KADS-II

ESPRIT Project P5248 KADS-II

An Advanced and Comprehensive Methodology
for Integrated KBS Development

Task Features and their Use in CommonKADS

Agnar Aamodt *
Bart Benus †
Cuno Duursma *
Christine Tomlinson ‡
Ronald Schrooten *
Walter Van de Velde *

* *Vrije Universiteit Brussel*
† *University of Amsterdam*
‡ *Lloyd's Register*

Document Id.	KADS-II/T1.5/VUB/TR/014/1.0
CEC Reference	Deliverable D1.5, Version 1.0
Date	January 14, 1993
Distribution	Consortium

KADS-II Consortium

This document is part of a research project partially funded by the ESPRIT Programme of the Commission of the European Communities as project number 5248. The partners in this project are Gap Gemini Innovation (F), Cap Gemini Programator (S), Netherlands Energy Research Foundation ECN (NL), ENTEL SA (ESP), IBM France (F), Lloyd's Register (UK), Swedisch Institute of Computer Science (S), Siemens AG (D), Touche Ross MC (UK), University of Amsterdam (NL) and Free University of Brussels (B).

Cap Gemini Innovation
118, Rue de Tocqueville
75017, Paris
France
Phone: +33 1 40 54 66 32 ; Fax: +33 1 42 67 41 39
Contactperson: M. Georges
E-mail: mari@capsogeti.fr

Cap Gemini Programator
Norra Ringv. 36 Box 7
S-721 03 Västerås
Sweden
Phone: +46 21 14 44 50 ; Fax: +46 21 11 41 00
Contactperson: W. Menezes
E-mail: wm@industri.vs.cgl.se

Netherlands Energy Research Foundation ECN
Software Engineering & Research Department
P.O. Box 1, NL 1755 ZG Petten
The Netherlands
Phone: +31 22 46 44 58; Fax: +31 22 46 18 64
Contactperson: Dr. J.M. Akkermans
E-mail: akkermans@ecn.nl

ENTEL SA
P. de la Castellana 141
28046 Madrid
Spain
Phone: +34 1 348 12 23 ; Fax: +34 1 279 10 74
Contactperson: S. Gala
E-mail: sgala@ic.eritel.es

IBM France
Rue de la vieille Poste
BP 1021, 34006 Montpellier
France
Phone: +33 67 34 40 43; Fax: +33 67 34 48 96
Contact person: G.Rondot
E-mail: rondot@mopprofs.vnet.ibm.com

Lloyd's Register
29, Wellesley Road
Croydon CRO 2AJ, Surrey
United Kingdom
Phone: +44 816 814 040 ; Fax: +44 816 816 814
Contactperson: R. Martil
E-mail: rob@lloydreg.uucp

Swedish Institute for Computer Science
KBS Department
Isafjordsgatan, 16428 Kista
Sweden
Phone: +46 87 52 15 20 ; Fax: 46 87 51 72 30
Contactperson: R. Gustavsson
E-mail: rune@sics.se

Siemens AG ZFE 1S INF 32
Otto Hahn Ring 6
D-8000 Muenchen 83
Germany
Phone: +49 89 636 3361 ; Fax: 49 89 636 441 50
Contactperson: C. Bauer
E-mail: cb@ztivax.uucp

Touche Ross Management Consultants
The Knowledge Based Systems Centre
Peterborough Court, 133 Fleet Street
London EC4A 2TR, United Kingdom
Phone: +44 71 936 3000 ; Fax: +44 71 583 1198
Contactperson: L. Land
E-mail:

University of Amsterdam
Department of Social Science Informatics
Roetersstraat 15, NL-1018 WB Amsterdam
The Netherlands
Tel: +31 20 525 6789; Fax: +31 20 525 6896
Contactperson: B.J. Wielinga
Electronic mail: wielinga@swi.psy.uva.nl

Free University of Brussels
Artificial Intelligence Laboratory
Office: Lentestraat 50-54 Mail: Pleinlaan 2, B-1050
Brussels
Belgium
Phone: +32 2 640 2535 ; Fax: +32 2 640 6326
Contactperson: W. Van de Velde
E-mail: walter@arti.vub.ac.be

This document was produced using the latex wordprocessor

Task Features and their Use in CommonKADS

Agnar Aamodt *, Bart Benus †, Cuno Duursma *, Christine Tomlinson ‡, Ronald Schrooten *, Walter Van de Velde *

* *Vrije Universiteit Brussel*
† *University of Amsterdam*
‡ *Lloyd's Register*

Synopsis

This document summarizes the results to date of Task I.5. in the ESPRIT project KADS-II (P5248) "Analysis of relevant task and domain features".The goal of this task is to define a vocabulary of task reality features - referred to merely as task features - and to relate them to the models developed in task I.3 and I.4. Further, the task features should be accompanied by guidelines for their use in identifying appropriate library components, as well as in supporting knowledge modelling within CommonKADS in general. The approach taken in this task is to lay out spaces of abstract task features, model components of the library, and expertise models under development, and relate them to each other as well as to the space of task realities. A framework has been developed to enable a unified discussion of the topics involved, and to relate the structure and contents of the spaces to their role in the modelling process. The latter enables a bridging to task I.3, and the emerge of a unified view to the use of task features within CommonKADS. The document presents lists of task features and of modelling guidelines, and illustrates their use in the modelling process through examples.

Document Id.	KADS-II/T1.5/VUB/TR/014/1.0
CEC Reference	Deliverable D1.5, Version 1.0
Date	January 14, 1993
Distribution	Consortium

Contents

1 Introduction **3**
1.1 General . . . 3
1.2 Scope of the task . . . 4
1.3 This document . . . 4

I The General Issues **5**

2 A Framework for Task Features and their Use **7**
2.1 Task Realities and Task Features . . . 7
2.2 Spaces of Task Realities and Models . . . 8
2.3 The Modeling Support Entities . . . 11
2.3.1 Abstract Task Features . . . 12
2.3.2 Library Components . . . 14
2.3.3 Modeling Operators . . . 16
2.3.4 Modeling Guidelines . . . 17
2.4 Combined use of model support entities during modeling . . . 19
2.4.1 Fitting it all together . . . 19
2.4.2 Search in a model space . . . 22

3 Task Features **25**
3.1 General . . . 25
3.2 Epistemological, pragmatic, and computational feature types . . . 26
3.3 Main categories of task-features . . . 27
3.3.1 *Roles and interdependencies of the categories* . . . 28
3.3.2 Using the categories for identifying a set of task features . . . 30
3.4 Structuring the inventory of task features . . . 31
3.4.1 Task features related to the task purpose . . . 31
3.4.2 Task features related to the task domain . . . 32
3.4.3 Task features related to the environment . . . 34
3.4.4 Task features related to grounding . . . 35
3.5 Focusing the identification of task features . . . 36
3.6 A mechanism for task feature identification and use . . . 36

4 Modelling Guidelines **39**
4.1 Introduction . . . 39
4.2 Structuring the set of modelling guidelines . . . 40

5 The Model Selection Tool 43
5.1 Introduction . . . 43
5.1.1 Problem statement . . . 43
5.1.2 Preparatory study . . . 43
5.1.2.1 Short evaluation of two existing tools . . . 43
5.2 Specification . . . 44
5.2.1 Problem Definition . . . 45
5.2.2 Requirements . . . 45
5.2.2.1 Task analysis . . . 46
5.3 Description of the Model Advisor Tool. . . . 47
5.3.0.2 Workbench . . . 47
5.3.0.3 Browsers . . . 47
5.3.0.4 Reasoner . . . 48
5.4 Working with the tool. . . . 49
5.4.1 Knowledge level analysis of the tool . . . 52
5.4.1.1 Analysis of the solution . . . 53
5.4.1.2 Comparison with the actual implemented model of MAT . . . 56
5.5 Conclusions . . . 57
5.5.1 Theory . . . 57
5.5.2 Application . . . 57

6 Knowledge Validation 59
6.1 Literature Review . . . 59
6.1.1 Software Engineering . . . 60
6.1.1.1 IEE standards organisation . . . 60
6.1.1.2 Adrion, Branstad and Cherniavsky . . . 60
6.1.1.3 Green and Keyes . . . 60
6.1.1.4 Sommerville . . . 60
6.1.2 Knowledge Engineering . . . 60
6.1.2.1 Suen et al. . . . 60
6.1.2.2 Bensabat and Dhaliwal . . . 61
6.1.2.3 Motta et al. . . . 62
6.1.2.4 VITAL . . . 62
6.1.2.5 VALID . . . 62
6.1.2.6 Ayel and Laurent . . . 62
6.1.2.7 Hoppe and Meseguer . . . 63
6.1.2.8 Meseguer . . . 63
6.1.3 Psychology . . . 64
6.2 Validation of Expertise Models . . . 64
6.2.1 Proposed Definitions . . . 64
6.2.2 Validation of expertise models . . . 64
6.3 Validation and the Modelling Process . . . 65
6.4 Validation and Quality Assurance . . . 66

II The Inventory 69

7 A Task Feature Inventory 71
7.1 Task purpose features . . . 71
7.2 Task domain features . . . 74
7.3 Task environment features . . . 77
7.4 Grounding features . . . 78

8 An Inventory of Modelling Guidelines **81**
8.1 Task identification guidelines . . . 81
8.2 Method identification guidelines . . . 86
8.3 Additional modelling guidelines from Drilling Fluid example . . . 93

9 Using the Inventory in Modelling - A Scenario **97**
9.1 General . . . 97
9.2 Overview of the hospital meal planning application . . . 97
9.3 Functional requirements definition . . . 98
9.4 Analysis with the modelling guidelines . . . 98
9.5 Task identification . . . 99
9.6 Method identification . . . 99
9.6.1 Propose & Revise (mig #1) . . . 100
9.6.2 Means ends Analysis (mig #2) . . . 100
9.6.3 Extrapolate from a similar case (mig #3) . . . 100
9.6.4 Cover & Differentiate (mig #4) . . . 101
9.7 Evaluation of selected generic components . . . 101

10 Conclusion and Future Work **103**
References . . . 104

A Description of the Hospital Meal planner Case **107**
A.1 Original Example . . . 107
A.2 Extended Example . . . 107
A.3 The Protocol Transcript . . . 109
A.4 Task Analysis . . . 112
A.5 Analysis: Starting situation . . . 112
A.5.1 Task analysis . . . 112
A.5.2 Object Analysis . . . 114
A.5.3 Task Feature analysis . . . 114
A.5.3.1 Task Analysis . . . 115
A.5.3.2 Method Identification . . . 116
A.6 Desired situation . . . 116
A.6.1 Task Design . . . 116

Preface

This report represents the results of Task I.5 "Analysis of relevant task and domain features". The objective of Task I.5 is to define a vocabulary of task and domain features, to describe the role these features play in the development of a conceptual model, and - more specific - to relate them to the models developed in task I.3 and I.4.

This deliverable brings together results described in various working documents that have addressed different aspects of the task. During spring 1991 VUB produced a document which suggested an initial structure of task features[1] at a high level of abstraction. This typology was later extended and refined, partially through a general analysis, and partially through the specification of an example application - the Hospital Meal Planner (HMP)[2]. The difficulty experienced in identifying relevant task reality descriptors, and placing them in an appropriate typology, gave useful insights to the role of task features, and to the further conducting of Task I.5. In particular, it was clearly seen that there is a limitation to how deep task features can be described, before assumptions about possible modeling components (domain knowledge type, problem solving methods, reasoning paradigm) are being made. Since the developments in Task I.3 have advanced into a rich and coherent framework for the library the work in Task I.5 has been taking into account the concepts and context of the library work. Basically a large part of the ideas on the CommonKADS library structure stem from work in Task I.5 which is largely manned by the same people.

The task feature framework is now the 'common play ground' for discussing the role and specification of task features and modeling guidelines in the modeling process, as well as the interaction between I.5 and I.3 in general. The recent work has focused on exemplification, and identifying additional task features and modeling guidelines through working on a set of example application (where the HMP is one).

We want to emphasize that after both this task and task I.3 had reached a certain level of maturity, continuous attention has been paid to the developments in I.3, and the results presented here should be regarded as coherent with the library structure and the intended use of the library as described in the D1.3a deliverable [Aamodt et al., 1992].

While focusing on the issues above, less time has been devoted to the issues of knowledge validation. It has been most important to solve the core problems of task feature identification and use, so validation issues could be more concretely related to what there is to validate. Meanwhile a coordinated effort at VUB indicates that validation and quality assurance are tightly coupled and perhaps should be treated together. This implies that the CommonKADS quality perspective [Williams et al., 1992], which is now sufficiently mature could be used as a basis for a more principled approach to the validation issue.

At present the feeling is that work on task features, the library, quality assurance and model states should have been more tightly connected. These elements have now reached a state of maturity and stability so that useful synergistic effects can be expected.

[1] Throughout this document, the term 'task feature' is used in general to mean any relevant descriptor of the task reality. The term therefore subsumes features of the task domain as well as other aspects of the task reality.

[2] Sometimes also referred to as the 'dish selection' application.

Chapter 1

Introduction

1.1 General

Task I.5 is about task features, characteristics of the entire temporal and spatial extension of the world which is relevant for accomplishing a real-life task. The goal of Task I.5 is to come up with a method, including knowledge engineering strategies and supporting tools, for how task features can be used to identify and utilize library components in building an expertise model. The objective is to come up with an inventory of such features, and a set of guidelines for how they can help to make the best possible use of the *Common*KADS library in developing knowledge-level models. This should, of course, be casted within the *Common*KADS knowledge engineering methodology, and supported by tools[1]. Task I.5 is therefore tightly coupled to - and more or less integrated into - Task I.3. It is clear though, that the perspective of I.5 is that of the task features, i.e. their identification and utilization, and not of the library. It is important to note, however, that these are two perspectives on the same knowledge engineering activity, namely the activity of modelling using the *Common*KADS methodology.

Given a definition of the task that a KBS is to perform, the selection of model components from libraries is done on the basis of features of the task (i.e. of the problem to be solved), the task environment and the domain. For example, features of a real-life task may characterize it as being (partly) a diagnostic task. In turn, this diagnostic task may be characterized by the complexity of the solution space, the presence of uncertainty, lack of reliability of the data, the possibility of multiple faults and the need for explanation of the solutions. Such task features determine the requirements for the components from which models for the task can be built.

The approach taken in this task is to describe the abstract spaces of task features, model components, and the target conceptual model, and relate them to the task reality. These spaces are structured according to various dimensions (the structuring of the model component space being the responsibility of Task I.3). Structuring of the task feature space, identifying the relationships between task features and entities of the model component space, specifying mechanisms for using these relationships in model building, and outlining a method for validation of knowledge models, are major ingredients of the task.

The outcome of task I.5 is

- a set of task features characterizing all relevant aspects of the task reality,
- a set of modelling guidelines which links task feature types to appropriate library components, or more generally to operators which select or manipulate these components,

[1] The major tool for task feature analysis and use is referred to as the 'Model Selection Tool'. It is discussed in chapter 5.

- a framework for describing the use of task features and modelling guidelines in the process of developing the model of expertise.

1.2 Scope of the task

Central to the *Common*KADS approach to KBS construction is the concept of modelling. *Common*KADS provides different models as focal points in the construction process. These models are the organization model, the the task model, the agent model, the communication model, the expertise model, and the design model. Task I.5 has been scoped as follows:

> *In Task I.5 we are concerned with task features that relate to development of the expertise model.*

The role of task features is to guide the process of modelling. Guidance for modelling is required when constructing any of the models in the *Common*KADS model set. Yet, in this subtask we are only concerned with guidance for conceptual modelling of expertise. The expertise model is central in *Common*KADS. Guidance for developing other knowledge-level models will be provided by other workpackages, while guidance for design is an issue in Task I.2.3 [Duursma et al., 1992]. However, even if the work in Task I.5 is limited to what is relevant for the expertise model, the results presented in this document may also be useful as a template for other models.

1.3 This document

This deliverable first gives an overview of the framework. This is followed by a description of task features, and of modelling guidelines. Exemplification is used throughout the general presentation, and a set of suggested features and guidelines are listed at the end. An integrated example of a modelling session illustrates all steps in a modelling process, also including the selection and utilization of library components and feedback to task feature identification from the current state of the expertise model.

Part I

The General Issues

Chapter 2

A Framework for Task Features and their Use

2.1 Task Realities and Task Features

Before we explore the role of task features in the modeling process, the term 'task' needs to be clarified, and related to the context within which it is used.

On the one hand there is the task to be accomplished in the real-world. This is a characterization of what the user actually want to achieve, e.g. what problem he wants to solve. Since it is a task of the real world, it belongs the the task reality as such, and not to a model of (part of) this reality.

Definition of task reality

> A *task reality* is the entire temporal and spatial extension of the world which is relevant for accomplishing a real-life task.

An example of a task reality is the case of diagnosing start-up problems of diesel engines by skilled workers in the workshops of the Belgian National Railway Company.

On the other hand there are the various *models* as they are constructed in the CommonKADS approach[1]. These are models of a task reality and they differ in the aspect which is modelled as well as in the nature of the model itself. It is important always to keep a clear distinction between the task reality and a task model.

Note that the task reality changes when a KBS is introduced in it. Specific qualities and limitations in knowledge and skills of agents, will influence the choice of methods and the task-distribution. This may result for example in a different task-distribution. We will distinguish these two task realities as the *initial* and the *target* task reality. The controlled transition from an initial task reality to the target task reality is what the CommonKADS development cycle is all about.

Task features - as viewed in I.5 - are features of the task reality, not features of models describing a task reality. The term 'task feature' may therefore be viewed as a shorter variant of the more precise term 'task reality feature'. In the most general sense, then, a task feature is a characteristic of anything which is potentially relevant for accomplishing a concrete task in a concrete situation.

This general notion of task features is too broad for our purpose. A reasonable question to ask is what *type of information* should task features capture. Any information about the task reality? If so, there would be zillions of task features for most tasks, and making an inventory would be both impossible and useless. Instead we follow a more pragmatic approach, related to the use of task features for modeling within CommonKADS. A task feature, therefore, is not just

[1] Defined in KADS-II/WPI-II/RR/UvA/018/3.0, The KADS-II Model Set

any characterization of a task reality, but a characterization which may *help to find an adequate*[2] *way* of accomplishing the task in it. For example, a task feature may be whether or not precise measurements can be taken in the task reality. If they can be taken, then reasoning may use data intensive methods that require these reliable measurements. If not, heuristic approaches may lead to better results. Another task feature is the time available to accomplish a task. This is usually seen as a requirement, but it has implications for the reasoning method to be used. For example, exhaustive search through a large hypothesis space may be infeasible given the time and the power of the machinery available for performing the task.

When this is view is coupled to modeling within CommonKADS, we arrive at the pragmatic definition which we will use throughout this document:

Definition of task feature

> A *task feature* is any characteristic of a task reality that guides the selection of adequate modeling components from the CommonKADS modeling library.

An immediate consequence of this definition is that task features have no absolute status. They are only useful with respect to a given model library. In CommonKADS a task feature is a feature of a task reality that may guide the knowledge engineer in using the CommonKADS modeling framework and libraries to construct an adequate model.

2.2 Spaces of Task Realities and Models

The framework described in this section offers a unified view on the task reality as well as the various (partial) models derived from it. A 'spatial view' is adopted, in which the task reality, task feature abstractions, library components, and the Expertise Model each is viewed as a separate space. This enables us to distinguish between the *internal properties* of these entities, the *relationships between them*, and to talk about in what way they can contribute to the *modeling activity* of constructing a knowledge level Expertise Model.

[2] Here, 'adequate' not only means 'competent' (i.e. suceeding in principle), but also the ability to comply with the particular circumstances in which a task is to be accomplished.

