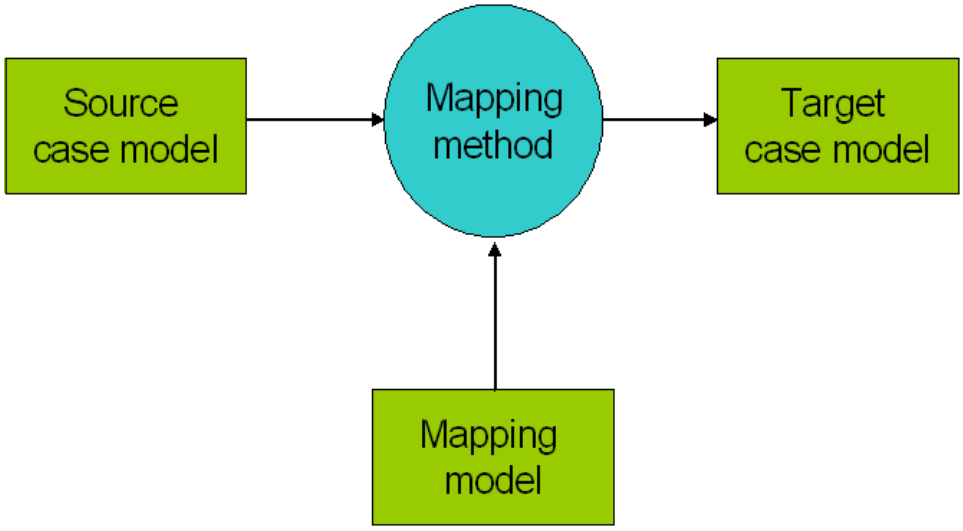

Figure 5.1: Mapping method uses a mapping (domain) model to construct target model (adapted from [Ste91])

One example of such models is the mapping from a problem case model to a solution case model [ABB+93]. Other examples are:

- *Linear-Mapping* (maps many input descriptors of a task to one output descriptor)

- *Differentiation* (identifies the most discriminating hypothesis on a differential, uses this to support evidence)
- *Top-Down-Refinement* (use domain class tree for downwards, stepwise discrimination)
- *Weighted-Evidence-Combination* (weights evidence item according to relative importance to each hypothesis)
- *Similarity-Matching* (uses metric to compute similarity between source and target case)

Expansion Methods

The expansion methods change the contents of case models. It develops a single case model. Constraint-Propagation is an example of such a method. Expansion method modifies the case model by propagation constraints imposed by the domain model.

5.1.4 Combining the PSM with TrollCreek

There are several possible ways of combining the problem solving methods with TrollCreek. However, there will be some problem solving methods more suitable than others. TrollCreek, being a ki-CBR system (mentioned in detail in section 4.2), will not have the same need to use i.e. quantifiable equations in order to reason about a domain. The most obvious possible solution is SIZZLER and the "extrapolate-from-similar-case" problem solving method. Section 6.1.2 discusses this in more detail. However, the KNACK KA-tool handles reports. When applying TrollCreek when building a system for i.e. the oil-drilling domain, daily drilling reports will be a important part of the knowledge. It might then be a possible solution.

When modelling in TrollCreek it is possible to combine several task decomposition and execution methods. All though divide-and-conquer is a favourite task decomposition method, just as a combination of Weighted-Evidence-Combination and some other method is for execution method. However, the TrollCreek system already has the execution built into it. It is used every time a case is matched.

In section 3.4.7 I introduced IBROW with its library structure and use of UPML. IBROW combines several PSMs using libraries (one for heterogeneous PSMs and one for configurable). The library structure allows IBROW to choose between several methods each time. This is one of many possible ways of including a variety of generic PSMs in a system.

5.2 TrollCreek and KL-modelling frameworks

In chapter 3 several KL-modelling frameworks were introduced. There are several possible ways of combining these frameworks with TrollCreek. However, the most relevant solutions would be to combine TrollCreek with CommonKADS (in section 3.4) or CoE (in section 3.3). This section discusses these two solutions briefly, before a choice will be made and the result described in more detail in section 6.2.

5.2.1 A Case Is Not A Case

It is important to be aware of the bewildering use of the word "case". A case in the CoE setting is not the same thing as a case in TrollCreek. A case in TrollCreek is more a part of the domain model in CoE (it is NOT the same thing as a case in the case model!). The CoE case model is the current description of the world, the situation as it is at THAT particular moment in TrollCreek. The system is working on at that moment the situation. To make the confusion complete, the current situation (the CoE case model) will be the "current situation" case in TrollCreek (when running the program).

This is extremely important to have in mind when discussing the combination of TrollCreek and both CommonKADS and CoE.

5.2.2 CommonKADS + TrollCreek

As mentioned in section 3.4.1 the CommonKADS framework contains a whole suite of models. When trying to combine the principles of CommonKADS with TrollCreek, the focus is the expertise model. The expertise model is the model that captures the agents competence involved in realizing the overall task. In the TrollCreek sphere, there will not be use for the more organizational oriented/higher level models.

The three Types of Knowledge

As mentioned TrollCreek has domain knowledge, relations between the entities and task knowledge in the ontology. Domain knowledge and interrelations (governed by inference knowledge) between them makes up the internal structure of the domain model in CommonKADS. The tasks are organized in a task-subtask hierarchy.

One difference between the knowledge structure in TrollCreek and CommonKADS is that the TrollCreek ontology contains all three different types of knowledge. Just as in CommonKADS, in TrollCreek a task-subtask hierarchy organizes the tasks. Interference knowledge is represented trough the relations between the entities in the domain for example trough a "causes" relationship. In CommonKADS, there is much more separated structure to knowledge. In the TrollCreek ontology is a spaghetti western when it comes to mixing taxonomy, typology and terms in one network.

The CommonKADS Expertise Model

The Expertise Model can be divided into application and problem solving knowledge (as mentioned in section 3.4.5). In many ways, the expertise model resembles the ontology in TrollCreek. The expertise model divides knowledge into system internal reasoning and the more external reasoning (given domain knowledge). Therefore, the expertise model has a different structure to its domain knowledge than the TrollCreek ontology.

TrollCreek being a ki-CBR system, the cases play an extremely important role. In TrollCreek the cases are described as a part of the ontology structure. CommonKADS have "Case Models", a subpart of the domain knowledge. Cases are also introduced under the "Domain terms". As explained in 5.2.1 the term case is not used the same way in TrollCreek and CommonKADS. However, it could be possible describe solutions/lessons learned through problem solving knowledge as it is defined in CommonKADS (in section 3.4.4). The "problem solving method" can contain a sequence of actions (for example lessons learned trough cases).

In CommonKADS the domain taxonomy is a part of the domain ontology (as seen in figure 3.13 on page 30). TrollCreek does not have a clear division with several hierarchies. The TrollCreek ontology consists of a non-uniform hierarchy. CommonKADS has larger differences between the domain typology, taxonomy and terms. There is a strict difference between the typology (defines what kind of concepts, what types of domain knowledge one has) and the taxonomy it self (a type hierarchy).

Modelling Methods in CommonKADS

If CommonKADS is chosen as the modelling framework to merge/apply to TrollCreek there are several important features to look at/learn from.

As mentioned in section 3.4.6 CommonKADS has several generic components presented in a library. CommonKADS also provides several modelling methods. These could be the basis for developing the modelling approach in TrollCreek if chosen. It might even be a thought to introduce library also in TrollCreek.

5.2.3 CoE + TrollCreek

Section 3.3 introduces the Components of Expertise (CoE) framework. The framework decomposes the knowledge level description into components. It is an alternative to combine the CoE with TrollCreek.

The Three Types of Knowledge

As mentioned in section 3.3 the CoE framework handles knowledge accordingly to the three perspectives in figure 3.6. Domain knowledge is referred to as models and it is easily identified in the TrollCreek ontology (discussed in section 4.2.6). The same goes for task knowledge. Even the methods are introduced under the "Process" child node of "Entity". They are introduced under the "entity" node in the TrollCreek taxonomy (as explained in section 4.2.6).

CoE makes distinctions between the case models and the domain models (described in section 3.3.4). The domain model enlarges the case model. The domain model is a fixed part of the application, while the application builds up the case model. In TrollCreek the domain and case models are integrated as a parts of the same ontology model. However, as explained in section 5.2.1 the "case" in the CoE case model is not the same as the case definition in TrollCreek. This is two completely different things.

There are three very important diagrams in the CoE framework. It is possible to use them in a modelling approach for TrollCreek. The model dependency diagram (described in section 3.3.4 and 3.3.3) is the link between the task and the model perspective. As described, the model dependency diagram shows the relations between the models and the tasks in addition to how they participate in models. This diagram would come in handy for TrollCreek.

The task diagram shows the decomposition of the tasks into subtasks (as explained in section 3.3.3). In TrollCreek this task-subtask hierarchy possibility is already an integrated part of the TrollCreek ontology. It is possible to make a hierarchy within the ontology of tasks and subtasks.

Event though there were no way of representing explicit method knowledge in the TrollCreek top-ontology, a solution is proposed in section 6.1.2. It is possible to apply the method perspective to TrollCreek. The control diagram depicts the control flow between the tasks in TrollCreek.

It is also possible to use a wide variety of problem solving methods in CoE (as mentioned in section 3.3.6). When choosing the problem solving method in CoE it is important to look at both the pragmatic and conceptual aspect (explained in section 3.3.7).

Chapter 6

Result

This chapter contains the results produced in this thesis. It contains a section on the choices made as a result of the comparison and analysis mentioned in chapter 5.The next sections are the concrete knowledge acquisition modelling method in TrollCreek (section 6.2) and example of the method applied in TrollCreek (section 6.3).

6.1 Choices Made

This section explains the exact choices made based on the comparison and analysis in chapter 5.

6.1.1 Focus in TrollCreek

In order to limit this master thesis to some extent one has chosen to have a look at the knowledge acquisition and modelling in connection with ki-CBR and more pure modelling approach (as shown in figure 6.1). This will utilize both deep and shallow knowledge. Even if reasoning using pure CBR is possible in TrollCreek, this will not be the focus of this thesis.

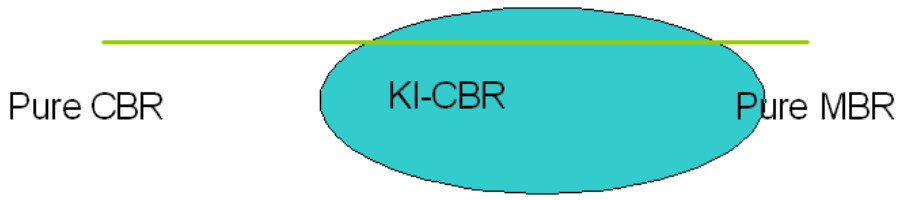

Figure 6.1: Thesis focus when modelling in TrollCreek

6.1.2 The Three Perspectives on Knowledge

Section 3.2 introduces the three types of knowledge. They are illustrated in figure 3.5. Section 4.2.6 describes how task and domain knowledge are structured under the "entity" in the TrollCreek ontology. The methods are introduced under "Process", a child node of entity.

The predefined top-ontology is just a framework. To be a complete model one has to elaborate the top-ontology. The user will need add tasks into a task hierarchy, just as the user expands the domain model. However, the "method node" should be added as an extension of the top-ontology. It is easy to overlook the node in the top-ontology all together. It is hidden as a child node to "process". The problem solving methods are quite generic, and a more detailed decomposition of them should be included in the top-ontology.

Refinement the method node in the top-ontology

The decomposition of the method node does not exist in the top-ontology as it is today. A possible refinement is proposed here. The subclasses of the "method node" are the actual methods divided into categories. There are several ways of dividing the methods into categories. One such division was introduced by Aamodt[Aam01] and discussed in section 3.3.5 (task decomposition, task execution, control methods and mapping methods). This is a part of the foundation of our method decomposition.

With our decomposition, the method node would get decomposed into several several "has-subclass" relations (as shown by screen shot 6.2) at the top level of the decomposition.