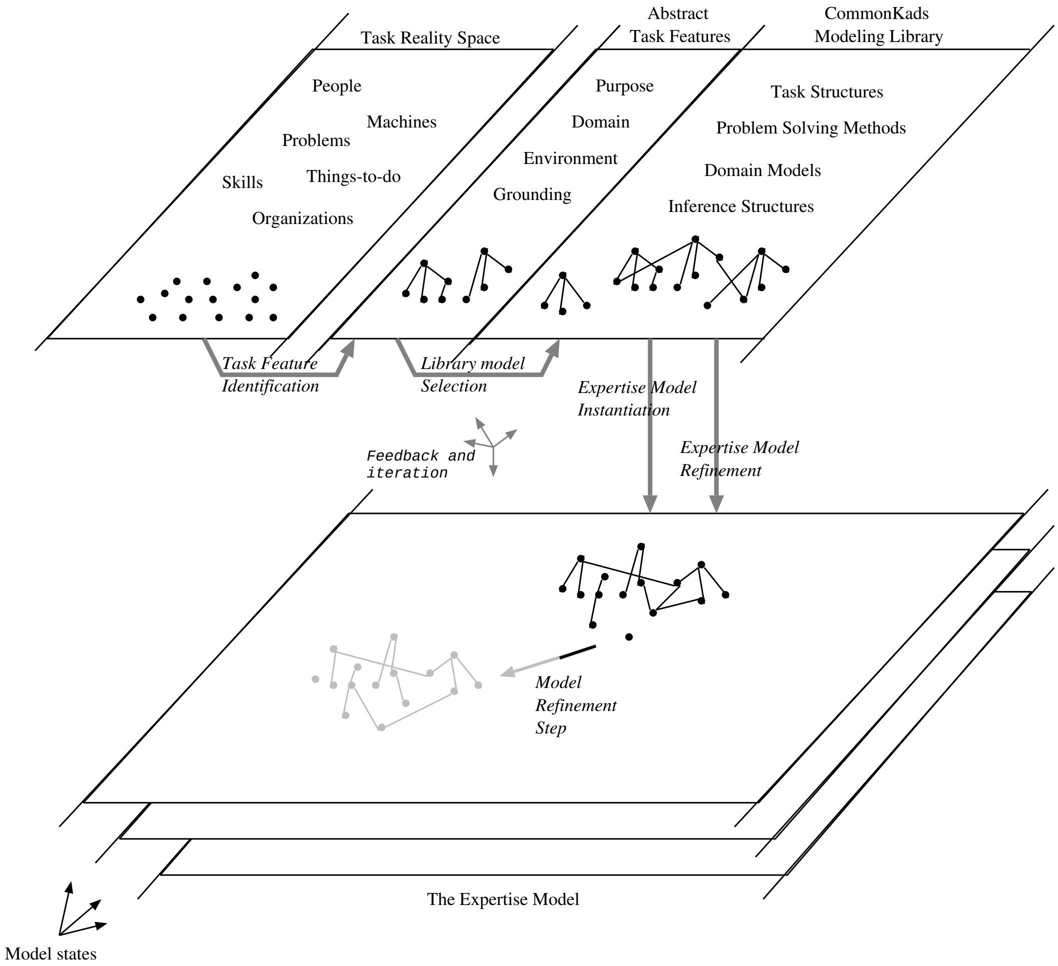

Figure 2.1: Spaces of Task Reality and Models. *The figure illustrates the four spaces identified for discussing the structure and role of task features during expertise modeling. The four spaces are projected onto planes, as shown. The area to the upper left - the Task Reality space - indicates actual situations and phenomena in the real world, the other three areas are model spaces. The dark dots illustrate actual 'things' in the Task Reality Space, and concepts in the other three spaces. The lines between these dots illustrate any type of relationship we may want to express. The thick gray arrows indicate activites of the modeling process which involve transfer of information between description spaces. The lower level of planes illustrates the space of the evolving Expertise Model, where also a transition step between two states of the model is indicated.*

In figure 2.1 the four spaces are pictured as portions of two-dimensional planes. In the upper part of the figure, the spaces of task reality, abstract task features, and CommonKads library components are illustrated. These three spaces are regarded as *fixed* in the sense that no changes to the contents of these spaces are made during expertise modelling. The library and the abstract task features are predefined structures, and although they may get extended and refined over time, for the purpose of their use during conceptual modelling within a particular project, they should be regarded as fixed. Correspondingly, the task reality is also regarded as fixed, since it is the current task reality at the time when modelling is taking place - and the current expectations of the resulting task reality when a KBS is introduced - which is the basis for developing the expertise model[3].

Some activities performed during modeling, in which information in one space is used to get into and explore another space, are indicated by the gray lines between spaces. In figure 2.1, four such activities are shown: Task feature identification, library model selection, expertise model instantiation, and expertise model refinement. Note that, at the present moment, these terms only indicate some important actions undertaken during modeling. We will later be more explicit about this, and also see how the modeling activities can be guided by task features through what is called *modeling operators* and *modeling guidelines*.

Note that only some of the modeling activities - and the corresponding flow of information - are shown. Information and knowledge within the task reality, for example, will - through the expert/knowledge engineer - be used in *every* modeling activity, and hence flow into every model space. However, the whole idea of using task features in CommonKads is as pointers from the task reality to the CommonKads library, while the abstract task features are designed as an help in coupling task features to library entities. Hence, the figure captures the main direction of movement between the spaces - at least in the initial phases of modeling. This is from the Task Reality to the Abstracted Task features, further to the Library, and then to the conceptual modeling space (Expertise Model space). There will typically be feedback and iterations along this path, of course. The entire process will be detailed later in the document.

The growing conceptual model (i.e. the Expertise Model) is illustrated as a set of planes at the bottom of the figure, each plane corresponding to a particular state of the model. During development, the Expertise Model is gradually extended and refined, resulting in a set of model descriptions (model states). In the figure they are indicated by several underlying planes. A transition between two states of the Expertise Model is indicated in the figure by an arrow pointing into the underlying plane.

The notion of a state as used here is a very general one. Some states will be 'milestone states' in the MCC sense. Others will represent steps between milestone states. In the present framework, a state is any static characterisation of a model which it is useful to talk about. In figure 2.1, three states of the Expertise Model are indicated. Through these model states we will have the possibility of explicitly representing the development steps towards the 'final' model. The model states will therefore also be a basis for constructing a *model development graph*, i.e. a trace of model development steps useful to keep for later reuse - as defined in the D1.3 deliverable (KADS-II/TI.3 /VUB/TR /005/1.0).

In the D1.3 document, the notion of an 'expertise model frame' is introduced. This is a general structure within which all the knowledge contained in the Expertise model is described. Knowledge is grouped into the three main categories of application knowledge (domain, task and inference), and problem solving knowledge. Further, knowledge may be described at various layers of precision (verbal, conceptual, and formal). A state of the Expertise model, then, corresponds to a shapshot of a filled-in expertise model frame, as illustrated in figure 2.2. The idea in I.3 is to also use this frame structure to describe one type of the library components, namely the generic models. This is also indicated in the figure.

In the modeling process, the use of editors play an important role, of course. In I.5, one

[3]During development of the model, we will surely get to know more about the task reality than we did when the modelling process started, but this is different from assuming changes in the task reality, as such, during modeling.

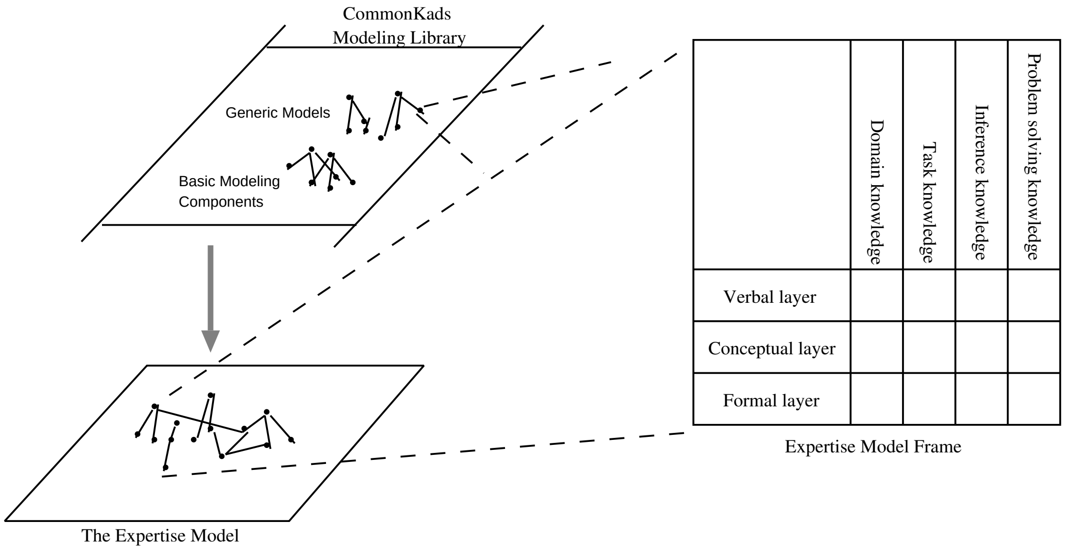

Figure 2.2: The Expertise Model Frame. *The figure illustrates (right part) the general top-level structure of an expertise model. One or more of the fields may be filled in. As also indicated, the same structure is used to organize the description of a generic model in the library. The left part is a 'cut out' from the previous figure, with the difference that the library component space is typed according to the two major component types defined in D1.3. (The previous typing is captured by the columns of the model frame).*

editor - called the Model Selection Tool - is being specified[4]. With respect to the four modeling activities listed above (and shown in figure 1), this editor assists in the *task feature identification* and *library model selection* activities. The activities of *expertise model instantiation* and *expertise model refinement* are covered by the Conceptual Modeling Editor.

The Assumption behind this framework is that the task reality space and the two fixed model spaces depicted in the upper part of the figure, form a sufficient basis for describing an effective utilization of task reality features in conceptual modelling. But so far, only a static picture of the four-space framwork has been given. CommonKads will also provide a *model development methodology* for how to make appropriate use of this structure. This is partly done by specifying a set of guidelines and operators to assist in an effective utilization of the modeling library, given a set of abstract task features and a certain modeling approach - which influences the use and type of modeling guidelines.

Hence, what we want to describe and discuss with respect to the the modeling process, can be captured by four entities: abstract task features, modeling guidelines, modeling operators, and library components. We will refer to them as *modeling support entities.*

2.3 The Modeling Support Entities

What we will refer to as the 'modeling support entities' in CommonKADS are:

```
- Abstract task features
- Modeling guidelines
- Modeling operators
- Library components
```

[4] An initial specification and prototype implementation has been made [Schrooten, 1992]

A summary of what the role of these entities are, and how they are structured, is given in this section. This is followed by a description of how they play together in the process of constructing an expertise model - referring to the task reality and model spaces previously laid out.

Basically, the main dependencies between the modeling support entities are as illustrated in figure 2.3. Modeling guidelines and modeling operators provide a sufficient linking mechanism between the spaces of task features and library components. The figure illustrates the interrelations between the entities in a highly abstracted and simplified way. For example, relations to the growing expertise model is left out, and no iterative relationships are indicated. Throughout this chapter this basic structure will be elaborated and refined.

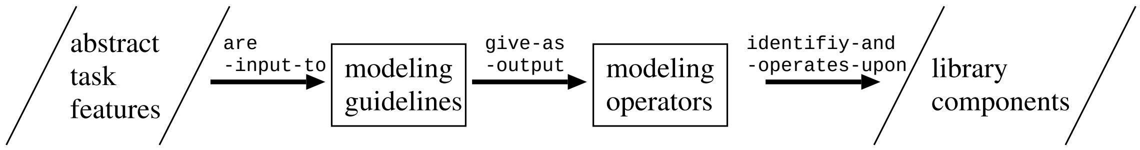

Figure 2.3: Modeling support entities. *The figure illustrates primary relations between the four modeling support entities.*

The main concern of task I.5 is the abstract task features and modeling guidelines. They are therefore described in greater detail than the other two. Library components are described in D1.3. The modeling operators - in a sense representing the entrance door to the library - is also a primary concern of I.3. The summary of modeling operators given below are therefore influenced by current views in I.3, although a perspective of task feature use is taken.

The structure of abstract task features and library components are described first, followed by modeling operators and modeling guidelines. For each modeling support entity, a top-level structure - e.g. a hierarchy of subtypes - is defined[5]. Next, the object structure of a single entity is described. Task features and modeling guidelines are discussed in detail in chapters 3 and ??, respectively.

2.3.1 Abstract Task Features

The number of task features that can be distinguished is as unlimited as the variety of domains of knowledge and real world tasks. Each domain, task and task environment will have it's own specific terminology. The direct use of task features would hardly allow for any kind of generalization. As we want to use task features for the selection of generic library components, this would be unacceptable. Therefore some abstraction of task features from domain and task specific terminology is required.

To abstract from task- and domain specific terminology we have defined a few categories of task features. The categories are divided over two groups which represent two different perspectives. These categories are based on an inventory of more or less coherent groups of abstract entities or aspects that are generic over almost any task and domain. One typology groups task features according to what *aspect of a task reality* they represent, the other according to major *types of constraints* that can be derived from the task reality. The former contains what is referred to as the *main feature categories*. These are features of the **task purpose**, the **task domain**, the **task environment**, and the **task grounding**. The other typology adds a different perspective. As stated in the task definition document (KADSII-VUB-I .5-TD-0002), relevant features for describing the task reality may be classified into *epistemological*, *pragmatic*, and *computational* features. These feature types can be viewed as types of constraints with respect to the four main categories.

[5]Given that the entire framework will have to be matched and tested within a context which also includes unresolved I.3 issues, the typing of the entities should not be regarded as absolute and fixed. This particularly applies to the modeling guidelines and modeling operators

Although the hierarchy represented by subtypes of these classes can be viewed as orthogonal to the other type hierarchy, the feature types do not combine equally well with all the main classes. Epistemological features, for example, define constraints that are more relevant for task domain features than for the other three feature categories.

The abstract feature categories can be seen as a first template, which will initially guide the description of the task at hand in terms of features. The abstract feature categories will provide a structuring of the description of the real life task. The generic machinery for guidance in modeling may be used to realize this structuring of the task description. Still, the translation of concrete task features into abstract ones may require a lot of interpretation of observations by the knowledge engineer. The abstract feature categories will be described in the next subsection.

A high level structure of the Abstract Task Feature space is shown in figure 2.4.

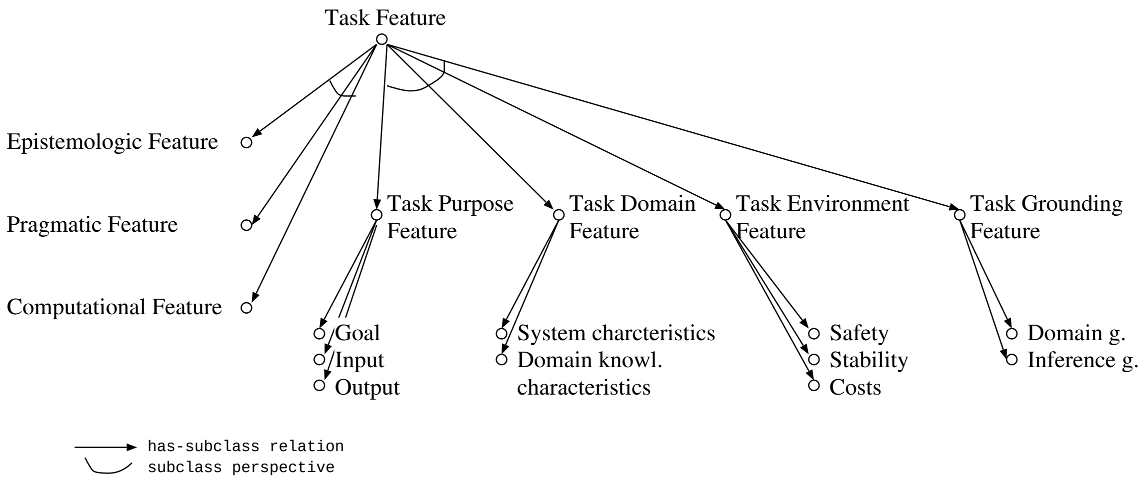

Figure 2.4: Abstract Task Features. *The figure illustrates the top level structure within the Abstract Task Feature space. Two perspectives, more or less orthogonal, are indicated.*

Object structure of a task feature The object structure of a task feature, i.e. the set of descriptors characterizing it, is shown below. The left column names each of the descriptors (slots), while the right column defines the value types each descriptor may take.

```
task-feature <name>
   comment              <textual description>
   subtype-of           <abstract-task-feature>
   has-subtype          <task-feature>
   feature-value        <value>
   value-constraint     <value type, value range, or value set>
   value-format         <objetc-name, predicate, number, list, string>
   question-prompt      <question to promp user for value>
   occurs-in            <modeling-guideline>
   associated-with      <feature value pair>
   external-model-ref   <other model in the model set>
```

A task feature is described by its name, its feature type, its value (if instantiated), the values or value typoes it may take, the format of the feature value, a question prompt for getting it from the user, a list of pointers (names) to the modeling guidelines in which it is referred, associated features, and a pointer to other models in the model set where the feature is described (e.g. task model, organizational model). Associated features is a nested list of other task features which form conjuncts in the modeling guidelines with the feature described. Depending on the semantics of the modeling language, some slots - e.g. value constraints - may be specifed at an appropriate abstraction level, and inherited downwards along subtype links. Or, the abstract features may contain only subtyping information, basically, assuming that semantical constraints etc. are filled in at a more specific level. In the examples below, it is assumed that the three slots 'value-constraints', 'value-format', and 'associated-with' are inherited, and that they define implicit default values for subtypes and instances of abstract features.

Example, an abstract task feature:

```
task-feature system-characteristic
   comment                ``Describes the real world system which the
                            task is about''
   subtype-of             domain-feature
   has-subtype            system-type target-object-type
```

Example, a more specific abstract task feature:

```
task-feature target-object-type
   comment                ``Describes the type of object which the
                            target sytem will reason about''
   subtype-of             system-characteristic
   occurs-in              modeling-guideline-#019, modeling-guideline-#028
   value-format           object-name
   value-constraint       legal-values: human-body, human-mind, organization,
                           people, physical-entity, abstract-entity
```

Examples, instantiated task features:

```
task-feature target-object-type-#1
   comment                ``This system is about planning of meals
                            in a hospital''
   instance-of            target-object-type
   feature-value          organization, physical-entity

task-feature target-object-type-#2
   comment                ``This system is about diagnosis and treatment
                            of pulmonary diseases''
   instance-of            target-object-type
   occurs-in              modeling-guideline-#067, modeling-guideline-#111
   feature-value          human-body

task-feature target-object-type-#3
   comment                ``This system is about assigning a working space
                            and desk to people in a research group''
   instance-of            target-object-type
   occurs-in              modeling-guideline-#089, modeling-guideline-#090
   feature-value          organization, people
```

2.3.2 Library Components

The structure of the library, its component types, and the roles of these in modeling has been described in the D1.3 report. Further, the relationships between library components and task features has also been addressed to some extent - as seen from a library point of view. Figure

2.5 is copied from D1.3, and illustrates the library structure and how library components may be selected by task features, through modeling operators.