The mapping methods will be a subclass of task execution methods (illustrated in figure 6.5). This goes against the division of methods in [Aam01], but follows more the basic principles from the components of expertise framework [Ste91].

However, the different methods will be sub classified into a more detailed description of sub-parts. When decomposing a method into more details, the resulting sub classification will not be a pure subclass of the original method node. The detailed sub classification of the methods will be a part of the method and will have "has submethod" relations. The "has submethod" relation was added to the "Structural Relation" node in the top-ontology in TrollCreek (shown in figure 6.3) and the inverse to the "Structural Relation (inverse)" node. The simplest solution is to use the "New Relation Type" button in the knowledge editor (shown in figure 6.4).

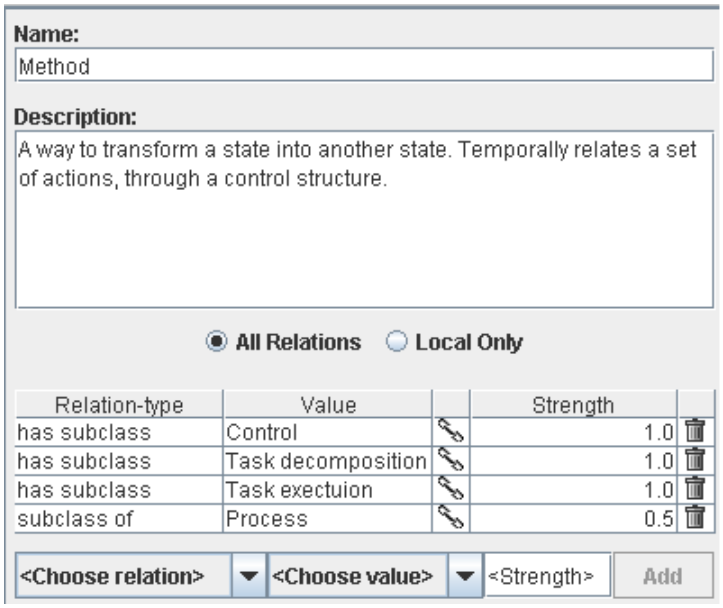

Figure 6.2: Method decomposed into task decomposition, task execution and control methods.

This is illustrated in figure 6.5 by the detailed decomposition of the Propose-and-Revise method.

For more detailed description of modelling, see [BSAB04].

Refining the task node in the top-ontology

The task node in the top-ontology also needs refinement. However, unlike the method node, there is no standard task decomposition. The task hierarchy will be created by the task decomposition part of the KA modelling method (described in section 6.2.1).

However, the tasks in the task-subtask hierarchy will not use the "has subclass relation". It is necessary to add a new type of relations. This will also be a structural relation. Figure 6.6 shows the result.

Structuring the KA Modelling Method

In this thesis the KA modelling method for TrollCreek is structures after the three different perspectives on knowledge (task, model, method) (discussed as a possible solution in section 5.2.3).The recommended problem solving method for i.e. task decomposition modelling will be defined

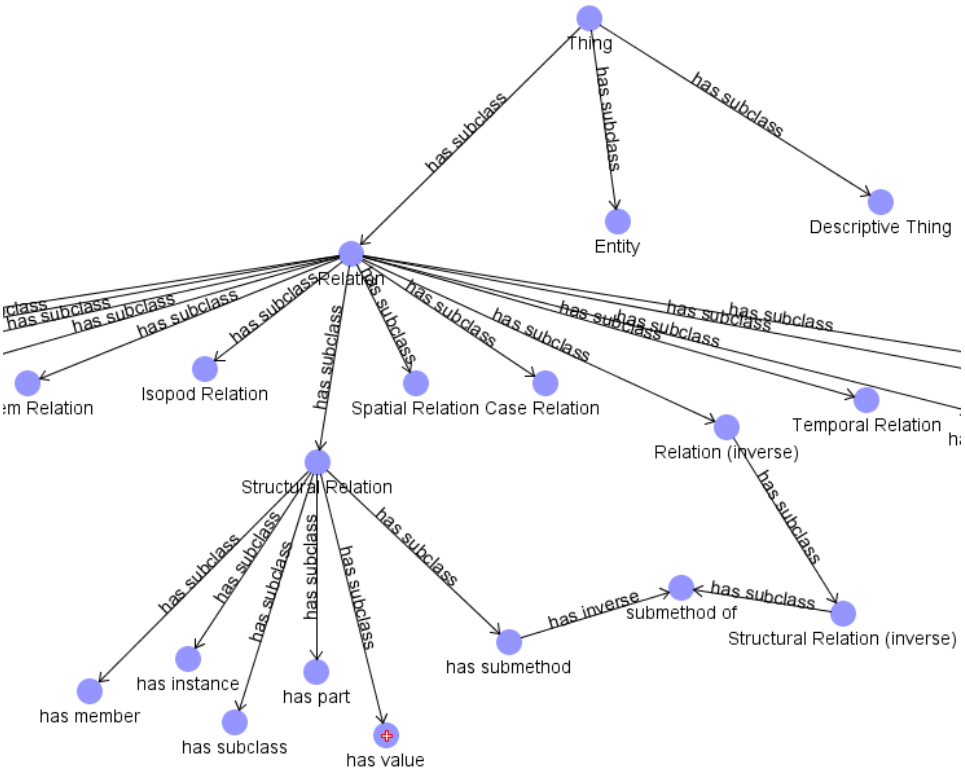

Figure 6.3: The "has submethod" relation when added to the TrollCreek top-ontology

in the next sections. The task execution methods are already included in TrollCreek (used at run time). It will therefore not be that important for this thesis. However, TrollCreek would benefit from a larger degree of freedom choosing all the methods. The proposed method sub classification needs further work (discussed in section 7.3). One option is also to include a library-oriented view into TrollCreek.

When following the thoughts of Aamodt, there are the four types of a method in a CBR system (explained in section 3.3.5). The following sections discuss the choices made.

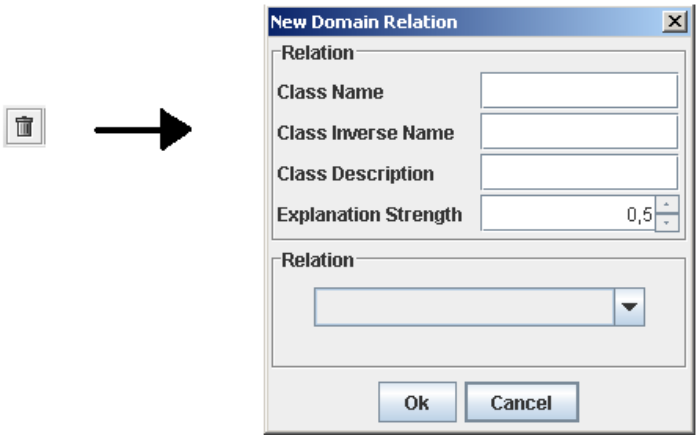

Figure 6.4: The "New Relation Type" button opens the "New Domain Relation window.

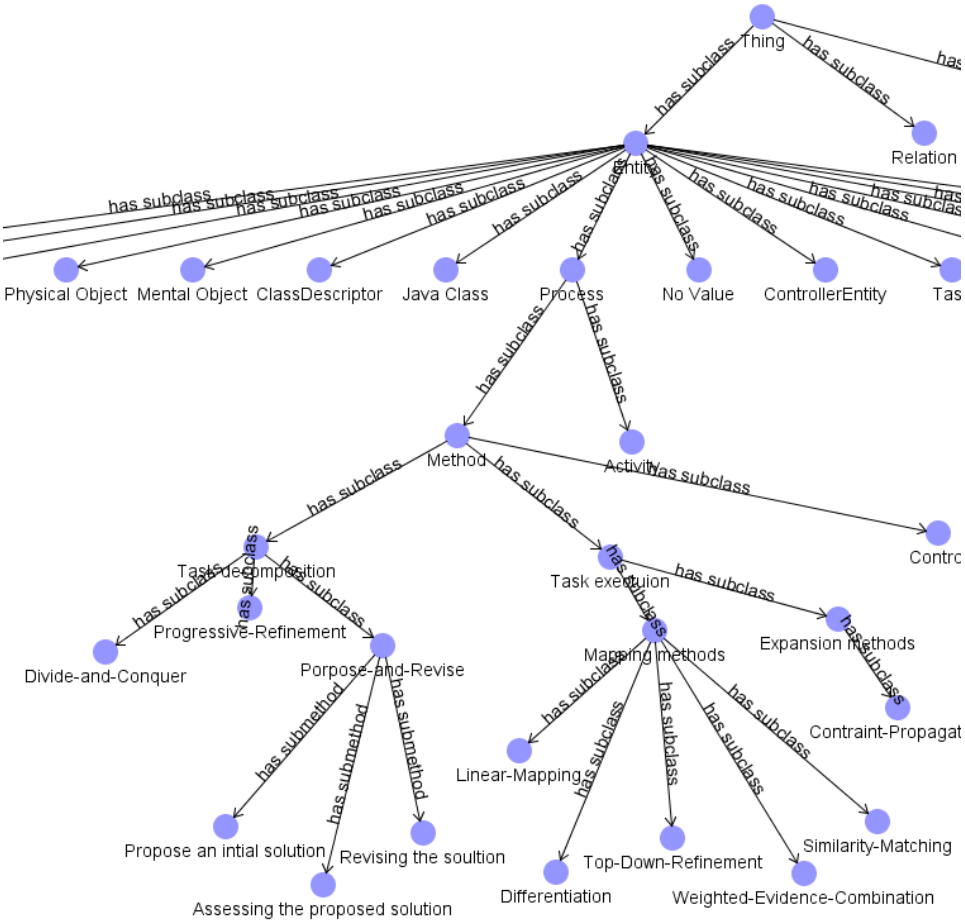

Figure 6.5: The method subclassified in the top-ontology in TrollCreek.

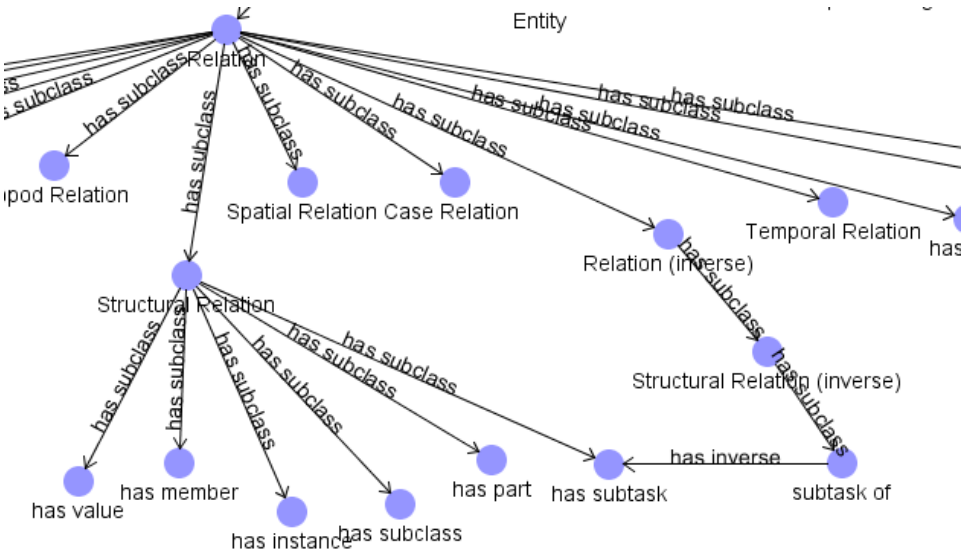

Figure 6.6: The "has subtask" relation placed in the relation hierarchy

Task Decomposition Method

The task decomposition methods (explained in section 5.1.2) are an extremely important part of the KA method. The task decomposition method decides what approach to take when making the task-subtask hierarchy. Being a ki-CBR system the "propose-and-revise" (discussed in section 5.1.2) method seems to be the ideal match. The propose-assess-revise steps are a perfect match to the already existing CBR process (explained in section 4.2.3).

However, it might already seem like a relative of propose-and-revise method is already in use in TrollCreek in the underlying case matching process (at runtime). Since this KA method is a form of KL-modelling, it is not bound by implementation constraints. From a user perspective, the divide-and-conquer method is more intuitive when dividing tasks to sub-tasks. The control flow is simpler and more understandable from a user perspective. It is an important consideration to have the simplicity for the end user in mind. This is users not interested in KA-methods from academic perspectives.

The methods will be presented in detail in the top ontology (as described earlier in 6.1.2 and shown an example of in figure 6.5). For an experienced user, it is possible to choose personal task decomposition method amongst the methods represented in the top-ontology. It is even possible to add new methods (by using the knowledge editor in "build" mode and adding entities when in map view (explained in section 4.2.1).

However, one have chosen to use the divide-and-conquer as task decomposition method described in section 6.3. This is because the method is simple, intuitive and effective for the user. The method has to be a useful tool, not a burden/overhead for the user. There is no need to use a more complicated method than one has to. The user is still able to chose what task decomposition method he or she wants to use.

Task execution method

TrollCreek handles task execution at run time (i.e. through the case-matching process (that includes elements from both "weighted evidence" and "similarity matching")). From the KA modelling perspective, the implementation of runtime execution methods is more or less a black box. The important features are just the input to the box and the output from the box.

However, some features from the task execution methods (explained in section 5.1.3) might come in handy when building the model dependency

diagrams (MDD). This is especially true for the mapping methods. For example, linear mapping is an important feature to be included in the KA modelling method.

There are several important methods in CommonKADS that is useful in TrollCreek. Section 3.4.6 explains some of them briefly. The difference between top down and bottom up is definitive an important asset to the KA method in TrollCreek.

Control methods

Even if the task decomposition process in itself is sequential, the sequence of the subtasks still has to be decided. There might be some interdependencies between the subtasks. When one is modelling the control diagram (as described in more detail in section 6.2.3) the control flow is decided.

Mapping

The model dependence diagram handles the mapping. Model dependency diagrams (MDD) is discussed in section 3.3.4 and 3.3.3. There is no need to define a specific general mapping method in TrollCreek since it already exists in the TrollCreek system. The mapping methods will wary from situation to situation. The different mapping methods are described in the proposed new top-ontology in section 6.1.2 and shown in figure 6.5. However, the mapping from task to model decomposition usually is a quite intuitive process.

However, in for instance in CommonKADS there is a limitation to only mapping tasks to methods in a 1-1 fashion (in the model construction operator). This limits the MDD and reduces the power to express relations. The limitation is mostly an implementation choice, and there is no need to implement that constraint in the TrollCreek KA modelling method. The user is able to associate several method with a task.

SIZZLER

As introduced in section 5.1.1 in table 5.6, the SIZZLER KA tool is a great match for a ki-CBR system. However, SIZZLE and *extrapolate-from-similar-case*(ESC) would be an addition to already existing methods in the case-matching process in TrollCreek.

TrollCreek, being a ki-CBR system is already built up around cases, a case-base, similarity measurements between the cases and a revise process. TrollCreek also models domains where there are a large number of factors to be considered (i.e. the oil drilling domain) and handles processes often not well understood and to determine needed quantities (i.e. torque pressure to avoid a stuck pipe event in the oil-drilling domain). In TrollCreek, it is possible to decide if a solution is incrementally worse or better than another solution during the revise process.

However, it is weakness that the ESC seems to handle only quantitative extrapolations. This might suggest that it would be a great addition to already existing problem solving approaches in TrollCreek. This is to de discussed in further detail in 7.1. The experts already provide the cases, the case indexing knowledge and the extrapolation knowledge in TrollCreek. These elements will allow the sizer to extrapolate from solved to unsolved case.

6.1.3 Competency Questions

The idea of using competency questions was introduced in [NM00] and discussed in section 3.5.4. By making a list of questions, the system should be able to answer helps the user determine the actual scope of the domain. This idea might come in handy when modelling in TrollCreek as well.

6.1.4 The Different Types of Models

As explained in section 3.3, there are several types of models in the components of expertise framework. However, in TrollCreek all domain knowledge is organized in the TrollCreek ontology. Instead of using completely different types of models for the KA modelling (i.e. during making the model dependency diagram), one only uses different segments of the TrollCreek ontology. This also goes for the different case models (the current ki-CBR case is also represented in the TrollCreek ontology).

6.1.5 The Nature of the TrollCreek KA Modelling Method

This KA modelling method for TrollCreek is a supplement to the already existing modelling tutorials (introduced in section 4.2.5). The existing tutorials are more practical and detailed when it comes to actual using

the TrollCreek editor. They are at the level of "point-and-click" instructions to modelling. The KA modelling approach will not be at that level of detail when it comes to the "clicking" aspect to modelling in the TrollCreek editor. However, some practical tips and challenges that one came across while modelling, will be included also in section 6.3.

The nature of modelling in general implies an iterative process. The TrollCreek KA modelling method is no exception. It might be necessary to repeat several steps as one refines the model. For instance, it is easy to get into a situation where one has to add more tasks as the domain model building uncovers new needs. Adding cases to the TrollCreek case-base can also make a change in the domain model inevitable. This is illustrated in section 6.3.

6.1.6 The Unified ki-CBR Case

As shown in section 4.2.6, the top-ontology in TrollCreek today has a caothic way of handling cases. The case structure node is located as a subclass of Representational Structure (shown in figure 4.11) Under the entity node (explained in section 4.2.6) there is a division between the case and the case contents (the actual situation).

This is discussed in more detail in section 7.3.1. However, strong indications from the AIL group here at IDI indicates that this will be fixed in the next version of the ontology. We therefore chose to use the new, improved structure for the example (section 6.3).

6.1.7 CommonKADS vs. CoE

The perhaps most obvious difference between CommonKADS and CoE is that CoE is less formal. It also does not contain many "extra features" like i.e. organizational models. The higher-level models would be great in a broader more higher level system. However, it might seem that the needs of the TrollCreek system are more down to earth and informal.

The strict hierarchical system of domain knowledge in CommonKADS expertise model would impose more structure to the knowledge in TrollCreek. If applied to TrollCreek the structure would most definitely kill the confusion around the taxonomy and ontology terms. However, the existing framework of top ontology in TrollCreek would not handle the structural change to good. In addition, the limitations would impose unnatural limitations to the modelling abilities.

The use of case models in CoE fits TrollCreek perfectly. As a ki-CBR system TrollCreek might use the case model in the problem solving process. In the model construction approach to problem solving the model is considered at each step of the problem solving. This fits perfectly with the CBR problem solving cycle (explained in section 4.2.3).

There is also several features that resembles each other when comparing CoE and CommonKADS. Such a resemblance is the natural result of CommonKADS being a combination of KADS and CoE. However, even if I have chosen to use CoE as the master influence on the knowledge acquisition method in TrollCreek, it is still possible to borrow important features from CommonKADS as well. It is for instance possible to play with the idea of a library when doing further research/development on TrollCreek.

It is possible to repeat the different steps of the process as many times as the modeller would like (explained in section 6.1.5).

6.2 A Knowledge Acquisition and Modelling Method in TrollCreek

The KA method is the initial knowledge modelling process. The goal of the modelling is to develop a conceptual model for later use. It enables communication within the development team and aids the further development and design of the system. KL modelling (explained in section 3.2) takes place between the knowledge level and the symbolic level. The KL-modelling presents the knowledge in an intuitive way. And it is at the symbolic level the KL components get implemented. However, the implementation is not the concern of this thesis. I only utilize already existing features in the TrollCreek knowledge editor.

TrollCreek contains a predefined ontology that is a part of every model produced in TrollCreek (explained in section 4.2.6). It is natural to elaborate on this ontology when starting a model. When using the TrollCreek framework, even for KA, the predefined ontology is an important feature.

An intuitive way of doing the modelling is first to use a top-down approach, before switching to bottom-up when handling the cases (for instance when having a series of time dependent cases that one relates to a "parent" case). When starting the modelling, it is natural to continue the already existing organization in the TrollCreek ontology. As explained in section 4.2.6 there is a division into relation, entity (both representing external things) and descriptive thing (internal).

It would have been a didactic move to present the analysis (task decomposition and model dependency diagrams) and the design (deciding the form of the model and the nature of the method that achieve each task). However, in this setting I have found it more useful to divide the KA method accordingly to the three perspectives of knowledge task, model and method. Within the different perspectives will be divided into identify (initial knowledge analysis), model (a paper based model) and realise in TrollCreek (building it in TrollCreek).

6.2.1 Task

The task handling process is in many ways the backbone of the KL-modelling process. The tasks one wants to accomplish first have to be identified, then decomposed into subtasks (here: using the divide-and-conquer method (as explained in section 6.1.2)). The task decomposition will result in a task-subtask hierarchy (as used in CoE explained in section 3.3.3).

Identify Tasks

When modelling in TrollCreek, one handles at least two different categories of tasks. There is a clear division between the domain acquisition tasks (in the design phase, express fixed knowledge about the domain) and the application tasks (used in CoE to develop case models (a CoE "case" is explained in section 5.2.1), handling the input/output contents of a case model to a running application). In this initial modelling, it is the domain acquisition tasks the KA modelling method has to identify.

These initial tasks are higher-level tasks, the root node of the following task-subtask hierarchy. They do not go into any details. Example of such tasks might be for instance "prevent downtime situations" from the oil-drilling domain, "production planning" from the production domain.

Then it is time to decompose the root tasks into subtasks, resulting in the task-subtask hierarchy. As discussed in section 5.1.2 and section 6.1.2, the divide-and-conquer method is our preferred method for decomposing the tasks in section 6.3. However, due to the nature of the task decomposition in the predefined top-ontology (described in section 6.1.2) the user is able to chose between several possible task decomposition methods for the modelling.

As long as a task is splittable and the task decomposition is feasible, the task decomposition will continue. When a task no longer is decomposable, it is considered to be a solution task. When looking at the overall

task hierarchy, the solution nodes will be the "leaf nodes" (at the bottom of the hierarchy).

The divide-and-conquer method we have chosen to use, divides the root tasks into subtasks. The subtasks will be decomposed until the decomposed tasks are solution tasks. The subtasks will then be ready to be solved recursively. The solutions of the subtasks make up the overall solution to the root node. However, in TrollCreek it is also possible to execute decomposable tasks (in runtime) in the higher levels (discussed in detail in section 7.3.1).

Modelling - Task-Subtask Hierarchy

It is easy to picture the nodes and leaf nodes in a hierarchical tree structure. The task-subtask hierarchy will be made during the decomposition. It will be beneficial to the user to make/draw a model on paper before entering it into the TrollCreek knowledge editor. The overhead is minimal as the structure is already being made through the decomposition.

An example of divide-and-conquer is the division of "production planning" into "acquire operation", "identify machines", "order machines" and "present production flow" [Ste93].

Realise in TrollCreek

In TrollCreek the graphical user interface in the knowledge editor gives us a user friendly way of presenting the task-subtask hierarchy, and it is easy to model the hierarchy. In the already predefined ontology in TrollCreek Task is a subclass of Entity (that in turn is a subclass of Thing). This is shown in figure 6.7.