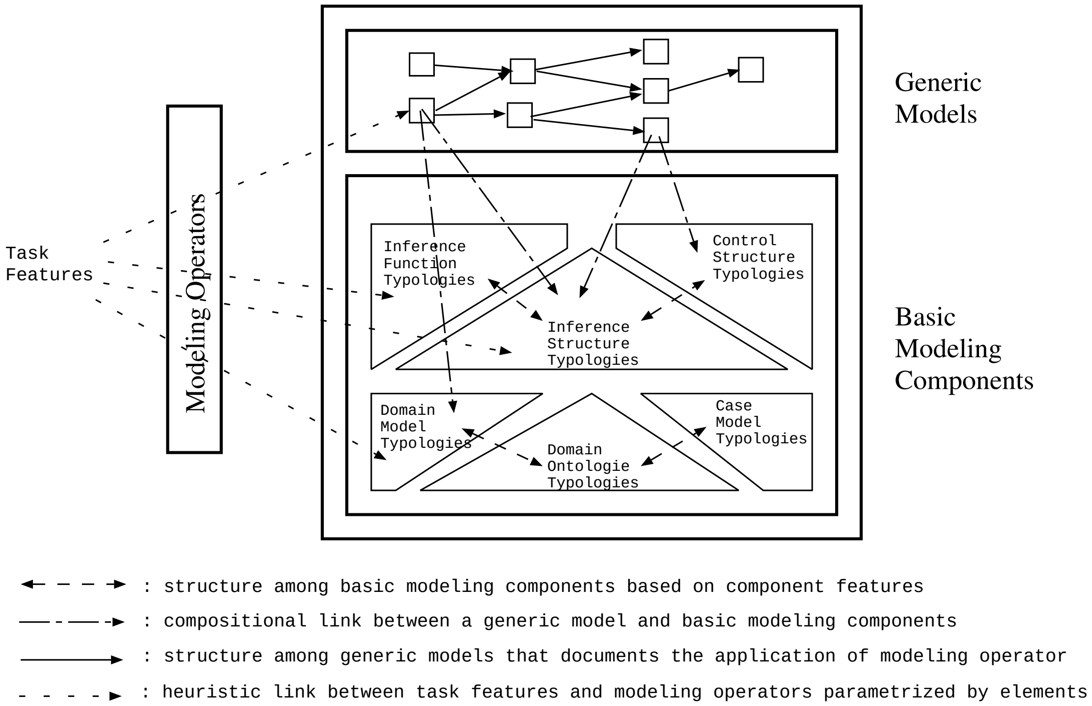

Figure 2.5: Library components and structure. *The library is structured in various ways to support its role of retrieving models and guiding the modeling process. The different structuring principles support access into the library (e.g. from task features), organize the two categories of library elements (component features, typologies and modeling operators) and link them to each other (compositional links). (The layout of the library in the figure is not significant here.)*

The figure presents a view into the 'library space' of the four-space model. In addition, it loosely indicates the coupling to task features. The library contains two types of modeling components, *basic model components*, and *generic models*. A modeling guideline can be viewed as the stipled line between a task feature and a modeling operator. The modeling operator, in turn, selects particluar library components or component types. For the generic models, this process is facilitated by structuring the components according to modeling operators. That is, the set of links between the boxes in the upper part of the figure correspond closely to the set of modeling operators (combine, specialize, instantiate, apply-method-to-task-def, etc.)

Structure of a Library component

Derived from the D1.3 deliverable:

```
generic-model
   subtype-of                          <generic-model>
   part-of                             <generic-model>
   has-part                            <generic-model>
   has-task-knowledge                  <task>
   has-problem-solving-knowledge       <problem-solving-method>
   has-inference-knowledge             <inference>
   has-domain-knowledge                <domain-entity>
   has-operator-link                   <modeling-operator-link>
```

2.3.3 Modeling Operators

A modeling operator basically takes an expertise model in a certain state, performs a modeling operation on the model, and returns it in another state. The granularity of a modeling operator is not fixed to a particular granularity of a state, which means that a particular modeling operator may perform only part of the operation between two given states. A modeling operator consists of two parts: the operation it performs (or assists in performing), and the library comonents, or component types, which th eoperation involves. Some example of modeling operations are:

- Select library component
- Instantiate expertise model
- Refine expertise model

Modeling operators could also be more specific:

- Assemble a model from model components
- Select a complete generic model
- Differentiate knowledge items
- Instantiate a domain structure for a particular domain
- Select a problem solving method for a task
- Apply a problem solving method to a task

The first of these, 'Select library component', is the operation performed by the most simple operator type. Although our framework does not contain direct pointers from modeling guidelines to library components, a selection operator has exactly this effect.

In general, the modeling operators describe operations over partial models. The models may be partial or total expertise models, as well as generic models in the library. [What about inter-dependencies between library components, i .e. one library component which strongly suggests the use of another? This also seems like a reasonable input to the selection of the next modeling operator.]

Modeling operators are structured into an operator hierarchy, with (e.g.) *Build model* as the top level concept. In figure 2.6 a possible top level hierarchy of modeling operators are illustrated. The top node is typed along two dimensions. One dimension expresses the major operation type (select, combine, ..), while the other is a typing according to whether model structures are build or model contents is filled in.

Structure of a modeling operator

Slots of a modeling operator:

Modeling operators are structures which contain two parts: The operation it performs and the library component(s) for which the operation applies. Instead of specifying a particular library component or component type, a modeling operator may specify a set of *library component features*, i.e. characteristic properties of library components for which the operator applies[6]. A modeling operator further contains a pointer back to the modeling guidelines in which it is referred, and assumtions about the resulting expertise model after application of the modeling operator. There are also assumptions on the current experetise model for which a modeling operator is applicable, but this is covered by a condition in the modeling guidelines (more specifically, the 'expertise-model-state' slot of the modeling guideline, as described in the next subsection). The entire picture of how these elements interact will be further elaborated in chapter 2.4.

[6] It is still an open question what the roles of library component features are, and how they should be used. This will have to be sorted out through common example exercises with I.3. Here, the slot is included to cover for the possible use as described.

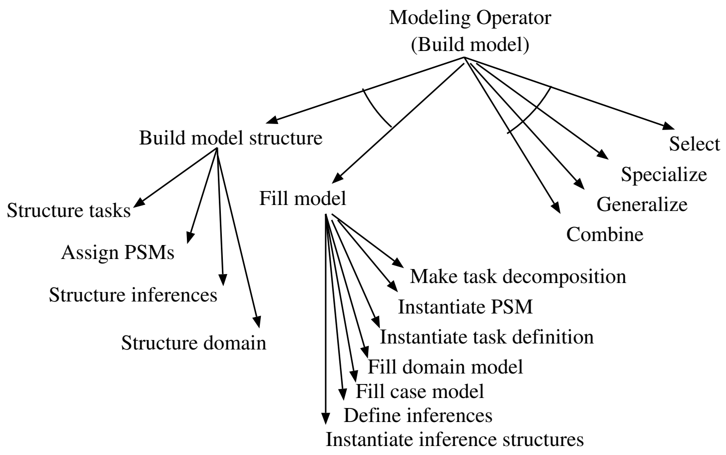

Figure 2.6: Modeling operators. *A top-level type hierarchy of modeling operators. Two subtype hierarchies (two typing dimensions) are indicated.*

```
modeling-operator
   subtype-of                           <modeling operator>
   operation                            <operation>
   library-component                    <library component>
   library-component-feature            <feature of library component>
   from-modeling-guideline              <modeling guideline>
   resulting-expertise-model-assumption <assumption statement>
```

Example of a modeling operator:

```
select-task
   subtype-of                           select, structure-tasks
   library-component                    task
   from-modeling-guideline              modeling-guideline-#004
                                        modeling-guideline-#019
                                        modeling-guideline-#129
   resulting-expertise-model-assumption task-knowledge = partially-filled
```

2.3.4 Modeling Guidelines

Modeling guidelines are associations between task features and library components. But that is a very general characterization, which does not say how the association is made, whether it is direct or indirect, whether task features is the only type of input, etc. As previously described, library components are identified and accessed through modeling operators. Which means that what a modeling guideline gives as 'output' is a modeling operator containing an operation type (select, combine, ..) and the type of library component(s) that the operation should work on (e.g. task, method, ..., or more specific).

The goal of a framework for task features is to provide for a way to formulate and exploit associations between task features and (applications of) modeling operators. However, in a practical modeling situation, task features will never be the only indicator for modeling operators. Two other important indicators are derived from the *current state of the Expertise Model*, and from the *current state of other models in the CommonKads model set.* The task features must therefore be related to the contributions to the modeling process provided by these two sources. Associations to modeling operators may come from one of the three indicator types, or from a combination of them. It is these associations we will refer to as modeling guidelines.

Modeling guidelines may be viewed as rules, where the If part contains descriptive states of one or more of the indicator types mentioned above, and the Then part contains the modeling operator. The modeling guidelines capture associations at various level of generality, and we may want to structure the rule set in a type hierarchy. In figure 2.7 a possible top level structure is illustrated, where the typology is according to the type of modeling operator a modeling guideline point to. Note that this is an initial outline only. We need to work through examples in order to come up with the most useful structure for these entities.

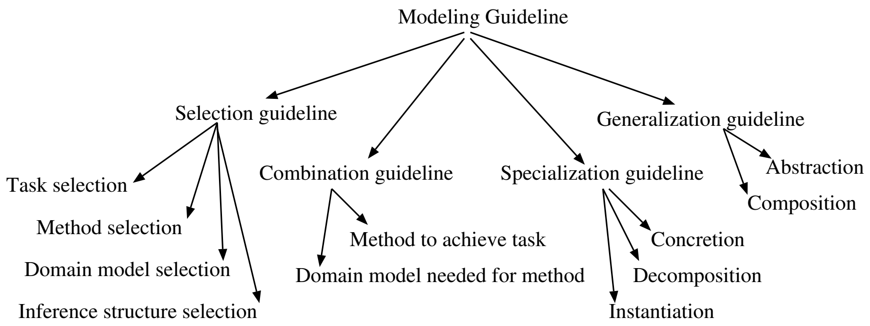

Figure 2.7: Modeling guidelines. *A top level type hierarchy of modeling guidelines.*

Note that the typology chosen here matches one of the typology dimensions of modeling operators. Hence, this typology hierarchies can be used in identifying and refining modeling operators appropriate for certain modeling guidelines, and vice versa.

(An integrated view, summarizing how modeling guidelines and modeling operators play together, is given in chapter 2.4, and illustrated in figure 2.8.)

Structure of a Modeling Guideline

A modeling guideline contains a set of conditions and a pointer structure indicating one or more modeling operators. A pointer to a modeling operator contains one or more modeling operators, and possibly also a type of library component that an operator applies to, and a preference ranging of the operator+library component combination. The latter enables expressing whether a particular modeling operator normally is preferred over others, whether some should not be used, etc.

The language in which modeling guidelines are expressed must allow the slots to be related to each other as conditions and actions, in a rule-like manner.

Slots of a modeling guideline:

```
modeling guideline
   sub-type-of                 <modeling guideline>
   task-feature                <task feature - value pair>
   expertise-model-state       <expertise model state>
   external-model-state        <milestone state>
   operator                    <modeling operator>
   operator-preference         <preference indicator>
   suggested-next-operator     <modeling operator>
   result-validated-by         <validation procedure>
```

Example, a modeling guideline:

Textual rule form:

If the goal of the task is to find an error, then the task is diagnosis.

```
modeling-guidelines-#003
   subtype-of                task-selection
   task-feature              task-goal = find-error
   expertise-model-state     task-knowledge = empty
   external-model-state      task-assignment = kbs
   operator                  select-task(diagnosis-task)
   suggested-next-operator   decompose-task
   result-validated-by       expert
```

A more detailed example:

Textual rule form:

If the goal of the task is to combine a number of components in order to create a device with a certain functionality, then the task is design

```
modeling-guideline-#014
 subtype-of                  task-selection
 task-feature                goal = combine-component
                                     IN-ORDER-TO create-device
                                      WITH functionality = specified
                             system-characteristic = EXISTS(components)
 expertise-model-state       task-knowledge = empty
 external-model-state        task-model = tasks-assigned-to-kbs
 operator                    select-task(design)
 suggested-next-operator     decompose-task
 result-validated-by         knowledge-engineer
```

Note that the description language has been somewhat refined in the above description compared to the previous example, in order to express a more complex goal. The description should be regarded as structured text, though, unless we want to write an interpreter for it (large letter terms express relationships between adjacent entities):

2.4 Combined use of model support entities during modeling

2.4.1 Fitting it all together

We will here assume that the library contains partial models at various levels of specificity. The primary use of these components is as suggested structures, to be assessed for relevance by the knowledge engineer, followed by a possible modification and instantiation in order to be integrated into the current state of the expertise model. In this process, the significance and usefulness of a task feature will depend upon the current state of the modeling process as a whole. During the initial steps of expertise modeling, for example, a first attempt will be made to select an appropriate generic model - or other library component(s) - from the library. At this point, the Expertise Model contains the model template only/footnoteAs defined in KADS-II/WPI-II /RR/Uva /018/3.0, The KADS-II Model Set, where the knowledge structure is according to the expertise model frame illustrated in figure 2.2. The global model construction process is basically iterative, incremental and uses, apart from selection, also various forms of specialization, composition and instantiation. Gradually the model template is filled, which implies that particular submodels will be instantiated, refined, validated, etc. The different operations involved in this correspond to particular types of modeling operators.

Modeling guidelines, then, support the selection of modeling operators on the basis of task features and the current modeling context - described by states of models in the model set. This is illustrated and briefly described in figure 2.8. Here, the 'four-space' model described in chapter 2.2 is extended with a fifth space - that of other models of the CommonKADS model set. This makes the picture complete with respect to the modeling guideline inputs. It is reasonable to assume that pointers from other MCC models will

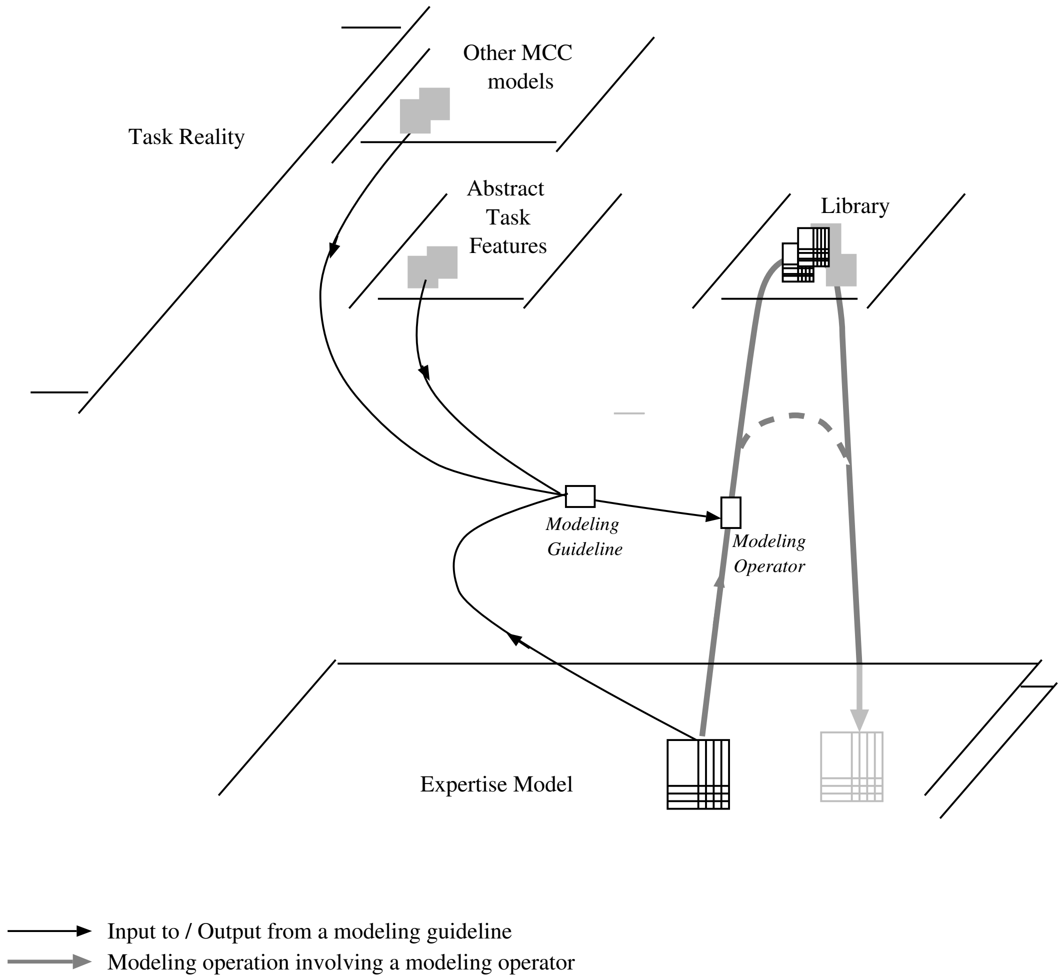

Figure 2.8: Modeling support entities in the modeling process.*The figure illustrates the role of modeling guidelines and modeling operators in a modeling operation (a modeling step). They are both shown as rectangular boxes on directed lines between modeling spaces. The boxes indicate that they are not merely pointers, but structured objects which takes some input and produces some output. A modeling guideline takes as input one or more abstract task features and parts of the Expertise Model, as well as states of other models in the MCC model set, and outputs a modeling operator.* A modeling operator *operates on the current state of a partial expertise model and modifies it into a new state. It takes a part of the Expertise Model as input and produces a modified model. It will typically use the library in this process, but not necessarily (the optional 'shortcut' is indicated by a dashed line).*

refer to their 'milestone states', while finer grained model states are needed for the Expertise Model. Throughout this document, the input types focused are those from task features and from the expertise model - inputs from other MCC models will not be further discussed in this document (for reasons of clarity and simplification).

As described in the previous subchapters, there are several types of modeling operators. The role of a *modeling guideline* is to select an appropriate modeling operator, given a set of task features, information related to the current state of the Expertise Model, and information about milestone states in other models of the CommonKads model set. The modeling operator is a structure containing, among other things, a modeling operation (e.g. 'find a method to decompose a task', and a set of 'parameters'. The actual job of the modeling operators are to map task features etc. to library components (or, more precisely, to certain features of library components), guided by the modeling guidelines. Object structures of both modeling guidelines and modeling operators are described in subsequent sections of this chapter.

[This description should be elaborated, and a textual example be provided, which is later described in terms of the support entity structures (after the next section). Maybe the scenario chapter can do a lot of the job. Also show the interplay between modeling guidelines and the library structure, e.g. by how a Refine-domain-model operator translates into a retrieval of the corresponding domain model in the library, followed by a refinement of this model into its submodels, which are then presented to the user, possibly with some suggestions. Iterative process.]

2.4.2 Search in a model space

In order to relate task features to their use in the actual modeling process, the framework also needs a way to talk about this modeling process as such, in a more detailed manner. This can be done by viewing model construction as a search process in a space of possible models. The notion of state space search, and the analytic and descriptive apparatus related to that view, could then be applicable to describing the roles of task features and library component usage in the modeling process. It has also been shown [Duursma, 1992a] that this view is a good candidate for describing and analyzing problem solving methods, and domain model organization, in general. Hence, the use of the library components themselves - or at least some types of them - in the process of building an Expertise Model, could be described uniformly by relating them to how they assist model construction within the 'state space search' language.

Figure 2.9 illustrates a possible interpretation of a state space for this purpose. It is characterized as follows:

- A *state* corresponds to a state of - a snapshot of - the expertise model. Its knowledge content is structured according to the expertise model frame (figure 2.2).

- A *start state* is the expertise model template, defined as part of the CommonKADS model set, without any knowledge filled in.

- An *end state* is an expertise model which is sufficiently developed to be delivered as part of the CommonKADS Product model. Or, a sufficient expertise model according to other criteria.

- An *operator* is a modeling step in the development of the expertise model. It is a general term which captures any kind of transition between states. It corresponds to a modeling operator, as previously defined, in the general sense of this term. However, if modeling operators are defined in terms of a predefined, limited set of operations, an operator in the state-space may or may not correspond to a modeling operator in that sense. Operators form a hierarchy from very abstract operators (e.g. 'construct model') to very specific ones (e.g. 'apply the plausibe-causal-reasoning method to the task').

- A *search method* corresponds, in general, to the arsenal of strategies, methods, and techniques which the constructor(s) of the knowledge level model has at his (their) disposal. Search methods are at various levels of granularity, i.e. they may cover one or a few operators - or a longer sequence of them.

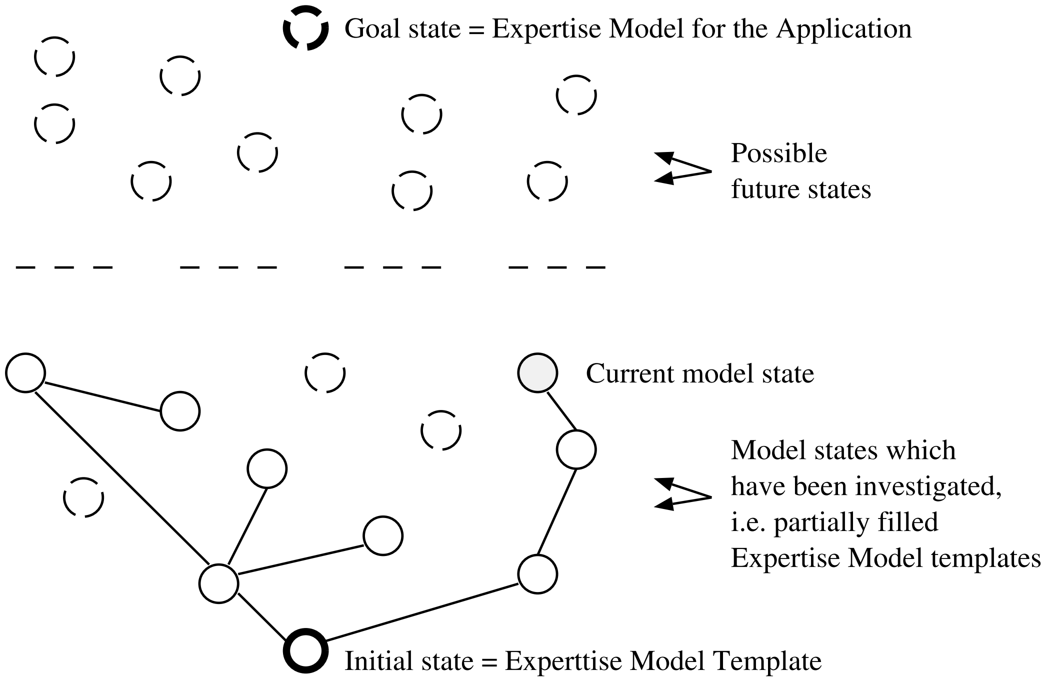

Figure 2.9: A state-space-search view of modeling. *The figure illustrates the structure within the space of possible model states. A snapshot of the current state and its dvelopment history is shown. Closed circles indicate states that have been visited, while open circles are non-visited states. Lines are operators that have been applied - connecting states that have been activated.*

Note that state-space search in AI also has been used to describe how to design and implement systems (e.g. best-first-search algorithms). What we are addressing here, however, is the use of the state-space-search view as an approach to expertise modeling, with no implications to design issues.

There is, in this description framework, no assumptions about how the search process may proceed (since this is part of the method specification). For example, the search path history shown in the figure (by the lines connecting already vistited states), may result from a depth-first, breadth first, or combined search. For example, the picture in figure 2.9 may indicate a previous activation along the 'left branch', followed by a backtracking which has lead to the current model state. This would imply that the continuation from the current state will extend from the 'curent model state' node. But the path history may also have been achieved in a breadth-first way, by which a continuation could proceed from any of the terminal nodes [7].

The state-space-search view thus gives us a terminology for discussing the construction of an expertise model, by relating it to the building blocks of our overall framework. In order to do do that, we need to map the terms of the state-space view to the terms of our framework. This will be done later in the document, after the structure, contents and use of the modeling support entities

[7] An ideal development methodology, of course, generates only one next state (the best) and avoids backtracking. That is, search is avoided. Although the CommonKADS methodology hopefully gets us close to this, for all by the simplest applications search is unavoidable

have been described in more detail. The idea is to use the state space view to better understand the mechanisms involved in the continual development and 'growth' of the Expertise Model - as pictured in figure 2.8.

Chapter 3

Task Features

3.1 General

An important part of Task I.5 is to make a structured inventory of relevant task features for *Common*KADS modelling. This chapter describes the detailed structure of the task feature space, based on the introduction in the previous chapter. A list of all the task features identified up till now are given in Part II of the document. Task features have been analyzed and collected from various sources, including literature, case studies, and theoretical analysis. In general a task will have a very large number of features. In accordance with the pragmatic definition of a task feature given in chapter 2, the analysis and identification of task features have been limited to those regarded relevant for modelling within *Common*KADS.

In this chapter, the task feature space is elaborated by detailing the roles and interdependencies of the abstract task feature categories. Further, a more detailed structure of the space is defined, by a refinement of the main feature categories into sub-categories.

In the preceding chapter, two hierarchies of abstract task feature categories were defined. One defines the following three categories:

- Epistemological features
- Pragmatic features
- Computational features

These classes represent one way to split up the task feature space. The other perspective distinguishes between features according to what aspect of the task reality they describe:

- Task domain features
- Task purpose features
- Task environment features
- Task grounding features

The latter perspective is regarded as more operational with respect to identifying a set of features for selection of library models. They can be viewed as describing the task reality in terms of meaningful real world entities and the relationships that exist between them. Hence, these four feature classes are proposed as the *main categories*. Epistemologic, pragmatic and computational categorisations can be viewed as a general partition of knowledge, but here we will view these types as constraints to the description provided by the main categories. Given an entity of the task domain - a patient in a medical diagnosis system, say - epistemological features characterize inherent properties of the knowledge needed in the task. An example is that deeper knowledge than pure heuristic rules are needed to accomplish the task.

In this chapter we first elaborate on the feature classes represented by the epistemologic-pragmatic-computational perspective, before describing the four main categories in more detail.

3.2 Epistemological, pragmatic, and computational feature types

In this section we propose a first categorization of task features.

Epistemological features

Epistemological features describe inherent properties of the knowledge needed to solve the task, i.e. the nature of the knowledge which should be available to a problem solving agent in order to transfer an input specification (a problem) to an output (a solution). Epistemological features characterize properties of this knowledge as such, i.e. of the knowledge needed to perform the wanted operations. The starting point is to identify characteristics of knowledge used by human agents when solving problems within the task reality:

- Whether the knowledge which is needed within the task is based on heuristic experience and/or a fundamental theory - i.e. the *depth* of the relationships between knowledge units.
- At what level of generality (scope) the knowledge is, ranging from very general constructs to case specific knowledge constructs - i.e. the *breadth* of the knowledge.
- How tight the knowledge is coupled to its use - i.e. the degree of *operationality* of the knowledge.
- The certainty of the relationships in the domain - i.e. the *strength* of the domain theory.
- The number and interdependency of the relationships involved - i.e. the *complexity* of the knowledge model needed to solve the task.
- To what degree a set of relationships exists which captures all the various types of knowledge needed to solve problems in the task reality - i.e. the *completeness* of the knowledge.

Pragmatic features

Pragmatic features of a task reality describe properties related to the input data, to the practical consequences of epistemological features, and to the output from a problem solving process:

- Pragmatic issues related to data acquisition, such as availability of test equipment, cost of data acquisition (time, money, personnel), reliability of input data.
- Pragmatic issues related to domain epistemology, such as the feasibility of modelling the various types of knowledge (e.g. availability of relevant domain expertise, cost and risk of failure of developing a particular knowledge model), availability of data types within the task environment, availability of previous cases and production rate of cases, dealing with uncertainty in input data, in the knowledge, and in the output, deciding on the degree of complexity needed in the knowledge model, dealing with incompleteness, uncertainty and inoperationality of knowledge in terms of appropriate control strategies, search methods, redundancy techniques, justification methods, etc.
- Pragmatic issues related to the output produced from a problem solving process, such as the intended use of the output, the degree of confidence in it, whether it has to be transformed for further use, etc.

Computational features

Computational features of a task reality describe properties related to the anticipated target system - i.e. the computing system realized (implemented) on the basis of a conceptual model. The focus in KADS-II is on conceptual modelling, but the types of constraints given by the fact that the conceptual model is to be implemented in a computer system is nevertheless of relevance when selecting library components. The impact of computational features may be grouped into two classes:

- Computational realizations of methods.The computational requirement of realizing a certain problem solving method, or primitive inference, may be small for some types of domain knowledge, but large when applied to other knowledge types (e.g. more complex knowledge, involving several relation types). Thus, when considering a certain method, computational features such as tractability and resource requirements (time, computer power) should be taken into account.

- Introducing a computing system into the task reality. This is related to the fact that the resulting computing system will end up being a part of the eventual *target task reality*, and the computational properties of the system will therefore influence this task reality. Hence, computational features related to performance, speed, cost, and integration into and interaction within the target task environment, put some constraints on the feasibility of models and methods.

3.3 Main categories of task-features

There are many task features and it is in principle impossible to foresee a complete, generic set of them. The reason is not only that there are so many. Also, the language for task features is in principle the language used to describe the task in the real world. Thus, any different domain, task or task environment may require a different language. This situation makes the direct usage of the task features unpractical for modelling, since it is not easy to generalize over different tasks.

To solve this problem we propose the following. First we organize task features by classifying them into a number of meaningful and useful categories, using the four main categories. Next, we introduce a hierarchy of abstract task features within these categories. An abstract task feature means the same as a concrete task feature but makes sense for a wider range of tasks. The idea behind this organization is that whenever a concrete task feature is encountered it should be easy to determine first what aspect of the task it characterizes (and thus to determine the class of task features it belongs to) and next to identify one or a number of abstract task features that express the same characteristics. From this point on the generic machinery for guidance in modelling can be used. Note that there may be a process of interpretation going on in translating from specific to abstract task features, a process which may be very difficult to automate.

As introduced in chapter 2, we propose the following main categories of task features:

- Features of the task purpose
- Features of the task domain
- Features of the task environment
- Features of grounding

We conjecture that these classes make sense and that they are sufficient to encompass all relevant task features. Basically the claim is that the task reality can be completely described by describing the task domain, the task purpose, the task environment and grounding of the intended agent within its actual operating situation.

3.3.1 *Roles and interdependencies of the categories*

The roles and main dependencies between the feature categories, as well as how they relate to an agent who performs a task, is outlined below. With respect to the expertise model, the agent is the target KBS system. Note that the agent is not part of the task feature framework, since the scope of task features is with respect to the Expertise model only. The properties of agents are described in the Agent model.

The task reality is exhaustively and disjointly partitioned into the *system which the task is about*, the *task environment*, and the *agent who performs the task*. In figure 3.1 this divisioning of the task reality is illustrated, within a broader context which includes the four feature categories and some essential relationships between them.

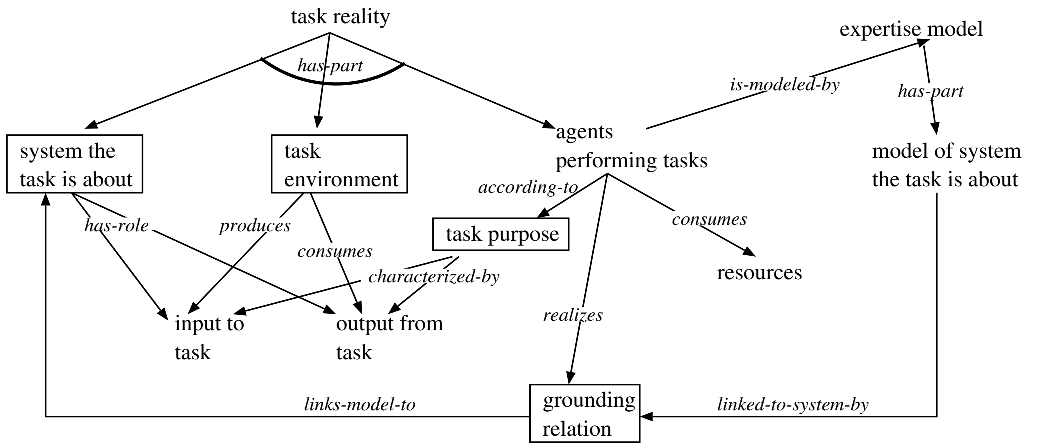

Figure 3.1: Task Reality and Models. *The figure illustrates the partitioning of a task reality. From the perspective taken here, the partitioning is complete and disjoint (indicated by a thick arc). The nodes in the network are characterizations of task reality or model entities. An entity may also be a relationship, as is the case for the grounding relation. Relations between the entities are named along the directed edges, and should be read in the direction of the arrows. The four boxes are nodes corresponding to the four task feature categories.*

Agents perform tasks - i.e. they attempt to accomplish tasks - according to the purpose (goal) of the task. Agents are part of the task reality but not part of the task domain since the task is not about them. They are (in general) not part of the task environment since they are not merely the producers of task inputs or the consumers of task outputs. They are also active problem solvers, perhaps taking initiative, implementing proposed solutions, doing part of the reasoning, producing partial solutions, performing tests proposed by another agent, etc. Resources are consumed by agents during the realization of various grounding relations in the problem solving process. Another characteristic is the fact that to any agent a model of the task-reality can be attributed. This model is derived from (implemented by use of) the expertise model. An agent must be thought of as a logical entity. Mapping physical entities in the task reality - as far as they are not part of the task environment (see before) - to logical agents, is part of the Agent model specification. Different agents can be embodied by the same physical system or person or vice versa.

On this background, features of the four task feature categories - enclosed in boxes in figure 3.1 - have the following general properties:

The *task purpose* specifies the *goal* of the task. The question is asked what one wants to achieve with the systems in the domain. For example the goal may be to bring a motorcycle into a working state, or to design a schedule for train personnel to assure a set of services between various stations - which also complies with social regulations. With respect to a task goal certain systems in the task domain fulfill the role of *input*, others of *output*. The input to the scheduling task, for example, may be the system of train services and the output a system of job assignments. Other systems in the task domain provide additional constraints on the task goal. For example in producing a system of job assignments, it must be assured that it complies with the system of social regulations. The task goal describes the intention of the agent performing the task in terms of the relation between the input - and the output role[1].

The *task domain* consists of the *systems* that the task is about. A system is an abstract or physical object or process in the task reality that is the subject of the task. The concept of system must be taken very broadly and anything that can be reasoned about can play the role of a system. Example systems are the human body, a motorcycle, a refinery process, stereo equipment, a network of train stations, a set of laws and regulations, a knowledge acquisition process. A task domain may consist of one or a set of systems. For example a task reality may consist of a system of train services between stations, a system of train drivers, a system of social regulations for working conditions and a system of job assignments to train drivers. Another task reality may contain just the motorcycle system. The system the task is about defines the application domain of the task. A system can be a patient in a hospital, a motorcycle, a particular plan, etc. The role of this system during task performance is to serve as input to and output from the actual task. This corresponds to two different states of the system - an input state, for example a faulty system, and the output state a repaired system. To make the domain features category more generic, we will have to abstract the actual systems in the world to *system types*[2].

The *task environment* is that part of a task reality that produces inputs and consumes outputs of the task. That is, the actual input values (measurements, answers to questions, volunteered user information, etc.) to the task are produced by the task environment. It also consumes outputs from the task performing process. The task environment for the task of repairing a motorcycle contains the owner who will have to pay for the job, the conditions in which the motorcycle is used, and so forth. The task environment restricts the set of task behaviors that are acceptable. These restrictions are not reasoned about, because then they would be systems and part of the domain. Rather, a problem solver is constructed in such a way as to comply with these features of the task environment. For example the personnel scheduling task requires to explicitly take into account the system of social regulations. Defining the boundary between domain and environment is part of the knowledge engineer's modelling task. Another important distinction within this context is the difference between the *initial* task-environment (before introduction of the KBS) and the *target* task environment (created by introduction of the KBS as an agent in the task).

The *grounding relation* relate a system in a task domain to its model as constructed by the agent involved in the task, i.e. the KBS in our case. In general, grounding concerns the relation between the actual system the task is about and an agent's model of this system, which relation is achieved by the agent's interactions with the actual system, the task-environment as well as other agents involved in the agent's task. An example of a grounding relation is the relation between a motorcycle and its model in a KBS. The user of a KBS, the KBS itself, a domain expert who cooperates with a KBS or a robot arm that works on a motorcycle are examples of agents. Agents consume *resources* when realizing grounding relations. Example resources are cost of observations, computing time, and real time available for doing a task. Since the relationship with the real system is focused, the model itself is initially regarded as a black-box. When identifying grounding relations, we are not initially concerned with the question whether or not the model can express

[1] As previously mentioned, input and output roles must not be understood in computational terms i.e. as input and output of a KBS. Instead they refer to the actual systems in the world. For example the input and output can be the same physical system at two different points in time. A malfunctioning motorcycle may be considered an input for the task of repairing it. The same motorcycle will then be the output as well, except that it is repaired. However, if we want to use the task purpose as part of a modelling guideline, we will have to abstract from the actual systems in the world to system types.

[2] There is a direct connection between this typology of abstract systems and the organizing principle of "fields" as proposed by Task I.3 for the library [Aamodt et al., 1992]. A hierarchy of system types - called 'industries' - is used as the primary feature category in the Spark-Firefighter-Burn methodology [Klinker et al., 1991].

the right aspects of the task reality for the given task. Answering this question, would mean the specification of the results of the knowledge engineering process in advance. Instead, the initial focus is on the way the agent interacts with the task reality, in order to have it's models state reflect the state of the actual system about which the reasoning takes place. If the system is an electrical circuit, for example, a possible relation between the actual circuit and the KBS model of the circuit is realized by a human agent, measuring a current in the circuit and providing the KBS with the value measured.

3.3.2 Using the categories for identifying a set of task features

Our space of task features has now been partitioned into the four main categories related to the task domain, the task purpose, the task environment, and the grounding. We also have an additional, more or less orthogonal partition of the task space, according to epistemological, pragmatic and computational feature dimensions. As previously stated, our claim is that the four main categories are sufficient, i.e. they cover all properties of the task reality which are relevant for conceptual modelling. The orthogonal dimension provides a focus within these categories.

It is important to understand the actual dynamics of using abstract task features, to support the claim that the categories we propose are sufficient. Modelling in *Common*KADS consists basically of the selection of domain models and task decompositions, possibly driven by the selection of a problem solving method. The point is that the selection of one such component restricts the selection of the others.

The following scheme shows how the structure of the model library leads one from one task feature to another, each belonging clearly to one of the categories. In this way features of mixed categories (for example, features of the task domain which are specific for diagnosis) may be avoided.

```
  Task Reality Feature          Task Model in Library
   -------------------          ---------------------
* Task purpose:
   Finding out what's
   wrong with a car       <--->   diagnosis task
                                        |
                                (has possible method)
                                        |
                                  causal reasoning
                                        |
                                   (requires)
* Task domain:                          |
   Knowledge explaining                 V
   car behaviour is needed <--->   causal model
```

The expressions in parentheses are relationships within the library - directly (or closely) matching modelling operators. In the above scheme the selection of causal reasoning for diagnosis based on a causal model could as well be recommended by a mixed task feature, like the one identified by the question "Is a causal theory available that allows to explain anomalous symptoms of a car?". A somewhat intelligent use for the task features as we envisage them can lead to such a question. The point is however that the causal model may be indicated for a task like behavior prediction (envisioning) of a car, based on the same task feature. Note that, if a model for causal diagnosis were present then it would be indicated by the conjunction of the task features.

```
  Task Reality Feature          Task Model in Library
   -------------------          ---------------------
* Task purpose:
```

```
   Finding out what's
   wrong with a car      <--->   diagnosis task
                                   |
                                   |
                                   \                      causal
                                    >-(combines into)--> diagnosis
                                   /                      model
* Task domain:                     |
   Knowledge explaining            |
   car behaviour is needed <---> causal model
```

3.4 Structuring the inventory of task features

On the basis of the four top-level feature classes (categories), a set of more specific feature classes has been derived. A top down analysis has been made, by gradually refining the four classes. The orthogonal dimension of epistemological, pragmatic, and computational features has been used as guidance, and the anticipated usefulness for library components selection and model construction has focused the work.