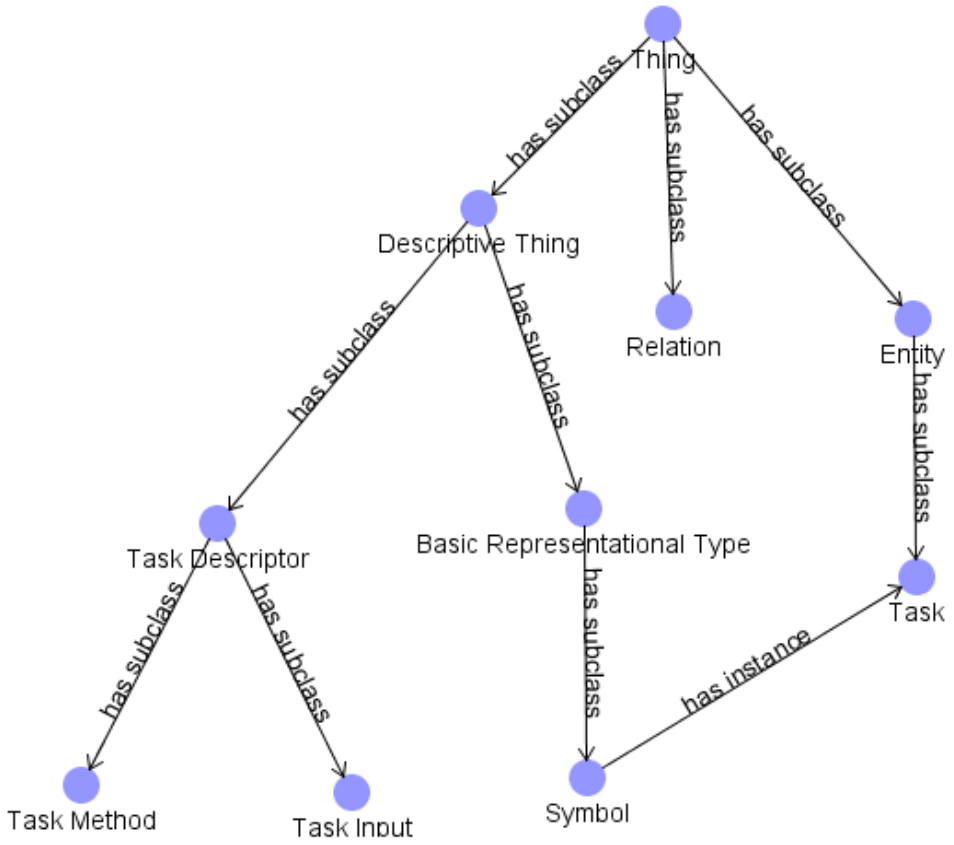

Figure 6.7: Task placed in the overall TrollCreek ontology

Just as tasks are a subclass of entity, a task is an instance of Symbol (just as i.e. environment, situation, parameter, role is).

It is easy to add new tasks by using the tools described in section 4.2. By using the build tool when in map view (explained in section 4.2.1), a new entity is added by just clicking. Then it is possible to open the frame view (explained in section 4.2.2) for the new entity and adding it to the task-subtask hierarchy (by adding relations). However, it is important to remember that a relation in TrollCreek is bidirectional. When a new subtask is added, there will be a "has subtask" relation from the original task to the subtask AND the subtask will (automatically) have a "subtask of" relation back to the original task.

This procedure will be shown in more detail through an example in section 6.3.

6.2.2 Model

A model in the TrollCreek KA modelling method characterises domain specific knowledge and the interdependencies between the domain specific knowledge. As explained in section 3.3.4, there is a clear division between the case model and the domain models in a CoE framework.

Identify Models

A ki-CBR Test Case It is useful to think through at least one test ki-CBR case. The process of identifying the different components of a ki-CBR case (i.e. the "relation-type" and the "value" from figure 4.6 on page 45) uncovers several important terms to include in the domain model and much about the structure of the current situation.

The Current Situation (the CoE Case Model) As mentioned in section 5.2.1 the CoE case model contains the current case, the current situation in runtime (and there are several different types as shown in section 3.3.4). TrollCreek will be working on the current situation case in runtime. The actual ki-CBR cases in the case database will be domain knowledge. The only features one needs to be concerned with is the input and output of the case model, as the case model will be used later in the model dependency diagrams.

The case model in this TrollCreek KA modelling method will be the "snapshot" of the current situation in runtime. For example, the state of the world when running problem solving in real-time.

Competency Questions Another important issue is the scope of the ontology. The scope is heavily dependent upon what the model is going to be used for and how much knowledge is needed about the domain. An important aid when defining the scope is the use of competency questions (discussed in section 6.1.3). By defining a set of questions your system should be able to answer, the user is able to define the scope of the domain model more accurately. In this initial stage, it is important to be critical what is really needed in the ontology. This will save a lot of work in later stages. The use of such questions will be illustrated through the example in section 6.3.

The Actual Domain Model During the previous steps, the user has already identified several important central concepts in the domain.

The domain model includes BOTH the previous experiences (ki-CBR cases) and general domain knowledge. With the exception of a few initial cases (to be discussed later), the user can concentrate on general domain knowledge in this initial part of modelling.

Through the test ki-CBR case, some of the important terms have already been identified. However, it is possible to add more in addition to the ones in the initial ki-CBR case. Traditionally concepts in the domain model are closely connected to objects in the domain. The concepts are even most likely nouns (while the relations are verbs) when describing the domain. It is also crucial to have in mind how the different entities are connected also.

Modelling

Model Dependency Diagram As explained in section 3.3.4 and 3.3.3, the model dependency diagram (MDD) connects the tasks to the models needed in the CoE KL-modelling. In the TrollCreek KA modelling method it is possible to connect several tasks to a model (as discussed in section 6.1.2). It overcomes the imitating 1-1 implementation constraint in CommonKADS.

The MDD illustrates the process of making new models, by showing what model are being used (both case and domain models) and the model construction activities (tasks) that creates a target model from the source models. The different models (see section 3.3.4) going into the MDD will be an excerpt from the ontology in TrollCreek (explained in section 6.1.4).

The MDD need only to exist in a paper version at it is more a aid to clear the thoughts and help the user see how the methods and the tasks are interlaced. A notation used by Steels in [Ste91] draws double framed boxes when it is domain models in the MDD. While the case models are depicted with only a single frame.

An example of the MDD process will be shown in section 6.3.

Paper version of the Ontology In TrollCreek the much discussed top ontology (in section 4.2.6) already exists. It lays down a clear guideline for how the domain knowledge should be represented in the TrollCreek system. Even if Steels divides into types of domain models (explained in section 3.3.4, the TrollCreek ontology incorporates all the different domain models in one network (i.e. descriptions, symbols, solutions and so forth). The ontology it self contains both terms in the

domain and the effects they have on each other (causal relations). It is time saving to use the appropriate relations defined in the already existing top-ontology in TrollCreek (introduced in section 4.2.6) when modelling. However, it is possible to add new relations also. These has to be specially treated when realising the model in TrollCreek.

The domain modelling is first done in a top-down fashion to capture all relevant, general traits with the modelling domain. It will be a continuance of the already existing top-ontology in TrollCreek. This top-down approach will be done when during the initial modelling and preparing the system for use. It will also include adding some initial cases (in order to build a small initial case database).

The ontology is structured as an extended version of the already existing top ontology. The extension follows the same pattern and is connected to the already existing nodes in the top-ontology. However, the modelling task might seem a bit overwhelming. Experience has shown it is easier model when thinking of the different parts of the ontology thorough the same "aspects" used in MDD. It is easier to model a whole domain when for instance starting with the physical aspect, then the states aspect, the causal relations before connecting the initial ki-CBR case.

Realise in TrollCreek

The TrollCreek knowledge editor is a valuable tool when realising the paper ontology in Troll Creek.

Top-down The paper version of the ontology already contains the general traits (both terms and the interconnections between them). To realise the paper based model is just to add the concepts and make the relations between them by using the different views in TrollCreek (for instance by using the map view (see section 4.2.1) and the build tool to add entities and drawing the relations between). By using the correct relations (in section 4.2.6) already in the paper based version, putting the model into TrollCreek does not present a difficult challenge. However, if the user has defined new relations that are not predefined in TrollCreek, the user has to add the relations to the ontology model (example of this is shown in section 6.1.2. By being critical when copying the paper-based version into TrollCreek, it is possible to rethink the design and catch errors at an early stage. It is a great opportunity to add a "safety valve" to the modelling.

Bottom-up When actually running the application, the experiences encountered during runtime will have to be added to the ki-CBR case-base. (There will be a transition from the CoE case model to domain model.) It will also be necessary to add other ki-CBR cases i.e. ki-CBR test cases. These will be added in a bottom-up fashion by using the TrollCreek editor by starting with the most specific ki-CBR cases and grouping them into more general ki-CBR cases. This will be useful when presenting time series of ki-CBR cases [JAS02], [Bak04]. The modeller starts with the leaf nodes, working her way upwards in the hierarchy, until the branch is ready to be connected to the predefined top-ontology. This results in an iteration, as it then becomes necessary to add new elements to the domain model.

A concrete example of modelling will be shown in section 6.3.

6.2.3 Method

The method perspective (described before in section 3.3.5) attaches methods to tasks and describes the control flow in diagrams. The method describes how to the task is organized (task decomposition) and executed (task execution). At the knowledge level, a method organizes and executes the model construction activities [Ste91].

Identify Method Algorithms

As introduced in section 6.1.2, divide-and-conquer method has been chosen for task decomposition in this example. The task runtime execution is already a part of the existing TrollCreek framework, and the control and mapping is handled otherwise (choices made in section 6.1.2). To some extent, these decisions in cooperation with the MDD and control flow diagram govern the control flow.

However, in general it is possible to use also other methods than the ones predefined by the ontology. Then the method has to be added into the TrollCreek architecture. The user is also free to use whatever methods she or he likes for model decomposition. The runtime executions are included in the TrollCreek system and not so easy to change.

One special case of methods is the causal relations within the domain model. These govern the causal relations between the different concepts. To some extent, these might function as control methods in the domain. However, this is more a part of the implementation of the domain knowledge than the control flow of the system.

Method - Control Flow Diagram

The control flow diagram is a state automata, with a range states (final or not) and transitions (the control flow) as described in section 3.3.5. The conditions makes up the control flow and regulate when the transitions between states takes place. When creating the diagram the different tasks are the states. The nodes in i.e. the task decomposition hierarchy will be independently executed and the solution will recursively make up the total solution of the root task (when having used divide-and-conquer for task decomposition). Due to the interwindings in the tasks, they will have to be executed in a special sequence. This control flow is depicted in the control flow diagram.

When it comes to the execution in runtime, the CBR process (sequential but cyclic as described in section 5.1.2) is already an integrated part of TrollCreek. This is done in the implementation and will not be of concern in this KA-modelling.

The control flow diagram is an important tool, even if it is just paper based. By checking the subtask sequences, the users might check the correctness of the task decomposition as well. It becomes another safety valve for the modeller to check that everything works properly and that the causal relations between the concepts are correct.

Realise in TrollCreek

If any new methods were identified during the "identify" stage they will have to be added to the ontology. This is done by just adding new nodes to the method node in the ontology (a possible version described in section 6.1.2).

The control flow diagram will be realised in TrollCreek by using the causal relations. This is done by altering the relations in the domain model between the concepts (for instance adding i.e. "causes" relations).

6.2.4 Executing the TrollCreek KA modelling Method

By executing the top-level KA tasks, the subtasks will be recursively addressed. This leads to the system acquiring all the models. In addition, by executing the tasks the system actually changes the contents of the target models.

Using the TrollCreek system would include adding new experiences/ki-CBR cases and case matching. The ki-CBR cases will be added in a

bottom-up fashion in runtime. The case matching is a way of testing if the method really works. It is also possible to revise the models previously made.

6.3 Using The TrollCreek KA Modelling Method

This section contains a practical example on how to model in TrollCreek using the KA modelling method layout introduced in section 6.2. The illustration of the TrollCreek KA modelling method is easies done through an example.

However, it is important that the example is simple enough. The goal of this example is to illustrate the method, not to model a complex domain to perfection. The nature of the example also has to be so that the reader is not caught up in details.

This example follows the modelling of the electric appliances domain. Imagine being unfamiliar with the wide range of electrical appliances that surrounds us in everyday life. If one had no prior knowledge on how to operate such mysterious electrical appliances, one might need a guide. The modelling task is then to give an example of such a modelling process, creating a "Electrical Appliances for dummies" model.