A structure of task feature types within each main category is presented in this section. In Part II of the document, the initial filling of this structure is presented. Three example applications are analyzed for that purpose, and an inventory of task features, questions for extracting the feature values, and corresponding modelling guidelines, is initialized.

Since the main role of the task features is to help in selecting and manipulating components of the library, the feature set must be able to appropriately discriminate between library components. Hence:

A criterion for the usefulness of a task feature is that it is - or will be - named in the condition part of a modelling guideline.

The set of task features and modelling guidelines listed in this document must therefore be updated and maintained along with the addition of components in the library, and on the basis of modelling experience from development of applications.

3.4.1 Task features related to the task purpose

In *Common*KADS a task is characterized by a desired state of affairs in a case-model which is accessed through a number of dynamic knowledge roles. This state of affairs corresponds to a *goal* and is described in terms of *input* and *output* roles. The relation between tasks and goals is in principle many-to-many. The goal specifies the overall type of the task. Characteristics of the input and the output further specify the task within the overall category.

- **Goal**
  Within this category of task-features an inventory of goal-types is specified. It is a slightly modified and extended version of the taxonomy of problem types specified for KADS-1. Modifications have been made to more clearly separate task types from method types (e.g. heuristic classification is regarded as a method type, and has been removed from the list), and to distinguish between tasks that are either analytic or synthetic, and more complex tasks that often are combinations of the two (diagnosis, configuration). Extensions are additional combined tasks:

  - system-analysis
    - identification
      - classification
        - interpretation

- fault-finding
  - single-fault-finding
  - multiple-fault-finding
- assessment
- monitoring
- prediction
  - prediction of behaviour
  - prediction of values
- explanation
  - explanation of behaviour
  - explanation of values

- system-synthesis
  - design
    - creative design
    - complicated design
    - routine design
  - planning
  - modelling

- combined analysis-synthesis of systems
  - diagnosis
  - repair
  - configuration
  - maintenance
  - control

- **Input**
  Features of the task input are grouped into characterizations of the form (syntax), the time pattern (frequency and regularity) and the contents of the information entered:
  - content of input
  - form of input
  - time pattern of input
- **Output**
  Features of the task out are grouped similarly to the input, i.e. into the form (syntax), the time pattern (frequency and regularity) and the contents of the information produced by the system. + Note that other, and more detailed, characterizations of input/output are regarded as part of the task environment.
  - content of output
  - form of output
  - time pattern of output

3.4.2 Task features related to the task domain

Task features related to the task domain characterize the set of systems involved in the task, their type and degree of decomposability, the type of domain relationships describing system behaviour, etc.

Sub-categories of task features related to the domain[3]

- **General characteristics.**
  Characterizes the system that the task is about, i.e whether the type of system is a particular type of object, a process, etc., and whether its real world realization is physical or abstract. Examples of other characteristics are complexity and physical location.
  - System type
    Object type
    Process type
  - Abstract vs. physical system
  - Other characteristics
- **Knowledge types.**
  Types of knowledge needed, or normally used, to describe the system and to accomplish the task. Typically, several of these knowledge types are combined.
  - Common sense knowledge
  - First principles knowledge
  - Temporal knowledge
  - Spatial knowledge
  - Heuristic, associative knowledge
  - Structural knowledge
  - Component-subcomponent knowledge
  - Functional knowledge
  - Behavorial knowledge
  - State knowledge
  - Causal knowledge
  - Knowledge about normal system behaviour
  - Knowledge about abnormal system behaviour
- **Domain knowledge generality.**
  Whether general or specific knowledge is needed.
  - Generalized knowledge (general relationships, rules)
  - Specialized knowledge (previous cases, specific facts)
- **Domain knowledge strength.**
  Characterizes the system knowledge in terms of how theoretical strong the knowledge of the domain is, i.e. how certain the relationships between domain concepts are.
  - Strong domain theory
  - Weak domain theory
    Closed domain
    Open domain
    Deep relationships exists
    Only shallow relationships exist

[3] The categorization is partly based on results reported in [Steels, 1991] and [Aamodt, 1990].

- **Stability of domain**.
  Maturity of domain, and anticipated speed of significant changes in the domain knowledge content.
  - Stable domain
  - Rapidly evolving domain
  - Slowly evolving domain

To enable more precise descriptions, we will also provide a set of domain features that apply to a particular purpose type. For example:

Domain features specific for diagnosis tasks.

Domain features specific for design tasks.

Domain features specific for configuration tasks.

3.4.3 Task features related to the environment

In this category features of the task-environment are specified which in one way or the other influence the various agents' behavior. The problem solving environment is the organization in which the KBS system has to operate. It includes the user(s), safety requirements, costs and stability of the domain.

Examples of environmental features are: Environment temperature, access permission to data, presence of competing tasks.

Below are the abstract features of the problem solving environment.

- **Environmental conditions**.
  A characterization of the operating environment of the target system.
  - Physical conditions at the site of operations
  - Physical interconnections to other systems
  - Accessibility of data
  - Reliability of data source
  - Time restrictions
- **Safety**.
  Characterizes the environmental risks of gathering data, or of realizing solutions.
- **Costs**.
  Costs related to gathering of data and realization of solutions. Cost types may be financial, man-power, or time. enditemize
- **User**.
  Refers to the 'non-agent' role of the user, and captures issues of information presentation, access restrictions, etc.
  - Presentation style
    - Screen interface type

      Printed report types
  - Type of user explanations
  - User competence level
  - Authority distribution

3.4.4 Task features related to grounding

Grounding features contain two sub-categories of features: features related to *agents*, and *resource features*.

- **Agents.**
  Agents realize two categories of grounding relations. Domain grounding concerns the relation between the systems in a task domain and their models as constructed by the agent(s) involved in the task. In case of the realization of the domain grounding the agent's reliability refers to the agent's reliability in performing an observation and reporting about it. In case of the realization of the inference grounding, agent reliability refers to the agent's reliability in performing certain types of inferences. A single physical agent can play a role in several grounding relations[4]. Inference grounding relates abstract inference in models (e.g. causal propagation, evidence combination) to the actual inference in a physical system (computer or human). An agent realizing the inference grounding of the task model, has the capacity to reason about or even accomplish the task. The computer(s) on which a KBS is running is always an agent involved in inference grounding. Computational features of the task reality are primarily related to inference grounding. Inference grounding as a category contains two major sub-categories of features.

  – Domain grounding
    - Acquisition of data

      Grounding relation to data source
      - Costs of observations

    Performing actions

  – Inference grounding
    - General computational limitations of agent

    Computational realization of inference in models

- **Resources.**
  In this category we specify what resources are available to the agents within the given task-distribution and task-environment, as well as how these resources are consumed and measured. Not all cost information is, however, a resource. For example when the task goal is to construct a device that accomplishes a certain function and costs less than a certain amount of money then this information is part of the task domain. Costs in that case must be explicitly reasoned about, but are in no sense consumed by the problem solving process. A similar example would be planning a route through town that gets you to a place in limited time. Here time information is part of the task domain, but in no way consumed by the problem solving process.

  With respect to the Expertise model:

  – Time resources of KBS agent
  – Cost of operations of KBS agent

[4] As we are concerned with the development of KBS in *Common*KADS and not with distributed AI, the model of expertise will in practice become the most elaborated agent model. The other agent models will be specified as far as information about other agents is required for KBS development. Decisions about the extent to which the various agent models will be specified in a KBS development project, must be taken either in advance during project configuration, or in the course of the project. In practice some guidelines will be required, specifying the conditions which suggest the need of agent models to be elaborated in full. The reason for this is pragmatic: before analysis starts, an estimation will have to made of the required development effort. Full specification of additional models is always a labour-intensive activity and will have a strong impact on these estimations. Features which may support this method configuration, are outside the scope of agent features as they are meant here.

(also introduce model grounding versus inference grounding and illustrate with figure and example)

3.5 Focusing the identification of task features

In order to focus the process of task feature analysis and identification, a method is provided that enables us to for discriminate between sub-types of the four main categories, according to their significance - or importance - within a certain modelling context. We do not define a fixed scheme here, but rather allow for a user-defined preference ordering of feature types, which can be used to guide the feature identification process.

Abstract task features may be classified - within each of the four main categories - as belonging to one of the following three *preference types*:

- Primary task features.
  A particular sub-category within each main category is selected as the primary one, based on its relative importance. An example is goal type, for the Purpose category.
- Secondary, specific task features.
  Collection of features which are specific to (make sense for, apply to) only one of the primary feature classes. This enables a focus on the most discriminatory features. An example is an input type which only is relevant for a particular goal type - e.g. an input type such as the measurement of a particular finding related to diagnosis.
- Secondary, general task features.
  Collection of features which are neither primary types, nor specific to one of their categories. An example is an input type which is relevant to several task goals.

The grouping into preference types enables a focused control over the task feature identification process. For example, it provides a means to control the questioning of the user in a natural, meaningful way. If such an ordering is used, there must be a pointer from a secondary feature to the primary features it makes sense for.

Below, the abstract task features are presented. The current set of task features - together with the user prompts which identify their values - are enumerated in chapter 7.

3.6 A mechanism for task feature identification and use

In this section we discuss the more practical aspects of how to use the task features, within the general framework, to identify library components. A simple method is described, which illustrates how an inventory of task features can be used in a focused way in expertise modelling.

Task features define the condition part of a modelling guideline, and this is what they are used for. So, what we need is a method for *identifying the relevant task features*, i.e. to describe the application in terms of (abstract) task features, and a practical means to *propagate the feature-value pairs* along the appropriate paths to library components. Such a path is defined by the linked structure of modelling support entities (figure 2.3). An identification of some task features may have been done in developing the Task Model of the model set, or by another kind of rational task decomposition. This provides valuable input to the identification of task features as classified here.

Figure 3.2 illustrates how task features are identified for the purpose of finding appropriate modelling guidelines. The task feature inventory is the pool from which task feature *names* are selected. To each task feature name, one or more questions are associated. The questions prompts the user for the task feature *value*. This value has to conform with the value constraints specified in the task feature definition. The task feature value (or, actually, the pair of task feature name and task feature value) is then matched with all modelling guidelines, of which a matching subset then gets defined.

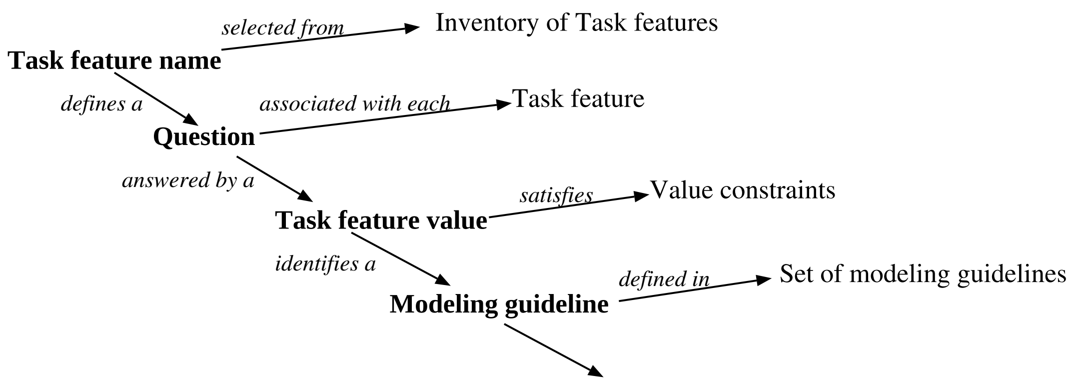

Figure 3.2: Identifying and using task feature values. *The figure illustrates the steps (downward-pointing arrows) taken in order to instantiate a task feature value, which in turn is referred to by a modelling guideline. The slightly upwards-pointing arrows shows where the entities written in bold come from.*

How, then, should the process indicated in the figure be controlled? In principle, there are two different approaches to identifying task features: A data-driven approach, which is to start from the task features, and select the set of modelling guidelines that match, and a goal-driven approach, which starts from the modelling guidelines and then tries to identify the values of task features named in the conditions part of the guidelines. Now matter which of these we choose (or whether they are combined), a strategy for question-asking is needed. The simplest way is, of course, to sequentially question the user about each task feature value. Another approach is to always make the next question - or group of questions - in some way dependent on already determined task feature values. The latter approach would require a preset (partial) grouping of feature types - or modelling operator types. Below, we show how the proposed structure of abstract task features facilitates a grouping of questions within each category according to the three preference types for task features[5].

As described in the previous section, within each task feature category, features may be typed according to whether they are *primary types*, *secondary specific types*, and *secondary general types*. To focus the task identification process, the questioning process may be controlled accordingly. That is, first, questions associated with the primary feature sub-type (within a top-level feature category) are asked. Of the remaining sub-types, the specific features are identified first, followed by the general ones. This scheme combines a focusing according to main characteristic types and most discriminatory additional features, with a subsequent adding (if necessary) of less precise but still useful features types[6].

A way to utilize this preferencing is as follows:

- Identify primary features of Purpose, Domain, Environment and Grounding (typically in that order).
- Identify secondary specific features within each of the categories. This may be done either as a separate step, or intermixed with the previous step.
- If more modelling guidelines are wanted, identify the secondary general features. This may also be done after a first attempt has been made to identify useful modelling components in the library.

[5] This scheme applies mainly to the data-driven approach. For a goal driven approach a similar preferencing of modelling guidelines (i.e. type of operations and/or library component types) can be made.

[6] How this process interacts with the utilization of library items in the expertise modelling process still has to be detailed.

Chapter 4

Modelling Guidelines

4.1 Introduction

The aim of exploring task features is to provide for a way to formulate and utilize associations between features of the task reality and modelling components of the CommonKads library. The initial approach was to study possible pointers from abstractions of task features - as defined in the abstract task feature space - and directly to library components. These pointers were named modelling guidelines. Based on further analysis of the problem, including an example application exercise (the hospital meal planner) and study of related research and experience reported in the literature, this initial platform has been extended and refined.

It turned out that the initial notion of a modelling guideline was too coarse grained. For example, it did not capture the distinction between the actual properties of a library component as such, and its potential use in the modelling process. Further, the notion of a modelling guideline was also too narrow, since its only input was a set of task features, or task feature classes. Input to the selection of a library component from the current state of the Expertise Model, as well as from states of other models of the model set, was not explicitly taken into account.

This has lead to that the notion of modelling guideline has become more specific, and that, in addition, the *modelling operator* has been more clearly specified. The major extension has been that the role of abstract task features are not treated in isolation, but related to other indicators for library access and use.

As described in Chapter 2, the associations represented by modelling guidelines connect task features to library components via modelling operators. Note that the modelling guidelines provide a focus and are not prescriptive in nature. They provide support knowledge for modelling a task, but do not prescribe how a task model should be constructed in the first place[1]. Examples of different approaches to arrive at a model of a task are:

- Model assembly from model components
- Selection of total conceptual models
- Knowledge differentiation and refinement of a general model

Figure 4.1 extends the steps for identifying task features, described in Section 3.6, with an illustration of how modelling guidelines link to modelling operators.

The modelling guidelines can be exploited in a bi-directional way. In the figure only the 'data-driven' (task feature driven) way is illustrated. A 'goal-driven' way would be to select modelling

[1] Similarly, the *Common*KADS library also has structures and contents to support various modelling approaches [Aamodt et al., 1992].

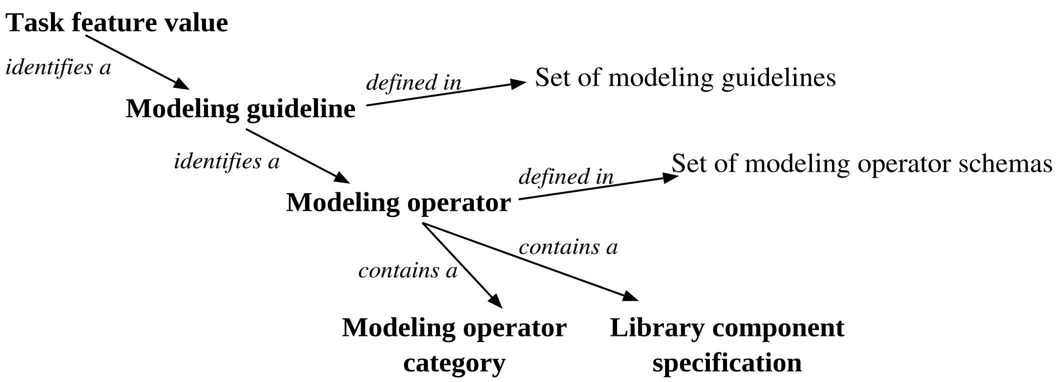

Figure 4.1: Modelling guidelines and modelling operators. *The figure illustrates the steps (downward-pointing arrows) taken in order to instantiate a task feature value within a modelling guideline. The slightly upwards-pointing arrows show where the entities written in bold come from.*

guidelines on the basis of what modelling goal they support. This is exemplified in the next subsection.

A modelling guideline thus on the one hand provides one with a heuristic focus on the space of library components and on the other hand provides one with a heuristic focus on the task reality. If one for example follows the selection approach to modelling, the modelling guidelines will suggest a selection operator applied to a particular type of library components - in this case a generic model with all knowledge types filled in. After the initial selection of a generic model, the choice will have to be confirmed with respect to details of the model's expected behavior in the task-reality. These expectations are derived by traversing the modelling guidelines back to task features. In this way, the modelling guidelines provides a focus on features of the task reality given the features of the selected model components.

4.2 Structuring the set of modelling guidelines

Unlike the abstract task features, modelling guidelines do not need to be explicitly structured in a predefined hierarchy. However, it is useful to view the modelling guidelines as defined, implicitly, by the modelling operator hierarchy. This means a grouping according to type of modelling operation (i.e. select, combine, etc.) which the guideline support. This is illustrated in Figure 2.6, Section 2.3[2].

An example of how modelling guidelines can be typed via the modelling operator hierarchy is as follows. This typing can be used to for a *goal-driven task feature identification*, starting from the modelling guidelines. For example, a strategy could be to start asking for task features referred to by the selection guidelines, at some later moment to identify features for method selection, etc.

- Selection guideline
  - Task selection
  - Method selection
  - Domain model selection
  - Inference structure selection

[2]It may later turn out useful to impose a second type hierarchy - that of library components, as well. This depends on what control strategy (manual or computerized) we eventually will implement - an issue which needs further investigation, and which should be related to the tool discussion.

- Combination guideline
  - Method to achieve task
  - Domain model needed for method
- Specialization guideline
  - Concretion
  - Composition
  - Instantiation
- Generalization guideline
  - Abstraction
  - Decomposition

The subtypes may be further specified (type of task, type of domain model, etc), corresponding to the level of specialization of library components. The typing of modelling guidelines via modelling operators facilitates maintenance of the set of guidelines, making it easier to extend and revise the sets. Further, it facilitates the use of these modelling support entities, since it enables a user - or a modelling tool - to inspect the hierarchies and explore more general or more specific entities. This may, for example, be used in a strategy for gradual refinement of task features, by initially specifying only high-level abstract features, leading to only very general library components. The modelling operators triggered in this process is, correspondingly, likely to be of a rather general type. By inspecting the type hierarchy, identifying the types operators which has been triggered, more specific operators - corresponding to more specific guidelines - may be inspected and activated in order to identify more specific task features.

By opening up for the explicit usage of type hierarchies of the modelling support entities, we provide the entire task feature framework with more flexibility. Hence, it can be easier adapted to a user's needs and the modelling strategy he wants to use.

Chapter 5

The Model Selection Tool

5.1 Introduction

The Model Advisor Tool is a research vehicle that has been used by the I.5 task to experiment with the current set of task features and modelling guidelines. Note that the tool is officially called "Model Selection Tool". This chapter contains the specification of the tool and gives an impression of how it has been designed, how it works, and in which way a knowledge engineer can use it.

5.1.1 Problem statement

The purpose of the tool is to support the knowledge engineer in the selection of problem solving components for real world tasks. The tool should guide the user through the selection process by setting up a dialogue in abstract terms about the problem domain. The system should indicate to the user when it has sufficient knowledge to do a proposal for the selection. This proposal is an advice to the user together with an explanation and motivation for this this advice. The implementation of the tool should be integrated in the *Common*Kads workbench.

5.1.2 Preparatory study

First we investigated two earlier built tools that related to the Model Advisor Tool.These two tools are the Model Selection Tool [Belis, 1992] and the Task Model Tool [?]. A short description of these tools follows after this paragraph.

5.1.2.1 Short evaluation of two existing tools

This part contains a evaluation of the "Task Model Tool" and the "Model Selection Tool". The focus of the evaluation is on those parts which have influenced the design of the Model Advisor Tool.

Task Model Tool This tool[1] is implemented in Shelley. The tool has the appearance a drawing program. Note that this is not as silly as it sounds like, because experience has pointed out that human experts can work very well when they have a software package in which they can make visual representations on which they can reason about. Within the tool is it possible to represent the task with a task decomposition and its associated task features. The links between the tasks and the features can be represented and also a categorization of the different task features in the *Common*Kads abstract task feature categories is possible. To come to a conclusion about this tool

[1] It must be said explicitly that the tool is the first KADS-II tool and that it should serve as an example of an operational specification.

we note that especially the interface part is very useful as a memory and representation support for the domain expert.

Model Selection Tool The Model Selection Tool is written in Kan. Kan is a software tool for developing rule based expert systems [Caekenberghe et al., 1991]. This tool contains a reasoning part but no visual support for the knowledge engineer. It is also very model and method directed. The problem with such an approach is that the risk of selecting a wrong model or method is very high, because the system starts asking the user from the start if particular methods or models are true for the real world task. In this way the knowledge engineer would add features to the task that are actually not so important. The result of this is that the knowledge engineer never had the chance of describing the task as he wants it and adding the features that he things that are important. Now the system describes the task and will use this description to select a model and method. Also the interface is poor and adding methods and models requires explicit changes in the code. There is also something positive about the tool and that is that the idea of having an active reasoner. The fact that the system is actually doing some active reasoning can lead to an enormous support for the knowledge engineer in his analysis. Especially when the knowledge engineer gets stuck at some point in the modelling phase, then the system will guide him through all the generic models and put his attention back on the more important and maybe most applicable generic models for that real world task.

Reflections of these two approaches for the Model Advisor Tool From the Task Model Tool we have taken the user friendly interface and the strong representation ability as an inspiration source.The idea of an active reasoning part comes from the Model Selection Tool.

5.2 Specification

MAT is a tool which assists the knowledge engineer in the development of a model of a real world task. The assistance takes different forms:

- There is a user friendly interface which enables the knowledge engineer to visualize the real world task.
- It offers the possibility to specify and describe this real world task. This can be done by decomposing the task, adding descriptions and linking task features to the task.
- It provides a clear and handy frame where the generic model library is visualized; so that the knowledge engineer can get an overview of all the possible generic models that the tool is aware of. Note that the knowledge engineer can change this library by adding or deleting generic models.
- It also enables the knowledge engineer to see the relations between the different tasks, methods and models in the library. He can also make new relations or delete old ones. In this way the knowledge engineer can lay a new structure on the library so that the library becomes more usable for a particular real world task. This new structure may help the knowledge engineer in reasoning about the task he has to model.
- It gives the knowledge engineer a better understanding of the different generic models that are available in the library. This support is given by perceptible links between the generic models and the features which may trigger these models and also by using "hypercards", which contain descriptions of the different generic models and features.
- There is an active reasoning part that not only waits for input data in the form of task features, but that asks the knowledge engineer if particular task features are true. If a

certain task feature is true the tool gives the knowledge engineer the opportunity to add that task feature to the real world task.

- The active reasoner gives the knowledge engineer advice about which model to choose. This advice is given in two ways :

  1. Implicitly: the tool works on the generic model in a particular order. This order depends on the possible applicability of the generic models. In this way the attention of the knowledge engineer is focused on a set of models which are possible applicable for that specific task. With possible applicability is meant that they have a higher probability of applicability for the real world task with the knowledge that is already available in the system. This is because the tool has not yet all the knowledge that it needs to model because not all the task features of the real world task are filled in. So it proposes the models that map the best at that moment.
  2. Explicitly: when the knowledge engineer has worked out a particular model, the tool will give a summary of how that model is related to the real world task. So it gives the knowledge engineer an overview of the state of that particular model in that particular task. This gives the knowledge engineer extra support in deciding whether or not he should use this generic model.

     At the end of each modelling session, an overview of all the relevant generic models is given in order of their applicability. Note that this is very hard to do because there is not a good evaluation theory yet. I have implemented a rather general one so that this concluding list can not be taken very serious at the moment. However, if a good evaluation technique becomes available, then this would be easy to embed it in the system because this evaluation part is separated in a modular way from the rest of the program.

- It gives the knowledge engineer the freedom to work on every part he wants at any time. So it is possible for the knowledge engineer to work on a problem solving method; then go back to the real world task and do some work there; and then reason about a task at the knowledge level or to work out a domain model. This mixed interaction process of working and reasoning on the real world task as well as on the generic models at the same time, is one of the major advantages of the tool.

5.2.1 Problem Definition

> The problem is to find out which generic models are applicable for a certain real world task.

The tool has to come up with a set of selected tasks, domain models, and problem solving methods, which are appropriate models of that particular real world task. The selected tasks, models and methods come from a library of generic models. The selection is done by making use of task features. This means that the real world task as well as the generic models have to be described by features.

5.2.2 Requirements

The input of the Model Advisor Tool is a real world task and an incomplete description of this task by task features.

The output of the Model Advisor Tool is:

- a set of generic models in order of applicability;
- an advice about the applicability of the selected models;

- an elaborated task feature description of the real world task.

5.2.2.1 Task analysis

The task decomposition is shown in figure 5.1.

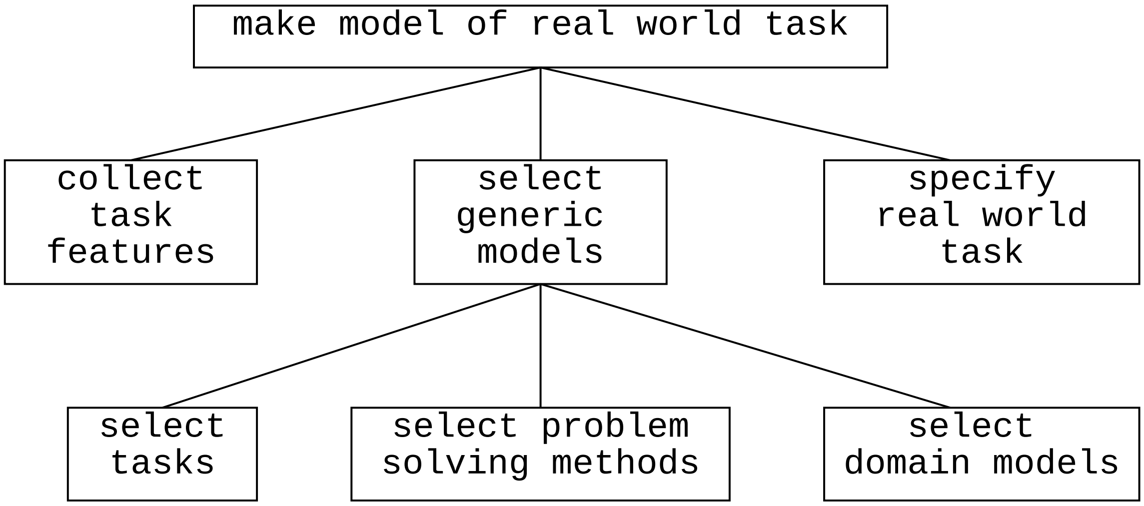

Figure 5.1: Task decomposition of MAT.

The top task of the Model Advisor Tool is to make a knowledge level model of the real world task. This task can be decomposed into several sub-tasks.

- Collect task features Input: Task features, applicable generic models with their features, real world task. Output: More relevant task features. Description: Add task features to the real world task.

- Select generic model Input: Real world task, task features, generic models. Output: List of applicable generic models. Description: Select models from the library.

- Select task. Input: Real world task, knowledge level task with the corresponding input/output relations. Output: List of applicable tasks. Description: Makes a list of applicable tasks.

- Select problem solving method. Input: Real world task, task features, problem solving methods, method features. Output: Advise of problem solving methods and a list of applicable methods and new task features. Description: Investigates the applicability of a problem solving method, makes a list of them in order of applicability and adds task features to the real world task.

- Select domain model Input: Real world task, task features, domain models, model features. Output: Advise of domain models and a list of applicable domain models and new task features. Description: Investigates the applicability of the domain models, makes a list of them in order of applicability and adds task features to the real world task.

- Specify real world task Input: Real world task, task features Output: Association between real world task and task features. Description: Adds task features to the real world tasks.

5.3 Description of the Model Advisor Tool.

The Model Advisor Tool consists of six sub windows (5.5). The right one is the workbench (a desktop in Shelley) on which the knowledge engineer can manipulate the representation of real world tasks. On the left side of the workbench you can find the model window and below this one the model feature window. To the left of the model window is the method window and beneath this is the method feature window. Completely at the left is the task window. The last five windows are browsers. For each real world task or sub task that is situated in the workbench, is it possible to start a reasoner. This reasoner will try to find the appropriate generic models.This search shall be done by setting up a dialog in abstract terms with the knowledge engineer.

5.3.0.2 Workbench

This is a desktop where the task can be identified by the knowledge engineer. Here he can represent and manipulate task decompositions and add the tasks features. It is also the place where the descriptions of all the objects are displayed. This is the place where most of the actual work is done.

Task Decomposition In many cases the real world task is too complex to work with directly. Therefore the task is decomposed into sub-tasks. A technique such as rational task analysis is often used to achieve such a decomposition. The decomposition takes the form of a tree, with the real world task at the root. Each of the tasks in the tree (i.e. nodes of the tree) can be investigated more closely by double clicking on it. Actually the node in the tree is an icon of the task. It is called the task node. This is similar to an previous built tool by Manfred Aben [?].

Task Node Each task node represents one task. This task must be described by task features. The task feature set, i.e. all the task features that are related to the task, should not not necessarily be completed at the beginning of a session. The system will work out some new task features that are relevant for the selection of a model or method.

Task Features Each task feature is related with a task. The task feature has a name and a category that has to be one of the abstract task feature categories as identified in the first part of this document. In the current version the knowledge engineer is responsible for choosing the right name for the task feature and he must also classify it under the correct abstract task feature category[2].

5.3.0.3 Browsers

This part of the Model Advisor Tool is the generic model library. The browsers serve as a solution space, and are a memory support for the user. Here the knowledge engineer can see the specifications of the generic models. Also the relations with their features and other generic models become clear in this part of the tool. The knowledge engineer can feel free to change this library part whenever he likes to do that. In that way he can make the library more specific for the domain he is working on.

Tasks A knowledge level library task is specified by its input and output relations. This specification can be done with questions. These questions are inserted by the knowledge engineer, together with the introduction of a new task. It is with the answers on these questions that the system will select which task is applicable for the real world task.

[2] On a later version of the model advisor tool there will be some sort of support to help the knowledge engineer classifying the task features. The slot is already available under the popup menu but it isn't implemented yet.

Methods and Method Features The methods and method features can be created separately from each other. The method features can be assigned to the methods so that the method can be fully described by these method features. It is with these features that the reasoning process will do the selection of a method.

Models and Model Features This is the same as with the methods and method features. Here also the reasoning process for the selection of a good model makes use of the model features. Be aware of the fact that the selection not only uses the model or method features, but that it also takes into account the already established knowledge. For instance the fact that the task is already known may help the selection process.

5.3.0.4 Reasoner

The reasoner can be started up by the knowledge engineer in the real world task visualizer (that is in the work bench). The reasoner will open three windows: method reasoner, model reasoner, and task reasoner. Before opening these windows, a first part of the reasoning process takes place. A first proposal of generic models is done. This is accomplished by using all the knowledge that is yet available. The general idea behind the process is as follows:

1. For each task feature of the real world task that maps with a task feature in the library, is been investigated which domain model and problem solving method it activates.

2. For each of the activated generic models is been counted how many of their features map with these of the real world task. Each generic model gets a priority that is equal to that number.

3. Increase the priority of each activated problem solving method with the number of domain models that are related to that method and at the same time are activated by the task features of the real world task. It also increases the priority of each activated domain model with the number of problem solving methods that are related to that model and at the same time are activated by the task features of the real world task.

4. Each task in the library receives a priority that is equal to the sum of the priorities of the domain models and problem solving methods, that it is related with.

Now we have three priority queues, one for domain models, one for problem solving methods, and one for the tasks. The first element in each priority queue is the generic model which has the largest probability of applicability for the real world task. These queues are input for the three separated reasoners as described below.

Method Reasoner The method reasoner will provide an interaction window in which the system tells the user its results so far. These results come from the reasoning process as described above. Then it asks the knowledge engineer some extra questions about the task that it is investigating. The questions can be answered with yes, no, or unknown. The model advisor always keeps the user informed about the things it is doing, and when it arrives at a solution it provides a short summary of the established facts which the knowledge engineer can use as an advise. When the system discovers some new facts about the real world task, it will pop up a prompter box which contains the name of the new task feature of the real world task. The knowledge engineer can now classify this feature in one of the abstract task feature categories and click the create button. Then the system will link the new task feature to the real world task and adds this knowledge to the knowledge base. When the knowledge engineer wants to stop the reasoning or when the system can't find any more relevant questions, it makes a new proposal for applicable problem solving methods. This proposal is established as follows: First the relevance of each feature is determined. This is been done by the following simple rule: "The sum of the relevances of all the method features for a particular method is equal for all problem solving methods, and for all method features of a particular problem solving method are the relevances equal to each other." Next each problem solving method gets a new priority based on the following:

- For each correct feature, the priority of the problem solving method is increased by the relevance of that method feature
- For each false feature, the priority of the problem solving method is decreased by two times the relevance of that method feature.
- For each method feature that is classified as "unknown", the priority of the problem solving method is left unchanged.

Finally the system proposes the methods to the knowledge engineer in order of assumed applicability, that is, the one with the highest priority comes first.

Model Reasoner This is based on the same principles as the method reasoner.

Task Reasoner The task reasoner displays its results in an interactive window. It is based on the same principle as in the method reasoner, but in this case the system starts asking the knowledge engineer questions about the applicability of the task. These questions are based on the input/output relations of these tasks. The relevance of each question is measured in the same way as it is been done in the method reasoner. But the priority of each task is derived a little differently; the priority is equal to the number of positively answered questions multiplied by their relevance. At the end of the session, that is when the knowledge engineer wants to stop or when there are no more relevant questions to ask, the system makes an overview of the knowledge it has gathered and displays a set of tasks in order of applicability.

5.4 Working with the tool.

The best away of getting an impression of the functionality of the tool is to use it. You will see that the interface is very user friendly and that all the functions are also very clear and easy to use. Nevertheless we present a worked out problem here. We will go step by step through the selection process. The example that we have worked out is the problem of making a knowledge level model of the problem solving behavior of the diagnosis of a printed circuit board (PCB) of an electronic device.

First the tool is started up by the knowledge engineer. Notice that the tool can come in two versions. One that is complete empty, and another one that already contains a library with generic models and relations between those models. Figure 5.2 shows the start of MAT with an instantiated library. Also the indexing features are present in this second version. The knowledge engineer can always change this library. This change can be: defining new generic models, deleting old relations and creating new ones, and also introducing new indexing features is one of the possibilities. In this way the knowledge engineer can adapt the library to the problem he wants to solve. If this is done the tool can come up with better usable results because it can be less general than it initially was. The abstract task feature classes are always present in the tool. These can also be extended or deleted by the knowledge engineer. Notice that the knowledge engineer can always save the tool in its current state. This can be done by simply clicking on the save button.

Now the knowledge engineer starts making a rational task decomposition of the task he has to perform. The visualisation of this decomposition can be represented in the rightmost window (i.e. the Shelley desktop, or workbench). Each task icon opens when it is double clicked (or open in the popup menu). In each open icon the task can be further specified. This specification can be done with adding task features to the task. Note that these task features have to be relevant

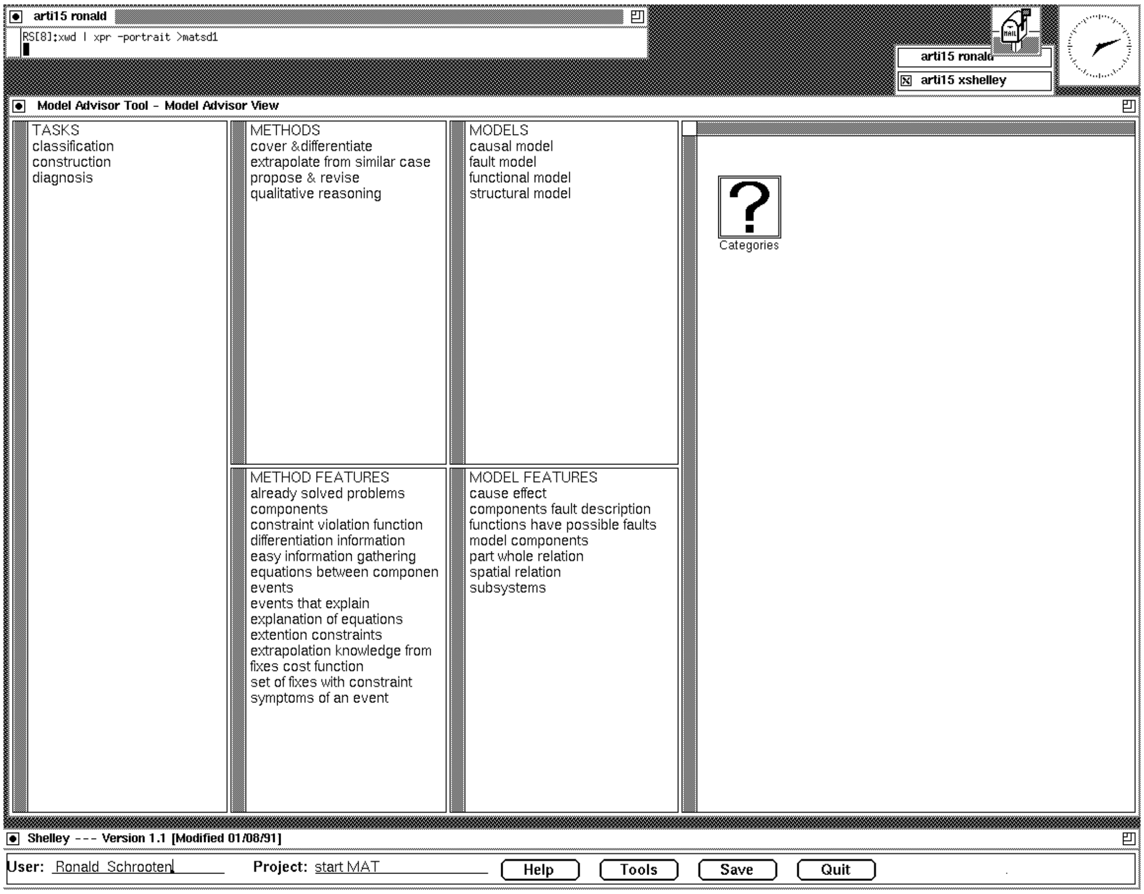

Figure 5.2: The start of MAT with an instantiated library.

for the selection process. (If they are not relevant the system can't and will not do anything with them and shall only pick out those who are relevant for the selection process.) The category tree can be used to support the search for relevant features. Also the fact that the knowledge engineer can look at the indexing features and the relations with their models or methods can help the knowledge engineer in identifying the relevant task features. If the knowledge engineer thinks that he has pointed out the most important features or if he does not know any new features anymore, then he can start the reasoner (see 5.4).