As mentioned earlier, the goal of this example is not to make a complete guide of every electrical appliance in the world. The guide starts small with introducing the newbie to a water boiler, before expanding the model with the big and scary video recorder (VCR).

It is possible to expand the domain even further later.

6.3.1 Task

The task identification and modelling follow the framework from section 6.2.1.

Identify and model

It is important to have in mind the expansion possibilities when defining what the root task is. A possible expansion would be to add for instance instructions for fixing the same mysterious electric appliances. It is also possible to add more appliances (for both fixing and operating). The later point is illustrated through a "operate VCR" task and the "modify electric appliances". It illustrates the AND/OR nature of

the task tree. It is possible to operate a water boiler without operating a VCR at the same time.

The actual root task for this limited example is the "operate electric appliances" node. As one divided the tasks into subtasks, the hierarchical model emerged. Figure 6.8 shows the paper version of the decomposition.

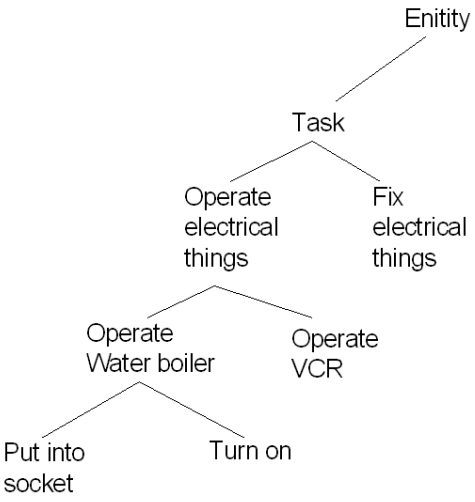

Figure 6.8: The paper version of the task hierarchy

Realise in TrollCreek

To realise the hierarchy (in figure 6.11) one added nodes to the already predefined top-ontology in TrollCreek by using the build tool in map view. When choosing the build tool one is able to add new entities to the model by just clicking in the map view (explained in section 4.2.1). It is also possible to easily make new relations between the entities by dragging lines between them when having selected "Building relation" in the window shown in figure 6.9 and chosen what kind of relation to make.

If an error occurs during the "clicking-and-making-entities-frenzy", it is easy to delete an entity. When using the "select tool", just right click and choose "delete entity" as shown in figure 6.10.

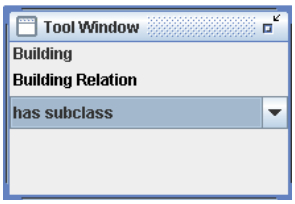

Figure 6.9: When the build tool is selected, this window appears, enabling us to make both new entities and new relations.

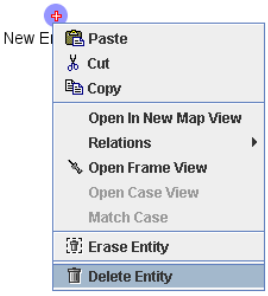

Figure 6.10: Deleting an entity by using the select tool

Figure 6.11 shows the result of entering the paper-based model into TrollCreek.

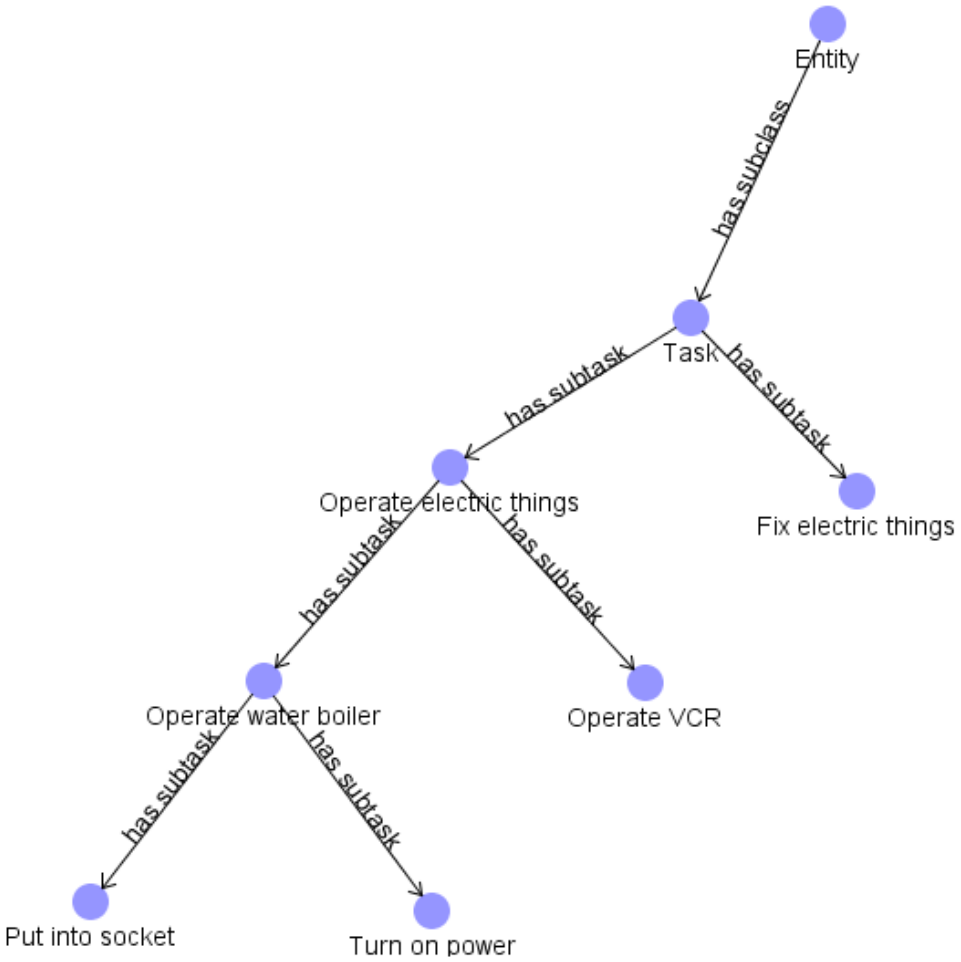

Figure 6.11: The task hierarchy in TrollCreek

6.3.2 Model

The electric appliances example then goes on to the next knowledge perspective, the model perspective introduced in section 6.2.2.

Identify Models

Just as with tasks, the first step is to identify the different models.

A ki-CBR Test Case To initialise the process a first electric appliance ki-CBR test case is created. (NB: notice the difference between the term

"ki-CBR case" and the term "case" (CoE). This is to avoid the confusion described in section 5.2.1).

Case name	Thing1		
Case type	Appliance case		
Status	*Solved*		
Relation-type	**Value**	**Importance**	**Predictive strength**
instance-of	water boiler	characteristic	indicative
has-state	not working	characteristic	indicative
has-state	socket plugged in	characteristic	indicative
has-solution	power turned off	characteristic	indicative

Table 6.1: The initial appliance ki-CBR test case

The Current Situation (the CoE Case Model) The case model in this scenario is the operation of an electric appliance. As seen from the initial case in table 6.1, the different input parameters will be the states (turned on, plugged in and so forth) of the electric appliance. The output from the case model would be if the thing is working or not.

Competency Questions Competency questions help limiting the scope of the model. In this scenario, the system only has to answer a couple of questions:

- How does the user operate the electric appliance?
- When an electric appliance is not working, what has been done wrong?

The limited scope of this model is due to the nature of the example, as mentioned in the start of this section. The example is just to illustrate the KA modelling method. It is not supposed to incorporate the handbooks for all the electric appliances in the world.

The Actual Domain Model As explained the domain model contains objects from the domain. The case in table 6.1 already introduces some important features in the domain. The findings in the initial case shows us the need for the values "water boiler" (instance of something), "not working", "plugged into socket" (both states) and "not turned on" (both a state and solution). The initial case also uses the relations "instance-of", "has-state" and "has-solution".

The domain model will be worked on incrementally as more findings get revealed and added.

Modelling

The modelling will also this time be done on paper. It is easy to scribble something and change ones mind when only on paper yet. The limit becomes much more higher the instance the modelling gets transferred to more time consuming medium (like the modelling in TrollCreek).

Model Dependency Diagram The model dependency diagram (MDD) connects tasks to models. And here it is possible to make as many as one wants. For the Operate Electric Appliances (OEA) domain we start with an MDD that describes how the models are connected in order to create a descriptive CoE case model. The important trick to doing this is to add a section of the domain model in stead of adding separate types of domain models.

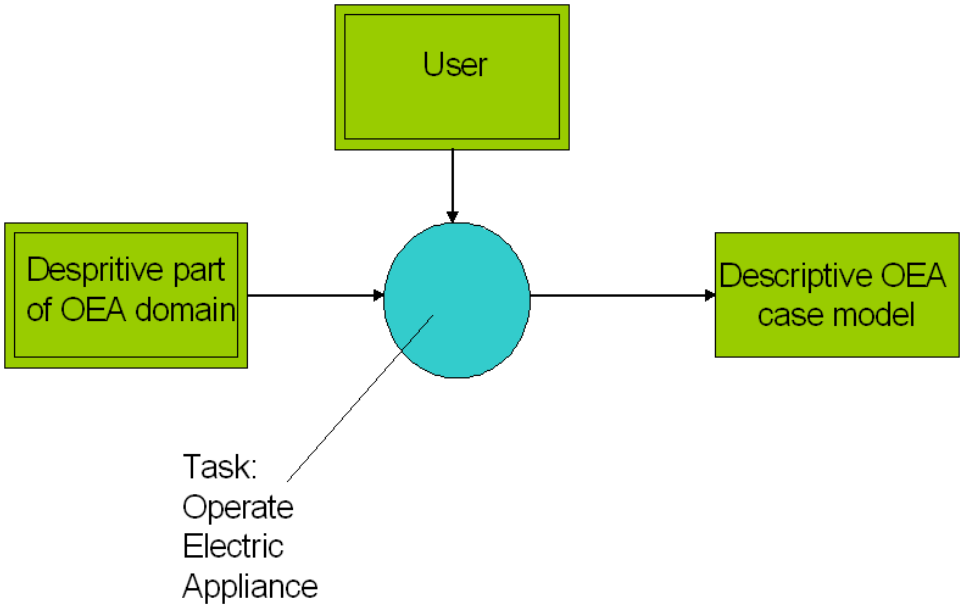

Figure 6.12: The MDD for creating an OEA case model

The first MDD model (shown in figure 6.12) was directly connected to the task tree created earlier (connecting operate electric appliance task to the model activity operator.

However, the next MDD (figure 6.13) shows more TrollCreek internal functions. This is just so that the modeller has thought through the process in TrollCreek as well. The modeller is aware of the states and causal relationships being used internal in TrollCreek. By connecting both the reasoning and the propose solution task to the model construc-

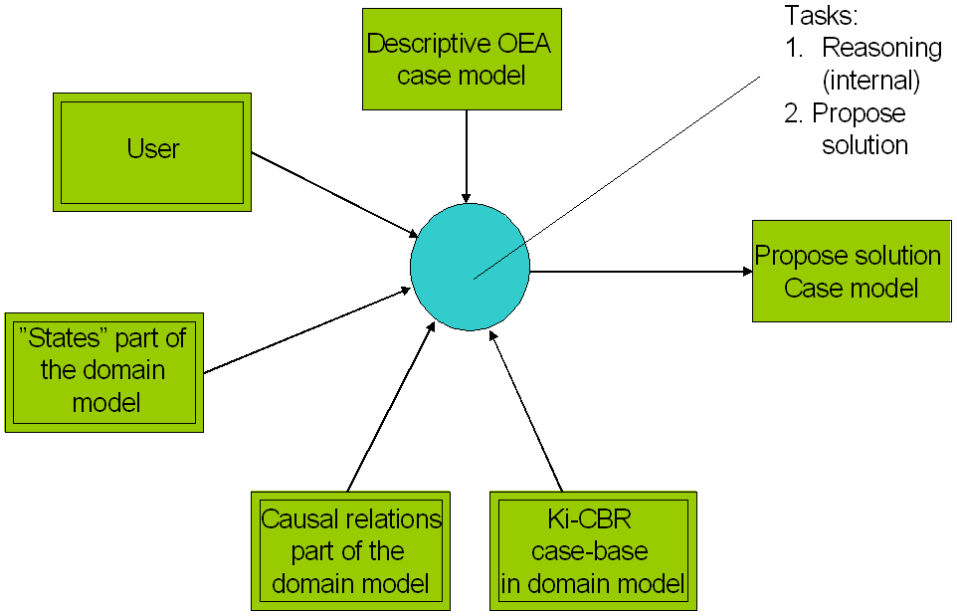

Figure 6.13: The MDD for poropose a solution

tion activity, this example show how it is possible to connect more than one task.