The active reasoner is started up by choosing "Start Reasoner" from the popup menu. After starting the reasoner we see three windows pop up (i.e. Method reasoner, Model reasoner and Task reasoner. Now the reasoner does its first proposal about the possible applicable generic models. This is done by looking at the already available task features and then mapping them to the indexing features. Next all the methods and models that are activated by these indexing features are investigated. There is also an order on the proposed generic models. This order in created by looking at the number of the indexing features that match with the task features of the real world task. Also the internal library relations are used to generate this list. With this list of proposed generic models the attention of the knowledge engineer is focused on only a few models. Thus the tool supports the knowledge engineer in selecting in this particular case the most important generic models from those who are irrelevant for the task. The reasoner starts know asking extra information about the generic models that it is going to investigate. These answers can be answered with yes, no or unknown. In this case we can see that the system proposes

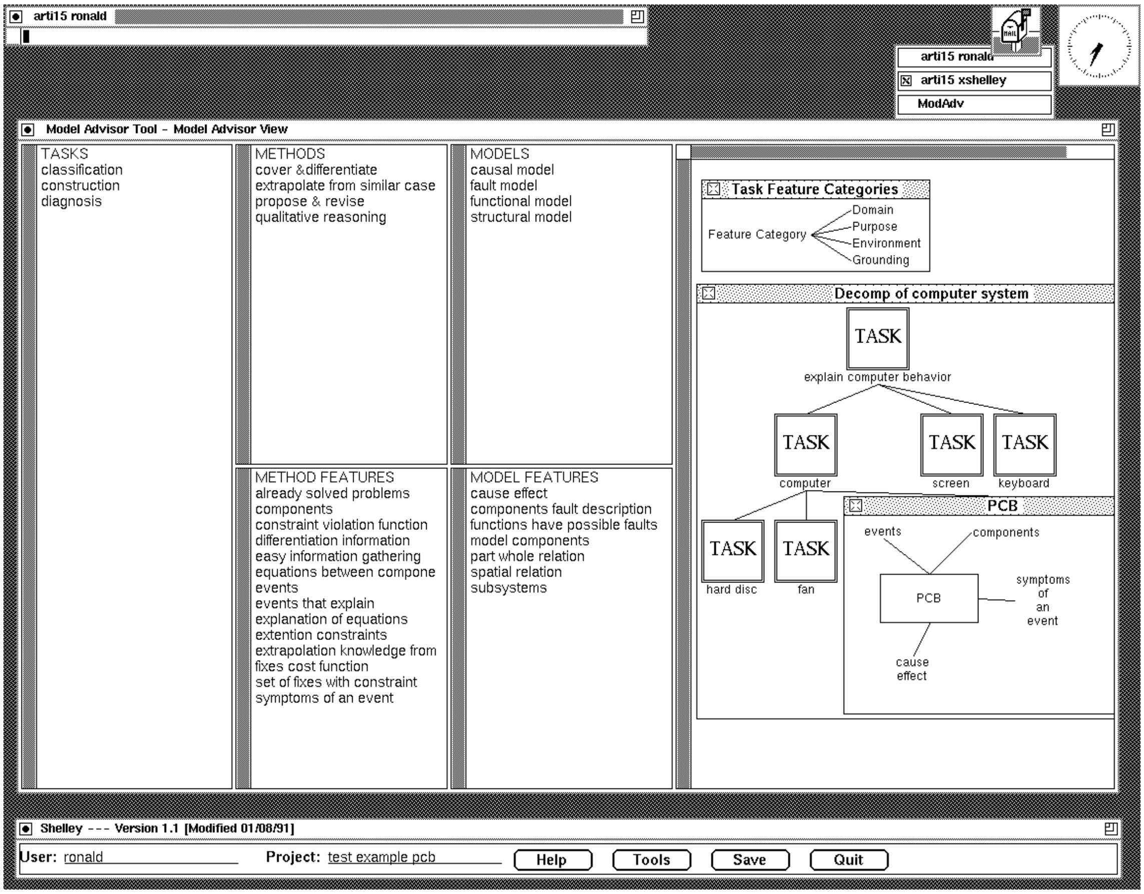

Figure 5.3: MAT with an instantiated library, a task decomposition of the task and a partially description of the PCB with task features.

the problem solving methods: "Cover & Differentiate", "Qualitative reasoning" and "Propose & Revise". The proposed domain model is "Causal model" and the proposed tasks are "Diagnosis" and "Construction" (see figure 5.4).

In figure 5.5 we are going to investigate the problem solving method into more detail. The knowledge engineer can answer the questions and gets after investigation of each problem solving method an advice in abstract terms about the applicability of this method. Also some remarks are made by the system about the use of this problem solving method If the system covers new features during this interaction with the knowledge engineer, it will popup a prompter box with the name of the new feature that it wants to add to the real world task. It always asks the knowledge engineer for permission to do this association of a new feature. Thus the knowledge engineer can now add this new fact to the knowledge base of the system by simply clicking the "create" button. Note that he first have to fill in the abstract task feature category to which the new task feature belongs.

Now the same is done for the domain models and the task.At the end of the reasoning process (i.e. when there are no more questions that the system can ask or when the knowledge engineer wants to stop) the system makes a new list of generic models. This list reports the generic models in order of applicability (see figure 5.6). This list can the be taken as an advice about which generic model to choose for this real world task. Note that the knowledge engineer can stop the reasoning process any time he wants. It is also possible to go back and forwards to the library/workbench

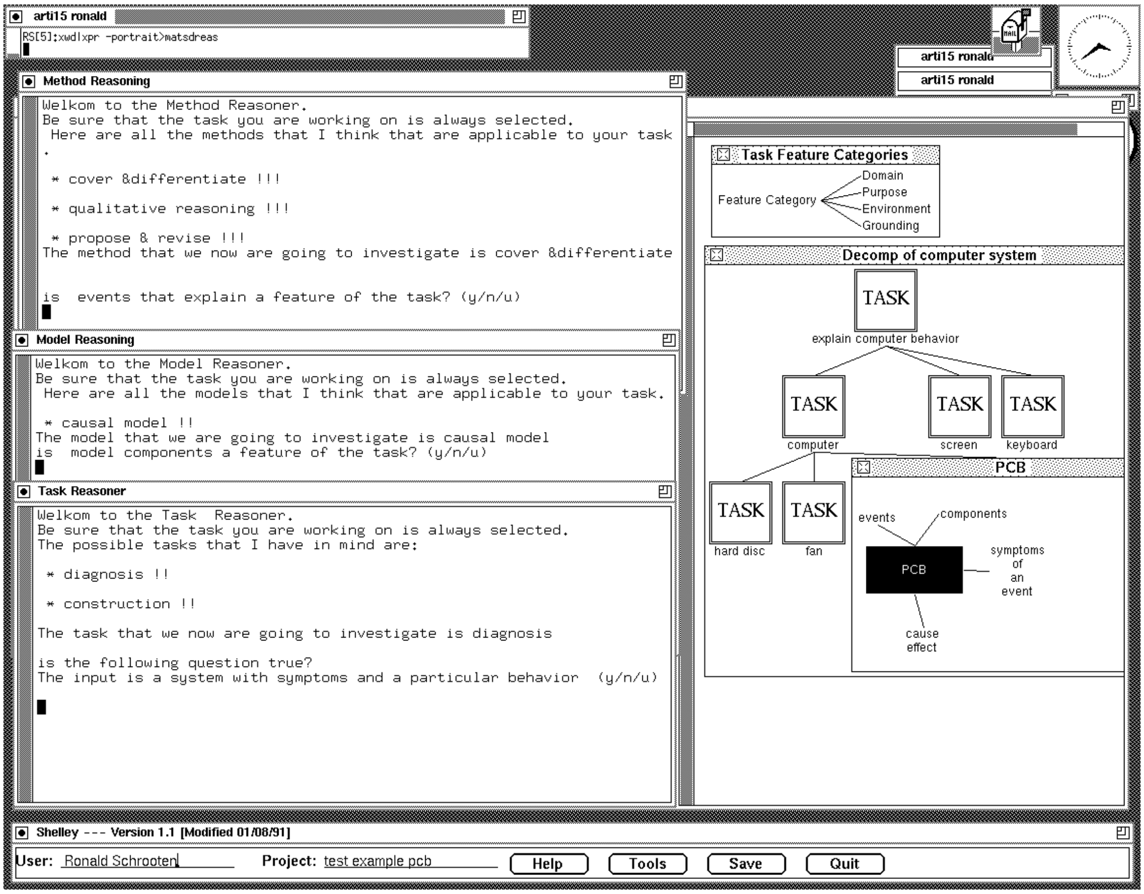

Figure 5.4: Mat after starting up the reasoner.

and the reasoning windows. Skipping between the reasoning windows is also no problem.

5.4.1 Knowledge level analysis of the tool

This example is a little bit a strange. Here we are going to analyze the model advisor tool by itself. Thus we are going to investigate which generic models are suitable as a knowledge level description of the task that the model advisor tool performs.

First we can decompose our task into two sub-tasks: "Determine abstract task features" and "Select generic models". Since the library is not well enough defined (i.e. not specific enough, not enough details) for these tasks, we will investigate the global task. This one is a little bit more general and since the library also contains the more general knowledge level components it is more suitable for handling this problem. Initially, we can assign the following feature to the task.

- Components: We can translate component to task feature. Thus the task features are the different components of the task, because the whole task is described with task features.

- Differentiation information: We can differentiate our solutions by the relations they have with their components (i.e. their indexing features that map with the task features).

- Symptom of an event: We can translate symptom of an event to indexing feature of the solution class, because an event can be seen as a class [Duursma, 1992b] and an indexing feature as a symptom that indicates this class.

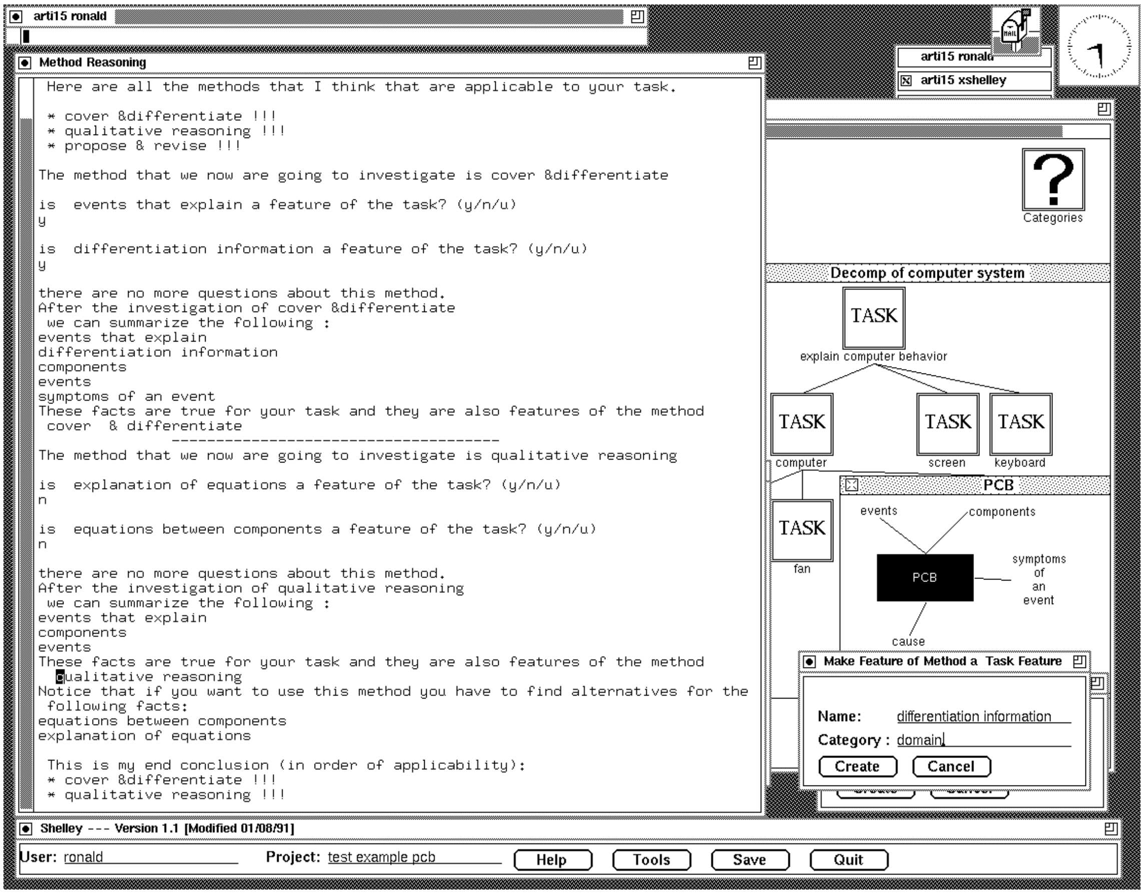

Figure 5.5: MAT with an instantiated library, a task decomposition of the task and a partially description of the PCB with task features.

- General class: All the generic models in the library can be seen as general classes.

Figure 5.7 shows the tools at this stages; the task features have been assigned and the reasoner has been invoked.

Next we start up the reasoner. The system proposes cover and differentiate, propose and revise, and qualitative reasoning as possible problem solving methods. It proposes a classificatory model as domain model and diagnosis, classification and construction as possible tasks. Now the system starts a dialogue with the user about these models. After the user has answered the questions posed by the system and the user has added some new relevant task features (proposed by the system) to the real world task, the system comes to an end conclusion (see figure 5.8).

Here we can see that the problem solving method cover and differentiate, the domain model classificatory model and a classification task are the knowledge level components that the system considers the best.

5.4.1.1 Analysis of the solution

An investigation of the provided solution is always recommended since the solution is only an advise of the system and should be interpreted as a possible solution that is worthwhile to investigate further.

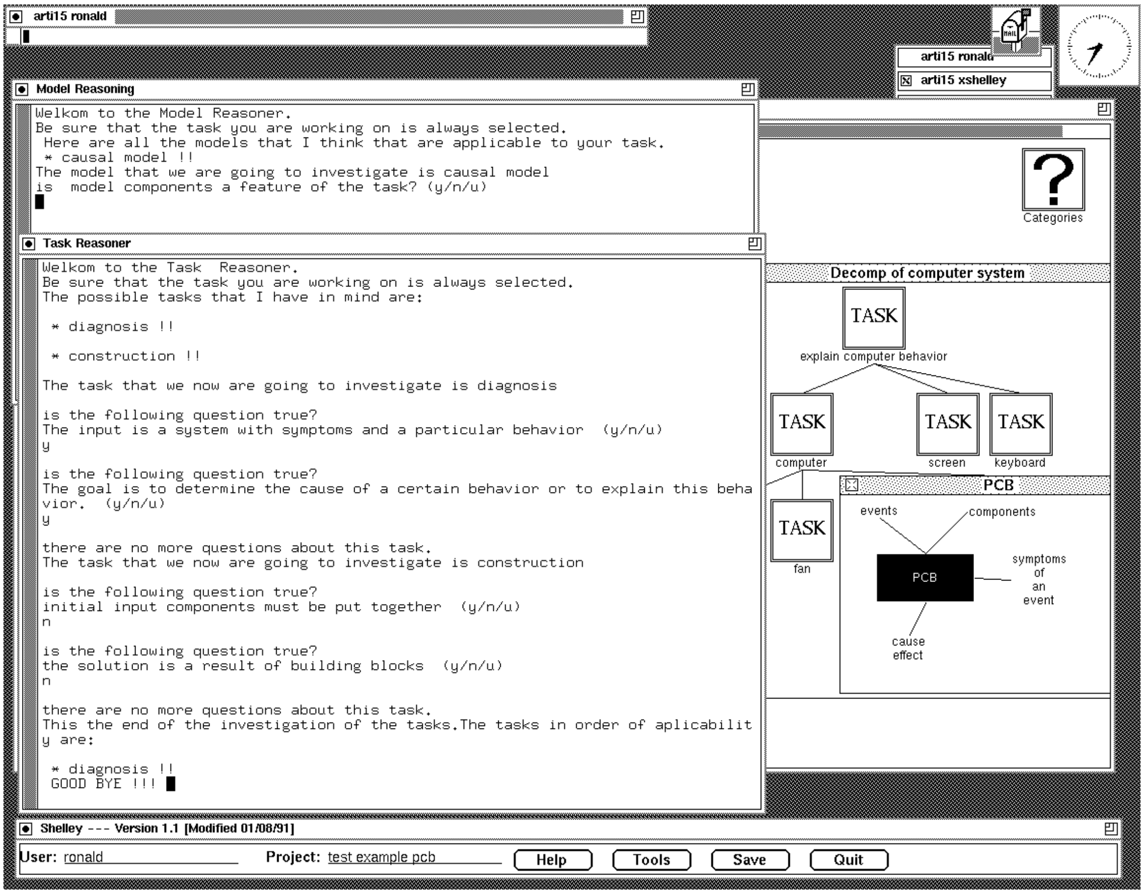

Figure 5.6: MAT is focused on the task and supports the knowledge engineer in reasoning about the applicability of this task.

- Cover & Differentiate: Translation of the original terminology of cover & differentiate to the domain specific terminology leads to the following problem solving strategy.

  1. Cover all the generic models that can possibly lead to a solution. This is done by taking the task features and determine which generic models the indicate.
  2. If there is more then one generic model then try to differentiate them by gathering extra information about this generic methods. This extra information consists of the not yet investigated generic features and/or internal knowledge level relations.
  3. Get this information and apply it 4 If this process uncovers new task features then go to step 1

- classificatory model Identifying the generic models as general classes is indeed a possible way to structure the knowledge. The generic model features are then used as descriptions of the elements the belong to these classes. This assumption is also correct. The problem solving method can use this structure too. The different classes can be taken as the solutions that are covered in the first place and later one differentiated by investigating the generic models which differentiate the covered classes since they describe the characteristics of the elements of these covered classes and in that way also the class itself.