Notice the use of single and double boxes as described in section 6.2.2.

Paper version of the Ontology The previous steps uncovered the need for several instances (both states and relations). Now is the time to put them into system. As mentioned in section 6.2.2, one way of starting the modelling is to look at the different aspects of the domain uncovered in the used views in the MDDs.

This example first modelled the physical aspects of the electrical appliances, then the statuses and the case. The causal relations were added at the end. The result is shown in figure 6.14.

The arrows with no name in figure 6.14 is "has-subclass" relations and the "h-i" relations are "has-instance" relations. This paper version of the ontology will inevitable be a bit chaotic. However, the impression gets better as the ontology gets realised in TrollCreek (next section).

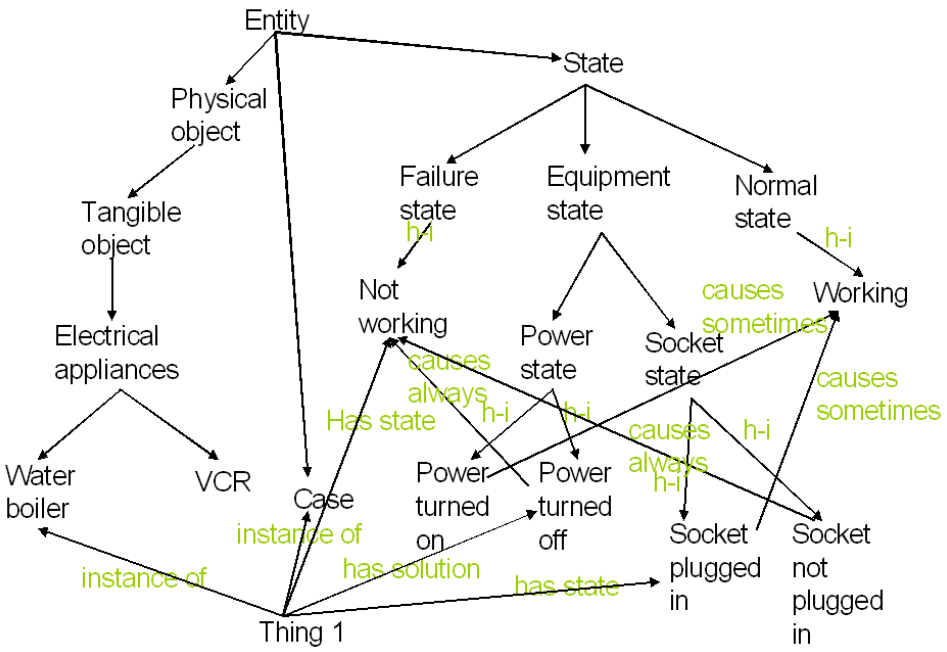

Figure 6.14: The paperversion of the ontology

Realise in TrollCreek

Then it is time to realise the ontology in TrollCreek. As mentioned in section 6.2.2 it is important to use this step as a safety valve. This is achieved for instance by being critical when copying the paper model into TrollCreek. This is an unique possibility to notice errors in a early stage of the modelling. To correct it in the paper model is not as time consuming as having to correct an error in the finished TrollCreek version of the ontology.

Top-down The first top down modelling is done by adding the entities and relations previously identified. This is done by expanding the already existing predefined top-ontology. It is easier to open a new view, adding the entities from the entity list ("Thing" as the ultimate root node) and expanding the ontology from there. It was also useful to include only the nodes I actually used in the view. This simplified the view and made the model more user-friendly. The result is shown in the following figures.

As explained in section 6.1.6, the example uses the next version of the ontology with only one case node. The examples does not use the "situation node".

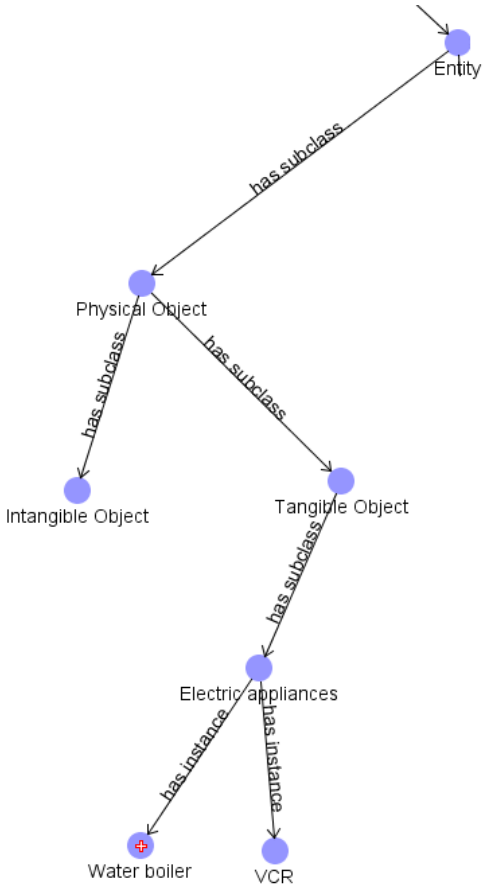

Figure 6.15: The physical aspect of the OEA model

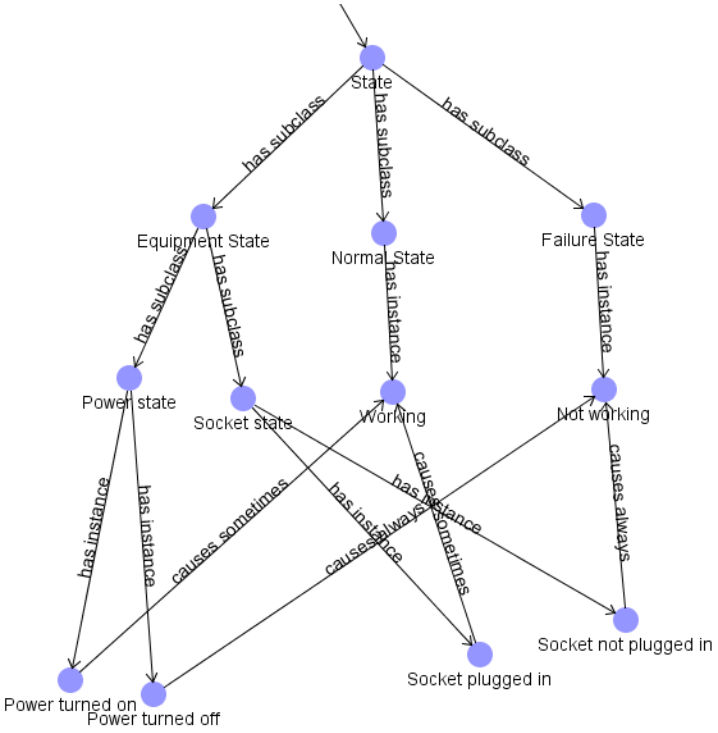

Figure 6.16: Realising the state aspect (with some causal relations)

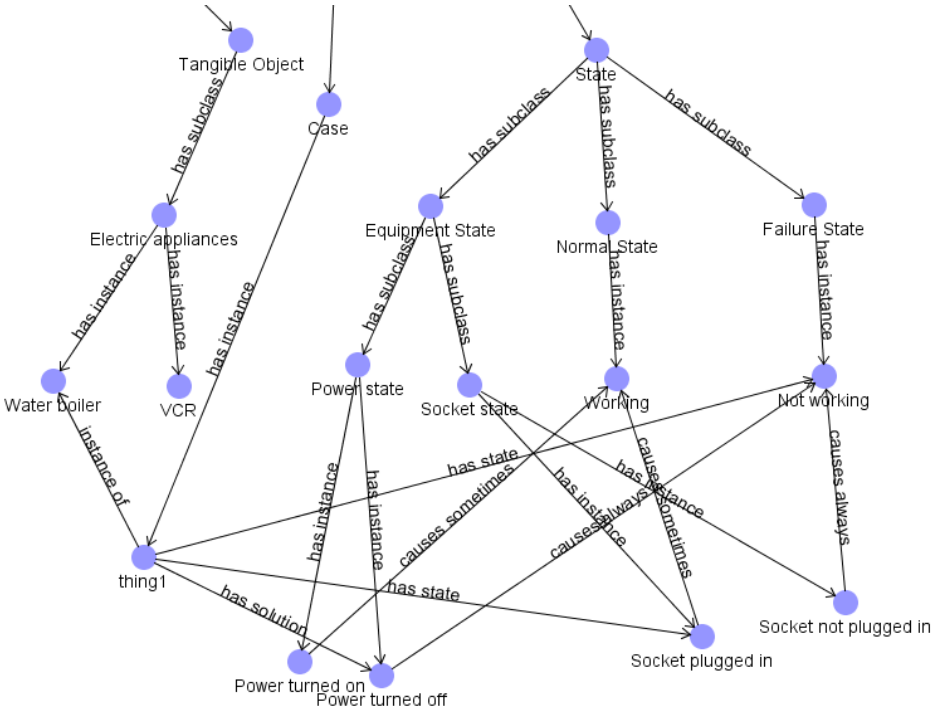

Figure 6.17: The domain model realised in TrollCreek (with a intial ki-CBR testcase)

Bottom-up As described in section 6.2.2 when adding new cases a bottom up modelling takes place. For instance, if one adds the case in table 6.2, there is a need to add a "radio" entity to the ontology (in addition to the case).

Case name	Thing2		
Case type	Appliance case		
Status	*Solved*		
Relation-type	**Value**	**Importance**	**Predictive strength**
instance-of	radio	characteristic	indicative
has-state	not working	characteristic	indicative
has-state	power turned off	characteristic	indicative
has-solution	socket not plugged in	characteristic	indicative

Table 6.2: Additional appliance ki-CBR test case

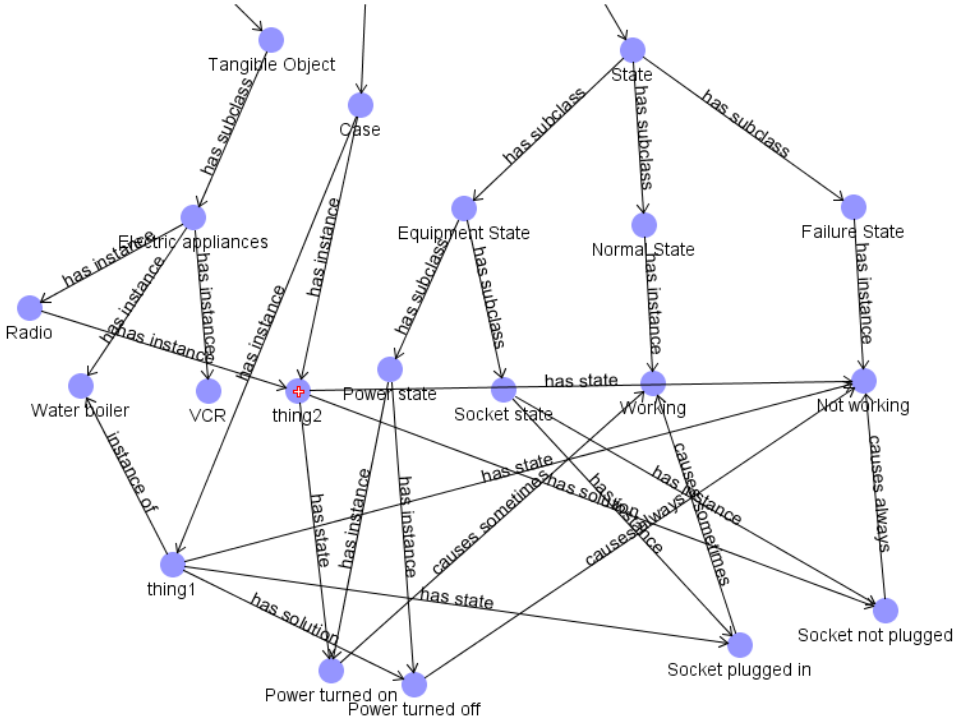

Figure 6.18: The domain model after adding the case thing2

Figure 6.18 shows the domain model after the radio case is added.

6.3.3 Method

This section follows the pattern introduced in section 6.2.3.

Identify method algorithms

The first step in the method perspective is to identify the methods used. Since we have consequently used divide-and-conquer during this example there have not been any new methods introduced.

The casual relationships have already been introduced in the domain model, so there are no new ones introduced at this point.

Modelling the control flow diagram

The control flow diagram for the operate water boiler is quite simple. It contains a start state, a "put into socket" state, a "turn power on" state and a success state ("working") and a fail state ("not working"). Figure 6.19 illustrated the control flow diagram.

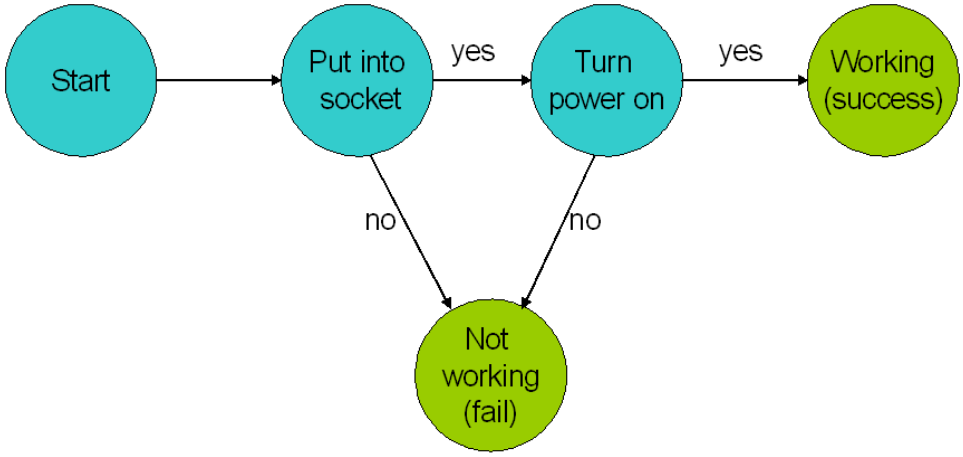

Figure 6.19: The control flow diagram in this example

The "working" (success) and "not working" are final states.

Realise in TrollCreek

There is no new methods added to the top-ontology in TrollCreek. Therefore, there are no need for making extra entities.

As seen by the control flow diagram (figure 6.19) and the domain model with causal relations (in figure 6.18), the control flow is been taken care off by the relations to the different states. For instance the "power turned on" has a "causes sometimes" relation to "working". This is due to the fact that it also need the "socket plugged in" state in order to work

(as shown in figure 6.19).However, the "power turned off" has a "always causes" realtion to "not working" state (failure).

6.3.4 Executing the Model

To test the model made, one should add an additional unsolved case and do a case matching to check that the model works as it is supposed to and is able to answer the competency questions. However, due to the choice in section 6.1.6 this is not possible with in this example. The case and situation has to be unified (as explained in section 7.3.1).

An example of a case that could have been added is shown in table 6.3.

Case name	Thing3		
Case type	Appliance case		
Status	*Unsolved*		
Relation-type	**Value**	**Importance**	**Predictive strength**
instance-of	water boiler	characteristic	indicative
has-state	not working	characteristic	indicative
has-state	power turned on	characteristic	indicative
has-solution	socket plugged in	characteristic	indicative

Table 6.3: A possible additional ki-CBR test case.

The reasoning would be done in runtime in the TrollCreek system.

Chapter 7

Discussion, Evaluation, Further Work and Conclusion

7.1 Discussion

This thesis presented a possible KA modelling method in TrollCreek. However, there are always viable alternatives to the choices made in this method. The KA modelling method has to be adapted to the anticipated extensions and the application one has in mind. The development of the ontology will also be an iterative process.

This thesis presents a method that is very focused on the TrollCreek system. A more general method would also fit other ki-CBR systems. The method would only need to be lifted to a higher level of abstraction. If the method contained fewer details on how to model in the tool, the method would be more general. It would have been possible to assume that the user already is familiar with the tool at hand.

However, due to the TrollCreek connection to the AIL group here at IDI, it was natural to use the TrollCreek system. By being able to show the reader a concrete example of using the method to model (section 6.3), the method described in this thesis is accessible to a larger group of users. It is possible for users that know nothing about Newell's knowledge level to use the method for modelling.

The TrollCreek system is continually evolving and with it some parts of the KA modelling method has to change. This is specially the "realise in TrollCreek" parts of the method. As TrollCreek evolves, this method

will also need to evolve. The method will never be completely finished. There always has to bee room for improvement.

7.2 Evaluation

There are different ways of evaluating the method. The first approach is to look at the predefined goals and evaluation from them. The second approach is to evaluate according to a more formal framework. The two approaches are discussed in the following sections.

7.2.1 Evaluation According to goals

The mail goal in this thesis was to suggest a knowledge-acquisition modelling method for TrollCreek (from section 1.2. We have based this method upon the study of state of the art method in the knowledge-acquisition and modelling research community. The related research was presented in chapter 3 and included a description of Newell's knowledge level (section 3.1), knowledge level modelling (in section 3.2) before presenting knowledge model frameworks like Components of Expertise (in section 3.3), CommonKADS (in section 3.4) and Proégé (in section 3.5). Then we introduced the ki-CBR system TrollCreek in chapter 4, before comparing different problem solving methods and analysing the different frameworks combined with TrollCreek (possible solutions) in chapter 5. The choices made (presented in section 6.1), the resulting method (presented in section 6.2) and an example of use (in section 6.3) are presented in chapter 6 "Result".

This disposition of the thesis follows the guidelines presented in section 2.0.1 and presents a solution to the main goal from section 1.2. This process and research approach has worked extremely well for this type of thesis. This has been a natural way of working. Since the field of knowledge acquisition was relatively unknown material when one started this thesis, chapter 3 was an important ingredient to understanding the field. Moreover, it proved to be a great difference between just reading an article and actually understanding the article well enough to be able to write about the subject.

7.2.2 Evaluation According to Cohen

Besides evaluating the work done according to the goals set and the research approach chosen, it is important to view the method in the

light of a more standardised approach. A more standardised evaluation aids the later users in how the research should proceed, and makes the results replicable. In AI research, in general there has been a vague methodology. A formal evaluation was not the standard practice until Cohen introduced his framework in [CH88], it was fare more common to build a system and relying on the code being informative.

However, the Cohen's evaluation framework was introduced already in section 2.1 and can be used to give a more formal evaluation of the method proposed in section 6.2.

How is the method an improvement over existing technologies?

Before this KA modelling method for TrollCreek, there existed no formal approach to KA modelling in TrollCreek. The modelling approaches that existed were based on an ad-hoc solution without the KA perspective.

Does it account for more situations (input)? The new KA modelling method has more or less the same input to TrollCreek as the existing approaches. However, the new paper based models (i.e. the model dependency diagram and the control flow diagram) depicts new aspects of the modelling. The user has to think through the same information in new ways. So, even if the actual input is more or less the same, the way it is structured is different from existing approaches.

Does it produce a wider variety of desired behaviours (output)? The output from the TrollCreek system will be the same. However, the additional paper based models (for instance model dependency diagrams, control flow diagram) is also output form the method. The paper based version act as a safety valve and saves the modeller time. It also depicts new sides of the system in new ways.

Does it hold more promise for further development? The method opens up for the possibility of adding new KA features in TrollCreek in the further development. This will be discussed in section 7.3.1.

Does a recognized metric exist for evaluating the performance of your method?

There is no pure performance metric for i.e. the execution of the methods. The method is concerned with modelling. The performance of the modeller will be heavily dependent upon the modeller's skills, the modellers experience and the complexity of the modelling domain. However, the method has a clear structure that follows the natural cognitively way of thinking. The method is also normative when it comes to modelling in ki-CBR domains more in general.

Does it rely on other methods? (Does it require input in a particular form or pre-processed input? Does it require access to a certain type of knowledge base or routines?)

The method requires the user to have a mental knowledge base about the domain. It is this knowledge that gets modelled through the KA modelling method for TrollCreek. The user has to know the tasks in the domain, the entities, the relations and the control flow between them.

What is the underlying assumptions?

The underlying choices for the KA modelling method in TrollCreek are presented in section 6.1. To some degree, these are the underlying assumptions for the method. For instance, the method assumes that the TrollCreek top-ontology is being used (a fixed version with "case" as one node). The method also assumes the reasoning to be a TrollCreek internal process. All the underlying assumptions are closely connected to the presented choices in section 6.1.

What is the scope of the method?

The scope of the method is to handle the KA modelling from the stage of identifying the components, modelling them on paper, before actually doing the modelling in TrollCreek. The method handles the three knowledge perspectives (task, model and method).

How extendible is it? Will it easily scale up to a larger knowledge base? As long as the domain is possible to model in TrollCreek, the method is able to handle the modelling. For extremely complicated domains with a large number of tasks and interrelated entities, the

modelling would take some more time. However, the safety valves in the method would prevent errors and save time for the modeller in the larger perspective.

Does it exactly address the task? Portions of the task? A class of tasks? The method addresses the predefined scope. The tasks in the method are closely connected to KA modelling approaches. Besides doing the modelling, there were no specific predefined tasks.

Could it or parts of it be applied to other problems? The method is domain independent. It is extremely possible to apply it to other, more complex domains than the one illustrated in section 6.3. It might also be possible to use the method for other KA modelling tools to some extent. The method has to be individualised for each tool, as it contains quite specific "how to model" in the tool descriptions.

Does it transfer to complicated problems (perhaps knowledge-intensive or more or less constrained or with complex interactions)? The method is made for knowledge-intensive case-based reasoning. This already is "complicated" problems. It handles interrelations and interactions within the domain during the modelling.

When it cannot provide a good solution, does it do nothing or does it provide bad solutions or does it provide the best solution given the available resources?

If the user is not able to provide i.e. the model dependency diagram, it is still viable to make the model in TrollCreek. Some of the KA perspective gets lost together with some of the understanding and safety valve. However, the user might still be able to model an adequate solution in TrollCreek. If the user is not able to do the modelling in TrollCreek, the user would still have the paper versions of the modelling. These can be used later and produce an excellent solution later as well.

How well is the method understood?

The question can be decomposed into four sub questions:

Why does it work? It works because it takes into account BOTH the KA aspect and the practical part of modelling in TrollCreek. The method assures that the KA perspective is being taken care of in the modelling.

Under what circumstances, will it not work? The method will not work if the modeller has wrong knowledge about the domain or lacks important knowledge about the entities and the interrelations. The method will not automatically work for other ki-CBR KA modelling tools than TrollCreek.

Are the limitations of the method inherent or simply not yet addressed? The limitations of the method is inherent as they lye in the nature of the task. The method is a knowledge acquisition modelling method for TrollCreek. If the user does not have enough, correct information to model in TrollCreek, this becomes a limitation for the system. If the user uses another system for modelling than TrollCreek, this is also a limitation because the method is developed for the TrollCreek system.

Have the design decisions been justified? The design decisions is based on a study of related research (in chapter 3) and features of the TrollCreek system (in chapter 4). The options are discussed in chapter 5 and the choices are concretely justified in section 6.1.

What is the relationship between the problem and the method? Why does it work for this task?

There is a close relationship between the problem and the method. The method is specially crafted for the ki-CBR system TrollCreek and the problem was to create a method for KA modelling for knowledge-intensive CBR. The method works for this task as it considers the KA perspective in addition to the "usual" modelling considerations.

7.3 Further Work

There are several point uncovered thought this thesis that could be improved upon. However, due to the limited period a master thesis is, these ideas might be continued in further research. It is natural to

divide the further research into two categories. There is further development/research on the TrollCreek tool, and there is research on the TrollCreek KA method. These will be addressed in the next subsections.

7.3.1 TrollCreek

When looking at TrollCreek from the knowledge acquisition modelling perspective, there are several issues to be dealt with. The handling of method knowledge was perhaps the most severe. In section 6.1.2 I proposed a possible solution. However, later development of the TrollCreek top-ontology has to incorporate the changes and elaborate upon the methods (figure 6.5 illustrates the detailed decomposition of the Propose-and-Revise method).

The end user will get more control of the problem-solving methods. Naturally, different methods fit different types of domains. However, there is a trade-off between user freedom and the risk of malfunction. The user has to know what she is doing when modifying the methods to fit a domain. If later extensions include also modifies versions of the task execution methods, this is even more important. The task execution methods have directly impact on the runtime behaviour of TrollCreek.

An idea much introduced through the CommonKADS framework is the thought of generic components in a library. This thought can be implemented into TrollCreek to some degree. The user will save time by using a library of general components. The future developers of TrollCreek should consider the library idea thoroughly.

I mentioned the SIZZLE problem solving method in section 5.1.1 and in section 6.1.2. The method uses quantitative extrapolations. In TrollCreek today, the matching at attribute level has to be identical (described in section 4.2.4). The symbolic matching mechanism returns only 0 or 1. In the later explanation process, there is partial matching. However, SIZZLE (or the extrapolate from similar case method) could be a great addition in the first initial matching process. It should at least be considered when planning new extensions to TrollCreek.

As mentioned in section 4.2.6 and section 6.1.6, the current version of the TrollCreek top-ontology the cases and its case and its contents is divided into two. This will be integrated into one in the next version of the TrollCreek top-ontology. To separate between the actual contents (the situation) that define what actually happens in real life and the framework it is put into, disturbs the discrimination between internal and external things.

It is quite unclear how TrollCreek actually handles the task execution from a KA modelling point of view (section 6.2.1). In the next version of TrollCreek, this point needs to be made clearer. If TrollCreek executes decomposable tasks (in runtime) in the higher levels, the system has a strategy for handling these tasks. Intuitively, the system has to be able to move down the task hierarchy and finally executing the leaf nodes.

In more long-term plans, it is worth looking at the possibility of adding model dependency diagrams and control flow diagrams. They would then have even larger impact on the modelling. Instead of being paper based, they could play an even more important role modelled in the system.

7.3.2 KA Method for TrollCreek

There are several possibilities for future work on the KA method as well. One course of action is to open up for a more organized reuse of ontology in TrollCreek. It could be quite simple to add a task for ontology reuse when developing the model (as described in section 6.2.2).

Another important feature is how the method has to evolve in parallel with the TrollCreek system. Changes in TrollCreek will also invoke changes in the KA method for TrollCreek, as the "realise in TrollCreek" part of the method will change.

One extremely important factor is the actual user. The user has to feel that this approach is an improvement. If the user feels that the KA method becomes just "red tape", the KA method has failed miserable. The users time is valuable, and the overhead has to be minimal. This is difficult to say much about before the method has been put into real use. However, by using the method (as shown in section 6.3) has at least checked that the method actually works and aids the KA. One solution to the uncertainty of the KA method is to make a revised edition after i.e. 6 months of active use.

7.4 Conclusion

This thesis contains a study of state of the art knowledge acquisition modelling principles and methods for modelling general domain knowledge. This includes Newell's knowledge level, knowledge level modelling, Components of Expertise, CommonKADS and the Protégé meta tool. The thesis also includes a short introduction to the knowledge-intensive case-based reasoning system TrollCreek. Based on this back-

ground knowledge, one did analysis and comparison of different possible solutions. Then, after justifying the choices made, a knowledge acquisition method for TrollCreek was created. The method was illustrated through an example ("operate electrical appliances").

The concrete result of this thesis is the method presented in chapter 6. The method gives a user the means to include the knowledge acquisition perspective into modelling in the TrollCreek system. The method handles the three different perspectives to knowledge (task, modelling and method) in three phases of the modelling (identify, paper models and realising in TrollCreek). Since this is not a traditional implementation thesis, it is difficult to test i.e. the runtime performance of the method. The user is human and the method is more a cognitive method for going about the modelling in addition to realising it in the tool TrollCreek.

The testing of the method has been done through the example in section 6.3 and through extensive evaluation in section 7.2. The example uses the "operate electric appliances" to illustrate the different aspects of the TrollCreek KA modelling method. The evaluation evaluates the method in two different ways. The first approach is through checking that every aspect of the thesis goal and follows the research approach introduces in chapter 2. The second aspect to the evaluation is a evaluation using the framework introduced by Cohen [CH88] and discussed in section 2.1. This is a formal framework for evaluating methods in artificial intelligence research. This gives us a way of looking at the results form in a structured and formal way.

This thesis has introduced the knowledge acquisition perspective to the ki-CBR system TrollCreek. It is a step towards a more conscious use of the knowledge acquisition perspectives in the process of knowledge-intensive case-based reasoning. The focus on the different aspects of knowledge also uncovered several important features about the TrollCreek system it self and how it treats the domain knowledge (discussed in section 7.3). This was especially connected to the organisation of the predefined top-ontology and the representation of the three knowledge perspectives.

The field of knowledge acquisition has lain dormant in the AIL group at IDI. Thus, in many ways this thesis opened up the field again. Modelling in the TrollCreek system has been done a lot. The innovation continued by this thesis is the combination between the knowledge acquisition and the knowledge-intensive CBR. By proposing a knowledge acquisition method for TrollCreek, this thesis gives the knowledge acquisition its renaissance at IDI and adds a new perspective to modelling in TrollCreek.

Bibliography

[Aam91] Agnar Aamodt. A knowledge-intensive, integrated approach to problem solving and sustained learning, 1991. Ph.D (Dr. Ing.) dissertation, NTNU, IDI.

[Aam01] Agnar Aamodt. Modeling the knowledge contents of cbr systems. *Proceeding of the workshop program at the fourth International Conference on Case-Based Reasoning, Vancouver*, pages 32–37, 2001.

[Aam04] Agnar Aamodt. Knowledge-intensive case-based reasoning in creek. *Artificial Intelligence*, LNAI 3155:1–15, 2004. Advances in case-based reasoning, 7th European Conference, ECCBR 2004, Proceedings.

[ABB+93] Agnar Aamodt, Bert Bredeweg, Joost Breuker, Cuno Duursma, Christiane Löckenhoff, Klas Orsvarn, Jan Top, André Valente, and Walter Van de Velde. The commonkads library. Document Id: KADS-II/T1.3/VUB/TR/005/1.0, CEC Reference: Draft Deliverable D1.3a, Version 1, 1993.

[Bak04] Elise Bakke. Extended representation for case-based prediction of unwanted events. Project report in TDT4745 "Kunnskapssystemer, fordypning", Fall, IDI NTNU, 2004.

[BSAB04] Tore Brede, Frode Sørmo, Agnar Aamodt, and Ketil Bø. Trollcreek - tutorial. Note by Trollhetta AS and IDI NTNU, 2004.

[CH88] Paul R. Cohen and Adele E. Howe. How evaluation guides ai research: The message still counts more than the medium. *AI Magazine*, 9(4):35–43, 1988.

[Cha86] B. Chandrasekaran. Generic tasks in knowledge-based reasoning: High-level building blocks for expert systems design. *IEEE Expert*, 1(3):23–30, 1986.

[Cha87] B. Chandrasekaran. Towards a functional architecture for intelligence based on generic information processing tasks. *IJCAI-87 Milan*, pages 1183–1192, 1987.

[Cha90] B. Chandrasekaran. Design problem solving: A task analysis. *AI Magazine (winter)*, 11(4):59–71, 1990.

[CJ93] B. Chandrasekaran and T. R Johnson. Generic tasks and task structures: History, critique and new directions. pages 232–272, 1993. From the book 'Second Generation Expert Systems', Springer-Verlag.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms Second Edition*. The MIT Press, 2001.

[CM01] M. Crubézy and Mark Musen. Upml validation and tool support. Technical report, Stanford University, 2001. Deliverable 25-05-2001 in the IBROW Project IST-1999-19005.

[dV93] Walter Van de Velde. Issues in knowledge level modelling. pages 211 – 231, 1993. From the book Second Generation Expert Systems, Springer-Verlag.

[EEMT87] Larry Eshelman, Damien Ehret, John McDermott, and Ming Tan. Mole: A tenacious knowledge-acquisition tool. *International Journal of Man-Machine Studies*, 26(1):41–54, 1987.

[EST+95] H. Eriksson, Y. Shahar, S. W. Tu, A. R. Puerta, and M. A. Musen. Task modelling with reusable problem-solving methods. *Artificial Intelligence*, 79(2):293–326, 1995.

[GMF+02] J. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubézy, H. Eriksson, N. F. Noy, and S. W. Tu. The evolution of protégé: An environment for knowledge-based systems development, 2002. Technical report.

[JAS02] Martha Dørum Jære, Agnar Aamodt, and Pål Skalle. Representing temporal knowledge for case-based prediction. *Advances in case-based reasoning; ECCBR 2002*, LNAI 2416:174–188, 2002.

[LG90] Douglas B. Lenat and R. V. Guha. *Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project*. Addison-Wesley, Reading, Massachusetts, 1990.

[McD88] John McDermott. Preliminary steps toward a taxonomy of problem solving methods. pages 225–256, 1988.

[Mus89] Mark A. Musen. Automated support for building and extending expert models. *Machine Learning*, 4:349–377, 1989.

[Mus93] Mark A. Musen. An overview of knowledge acquistion. pages 405 – 427, 1993. From the book 'Second Generation Expert Systems', Springer-Verlag.

[New82] Allen Newell. The knowledge level. *Artif. Intelligence*, 18:87–127, 1982.

[NM00] Natalya Noy and Deborah L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical report, Stanford University, 2000.

[NM03] Natalya F. Noy and Mark A. Musen. The prompt suite: Interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies*, 2003.

[PTM93] Angel R. Puerta, Samson W. Tu, and Mark A. Musen. Modeling tasks with mechanisms. *International Journal of Intelligent Systems*, 8:129–152, 1993. Medical Computer Science Group, Knowledge Systems Laboratory, Stanford University.

[Ste90] Luc Steels. Components of expertise. *AI Mag.*, 11(2):30–49, 1990.

[Ste91] Luc Steels. Knowledge system. An unpublished English version of a textbook by Luc Steels, 1991.

[Ste93] Luc Steels. The componential framework and its role in reusability. pages 273–298, 1993. From the book 'Second Generation Expert Systems', Springer-Verlag.

[WdVSA93] Bob Weilinga, Walter Van de Velde, Guus Schreiber, and Hans Akkermans. Towards a unification of knowledge modelling approaches. pages 299–335, 1993. From the book 'Second Generation Expert Systems', Springer-Verlag.

[Win96] Ole Martin Winnem. Integrating knowledge level and symbol level modelling - the creest workbench. Master's thesis, NTNU, 1996. Master thesis submitted at the Norwegian University of Science and Technology, Department of Informatics.

[WvAAJ03] B. J. Wielinga, C. van Aart, A. A. Anjewierden, and W. N. H. Jansweijer. Ibrow final report. Technical report, University of Amsterdam, 2003. Final version 29-03-2003.