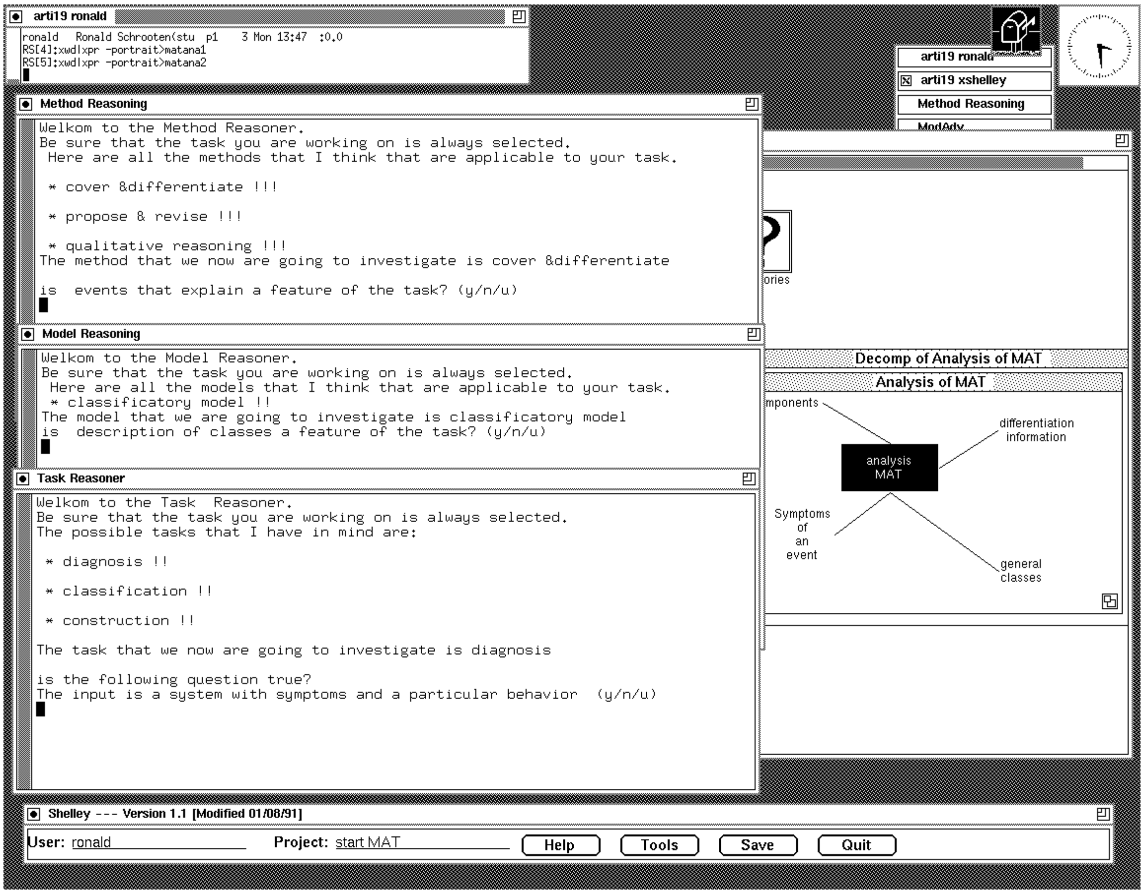

Figure 5.7: Interface of the tool when task features have been added and the reasoner has been invoked.

- classification task Classifying the real world task in one of the generic models is the correct way of looking at the problem. Maybe some people will claim that it is a selection task. I have two reasons that explain why it is not selection. The first is that selection was not included in the library but since this is a weak excuse for the reason that it is not selection I will give the second reason. My actual reply on why the task is not a selection task is that the real world task is never gone be completely defined. It is only described in terms of features that point out some of the characteristics of the task, and since selection assumes that it is possible to select exactly one correct solution from an exact defined solution space, it is not applicable for this case, because the solution space is not exactly defined. It is only described which characteristics it covers and in that way it is more some sort of a class. Also because of the nature of the description of the real world tasks we can only conclude that it fits the best in one of these classes because most of its characteristics are covered by this class, and not because it exactly triggers the correct solution. It is my opinion the the claim that a generic model is the complete high level description of a real world task is a to strong assumption. It is better to look at the generic model as an indicator that gives in an abstract way the major basics of how the design of the expert system that should provide this real world task should look like.

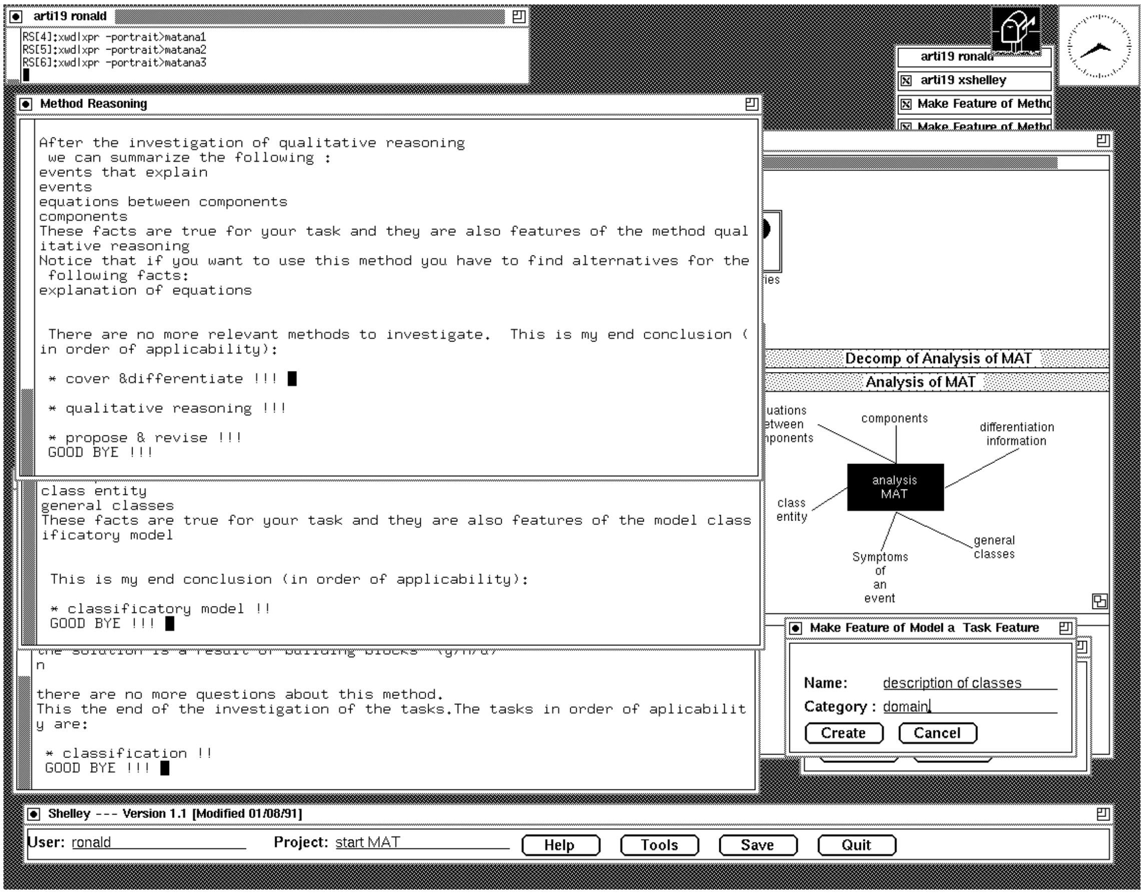

Figure 5.8: The end of the reasoning process.

5.4.1.2 Comparison with the actual implemented model of MAT

The problem solving method is indeed based on the cover and differentiate method. It first makes a general set of possible applicable generic models and the tries to differentiate those by asking extra information. Also the return step can be found back in the implementation model. This provides the feedback loop from the knowledge level to the real world and visa versa. Note that this is one of the major techniques that make the model advisor such an interesting and powerful tool. That the knowledge is structured as a classificatory model is also true. This is easy to see on the lay out of the library which contains the classes and the description of each of these classes. That a classification task is implemented can be seen on the problem solving is done. It is true for the system that it can find some generic models after only a few observations of the task features. That the solution becomes better and better according to the value of the observations is also the way the system works. The conclusion is that the knowledge level components that the tool has covered are the ones that where actually implemented. deciding that the tool is perfect would be to fare from the truth, the claim that it can be very useful in the use of supporting the knowledge engineer in a way that it helps him in focusing on the possible applicable models and that it helps him with the investigation of the different generic models is a more acceptable statement.

5.5 Conclusions

We have seen that the tool provides an active and a passive support for the knowledge engineer. The passive part is that he can make a representation of the real world task that he has to model. This representation can be worked out in a decomposition structure and a description with task features. Also the visualisation of the library and all the underlying links can be seen as a passive support. The active part is concentrated in the reasoning part of the tool. Here the system actually does something. It asks questions and gives remarks about the generic models and it also searches for a usable advise for the knowledge engineer. We have also seen that it is very hard to support the selection process when it comes to determining the best of all the possible applicable generic models. Currently we have almost only used quantitative techniques to determine the applicability of the generic models whereas a qualitative approach is surely needed to make the reasoning part better. This absence of a good evaluation theory is one of the major problems where the tool has to deal with. For the future this search of a good evaluation theory will be one of the major topics in knowledge base research. Since the tool is still in the development faze, (i.e. the current version is only a prototype of the tool) it is not very suitable to be already used by the knowledge engineer as an application for selecting generic models of complex problems. The current power of the tool is that it is extremely useful to evaluate the applicability of the task feature approach, and to experiment with the relations in the library.

Since the implementation of the tool was already started before there were clear thoughts about the modelling guidelines, we can not explicitly find them back in the tool. However the general thoughts of how modelling guidelines should work have been worked out implicitly in the tool. For example: the feedback from the knowledge level to the real world task, the abstract selection based on the mapping of features, and the heuristics that they use for determining the applicability of the generic models.

5.5.1 Theory

- Identifying a problem as a real world task and describing it with task features leads to a terminology problem when the features are intended to be used in the modelling process. I have explained that this language problem can partially be solved with translation dictionaries.

- Determining the applicability of a certain generic model is very hard to do. The solutions so far are based on a quantitative model. The disadvantages of such an approach are that the assumptions that are made for giving values to the features are not always true. It is the nature of the features that their values often depend on other features and this can not be covered by a quantitative approach. A better solution would be based on a qualitative approach. In this case a feature would be completely investigated to give it a value. I have pointed out this qualitative approach can be accomplished in two ways. 1. Asking the user. This is actually passing the problem to the knowledge engineer and so it is not such a good idea for a system that was supposed to support the end user. Note that it is very hard for the user to identify these values. 2. The second approach is trying to determine the value of the features by investigating previously built systems and finding out in which way the feature contributed to the solution. In this approach it is also possible to give values to a set of features instead of seeing them all as individuals.

5.5.2 Application

- The mapping based on pure syntactical comparison appeared to be difficult since it does not support the knowledge engineer with the language problem (note that in the current tool a slot for supporting this problem is already available).

- The idea of having both an active reasoning part and access to all its data is a very good one since this lets the computer do in what it is good at, i.e. search in a large set of generic models those that may be applicable for the task. It also provides an open interaction in which the user can choose at any moment to get advise from the system or stop the reasoning process of the advisory system. This open interaction idea during the reasoning part is a very strong one since it guides the user to all possible uncovered facts and the major advantage is that the reasoner and the knowledge engineer work both on the same time on the same data. Thus the user can switch from the reasoner to the data whenever he wants. He can work on the data apart from the reasoner without really switching the reasoner off. This provides him with the possibility to go back to the reasoner and go on with his work but now again guided by the reasoner that already uses the changed data. This is a feature that can not be found back in many other expert systems.
- The conclusions made by the reasoner are not very accurate since these are based on a weak quantitative reasoning process.
- We conclude that the tool guides the knowledge engineer through the selection process by focusing his attention on those models that are possibly applicable. The investigation of these models is supported by an interactive question/answer process and the overall support of the tool is that it provides the knowledge engineer a strong representation ability which is easy to use and which let him reason about the problem space as well as the solution space.

Chapter 6

Knowledge Validation

This chapter describes some initial results concerning validation in general, and knowledge validation in particular. The purpose of our work was to develop a method for ensuring "that the knowledge that has been modelled is appropriate for the KBS development and that sufficient care and attention have been paid to ensuring faithfulness to the source (or sources) of expertise. Proper usage of the task features must ensure that the conceptual model is, at each stage of its development process, acceptable to expert and knowledge engineer. Knowledge validation is undertaken as a sequence of stages paralleling the different stages of conceptual model development." [de Velde, 1991].

A brief remark about the proceedings of the work on validation is in order here. Initially the validation issue was thought of mostly in terms of validation techniques: "We therefore need to specify a suite of validation techniques which might be applicable at different stages, and issue guidelines for their use." [Tomlinson, 1991] An initial focus was therefore on validation techniques with respect to the expert: structured walk-through, test case collection, and so forth. At the same time there was a strong awareness that validation should be tightly coupled to the modelling process itself. The modelling guidelines have been used as the hooks for this. Each modelling guideline has, potentially, a validation procedure associated with it to validate the appropriates and the result of the modelling operator that it recommended in a certain situation. In this way, at least methodologically speaking, validation was integrated with modelling through a tight "review-plan-work-validate" loop. From here on the work on validation went on, unfortunately at a slow pace. The issues concerning task features and the modelling operators had higher priority. In the meantime another line of work within the project was developing the *Common*KADS Quality Perspective [Williams et al., 1992]. It is only recently that we have started to appreciate the tight relationship between validation and quality assurance. Very little work has been done on these topics considering knowledge systems [Barrett, 1990; Rushby, 1988a]. It seems worthwhile to explore the connection between validation and the *Common*KADS quality assurance in somewhat more depth in the future.

We start with a literature review of definitions for 'validation', 'verification' and 'evaluation' (Section 6.1). Then we propose our own definitions in Section 6.2 and outline an approach to integrating validation in the *Common*KADS modelling framework (Section 6.3).

6.1 Literature Review

Verification and validation has always been an important issue in classical software engineering. More recently the verification and validation issue for knowledge systems has rapidly gained in importance. Tools and elements of a methodology are being developed and specialized workshops and conferences are being organized. In this section we provide an overview of verification and validation definitions and approaches from the literature on classical software engineering and knowledge systems.

6.1.1 Software Engineering

6.1.1.1 IEE standards organisation

IEE standards organisation has provided the following definitions for conventional software (IIII Std 729-1983):

Validation: the process of evaluating software at the end of the software development process to ensure compliance with software requirements.

Verification: the process of determining whether or not the products of a given phase of the software development cycle fulfill the requirements established during the previous phase.

6.1.1.2 Adrion, Branstad and Cherniavsky

Adrion, Branstad and Cherniavsky [Adrion et al., 1982] give the following definitions:

Validation: the determination of the correctness of the final program from a development project with respect to the user needs and requirements.

Verification: the demonstration of consistency, completeness and correctness of the software at each stage and between each stage of the software development life cycle.

6.1.1.3 Green and Keyes

Green and Keyes [Green and Keyes, 1987] give these definitions as those used in software engineering:

Validation: The deliverable code correctly implements the original user requirements. The process by which the deliverable code is directly shown to satisfy (or fail to satisfy) user requirements is called validation.

Verification: Each level of specification and the deliverable code are traceable to a superior specification; that is, the specification or code fully and exclusively implements the requirements of the superior specification. This process of determining traceability is called verification.

6.1.1.4 Sommerville

Sommerville [Sommerville, 1989] basis his definitions on those of Boehm [?]:

Validation: "Are we building the right product?" [?]. Validation involves checking that the program as implemented meets the expectations of the user.

Verification: "Are we building the product right?" [?]. Verification involves checking that the program conforms to its specification.

6.1.2 Knowledge Engineering

6.1.2.1 Suen et al.

Suen, Grogono, Shinghal and Coallier point out that the IEEE definition of validation is inappropriate for Expert Systems as it recommends delaying testing until the end of the development process.

> "We divide evaluation into two components: verification and validation.... Verification occurs during the development of the expert system. During verification, we view the expert system as a 'glass box' and check that each component meets its specifications. During validation, we view the expert system as a 'black box' and observe its responses to test data" [Suen et al., 1990]

They give the following definition for validation:

Validation: validation is the process of evaluating an Expert System during and after the development process to ensure compliance with the initial requirements.

They provide the following list for verification of Expert Systems:

- structure checking finds:
  - inconsistency
  - redundancy
  - subsumption
  - cyclic dependencies
- semantic checking finds:
  - range errors
  - cardinality errors
  - illegal values
  - correctness rules

6.1.2.2 Bensabat and Dhaliwal

We quote from Bensabat and Dhaliwal [Bensabat and Dhaliwal, 1989]:

> "If the objective of the KA process is to capture the knowledge of an expertise source then the validity of the KA process is the degree of homomorphism between the representation system i.e., KBS, and the system that is supposed to represent, i.e., the expertise source (Vandierendonck, 1975). This representational homomorphism definition of validity is consistent with Bobrow's (1975) notion that at an abstract level, a representation can be viewed as 'the result of a selective mapping of aspects of the world'. A world-state (expert source) can be modelled into a knowledge-state (KBS) and questions about the world-state can be answered either by observing the world-state or by questioning the corresponding knowledge-state"

In [Hoppe and Meseguer, 1991] their definitions are rephrased as follows:

Verification: is the demonstration of the logical properties (e.g. consistency, correctness and completeness) in the development life cycle of a system.

Validation: is the determination of the degree of homomorphism between a system and its representation. The more specific term 'representational validation' is used to denote the 'validation' of a system against a source of expertise. under 'representational validation' they distinguish further between

- Structural Matching which refers to the validation of knowledge representation and processing, and
- Behavioral Matching which refers to the validation of the similarity of output performance between a KBS and the source of expertise.

Evaluation: is the measurement of certain properties (e.g. quality, utility and usability) of a system.

6.1.2.3 Motta et al.

Motta, Rajan, Domingue and Eisenstadt, in their work on KEATS, give the following definitions [Motta et al., 1990]:

Validation: "Validation assesses the mapping between a source and a target system. In knowledge engineering, validity should provide a measure of how faithful the system's architecture and bahaviour are with respect to the problem area. Unfortunately, this comparison is made difficult because of the many confounding factors that can arise at the implementation level, such as efficiency, software design and machine oriented issues. Therefore, we believe validity should mainly be concerned with assessing the adequacy of the conceptual model developed. At the knowledge encoding level, validity becomes a fuzzier concept, as computational issues may lead the knowledge engineer to design system architectures that are structurally different from the conceptual model."

Verification: "Verification is the process of assessing the consistency, correctness, and completeness of a knowledge base.... While correctness and consistency checking normally depend on the formalism being used, completeness is very much a semantic property depending upon the task or the problem solving method being employed."

6.1.2.4 VITAL

The definitions in the VITAL technical annex are based on the ones from KEATS (Section 6.1.2.3 and [Motta et al., 1990]). It is added: "As a KBS is both a piece of software and a model, validation will essentially occur for modelling aspects whereas verification will be addressed by software" (p.61) and "The validation tools will focus on appraising knowledge acquisition session effectiveness and the verification tools will address the issue of static analysis".

6.1.2.5 VALID

In VALID [Brender et al., 1989] one distinguishes between 'verification' and 'evaluation' as two sub-tasks of 'validation'.

Verification: proving that the KBS is in accordance with specifications or expectations. It is further distinguished between verification of knowledge base formation (concerned with content) and verification of the KBS functionality (concerned with behavior)

Evaluation: judging or measuring the KBS characteristics with the purpose of assessing KBS quality by determining the value of predefined quality measures. Again it is further distinguished between evaluation of knowledge base formation and evaluation of the KBS functionality.

6.1.2.6 Ayel and Laurent

In the introduction to [Ayel and Laurent, 1991] the validation and verification processes are distinguished according to the persons who use them:

Validation: "The 'customer', that is the person placing the order and/or the future user(s), is concerned with the validation process. For him the question is as follows: 'Is the system provided by the builder the right product? Does it actually solve my problem?'

The customer looks for Usability, Competences, Performances, Reliability of the product. The validation process is directly concerned with the behavior of the Knowledge-based System."

Verification: "...unlike the validation process, the verification process is based on the builder's thorough knowledge about the product he has developed. This knowledge is mainly concerned with technical details about the product and with its components. The verification process stems from this kind of knowledge.

The builder looks for syntactic properties, Coherence, Completeness, Correctness, Robustness but also Performance.

The same authors point out the difference between validation and verification of knowledge-based systems on the one hand, and knowledge validation or verification on the other.

6.1.2.7 Hoppe and Meseguer

From a review of terminology in KBS and classical software engineering Hoppe and Meseguer propose the following goals for verification, validation and evaluation [Hoppe and Meseguer, 1991]:

Verification: "which has the goal to check well-defined properties of a knowledge-based system against its specification." They point out that verification can be focussed on parts of the KBS or on specific aspects of KBS functionality.

Validation: "which has the goal to check whether a knowledge-based system corresponds to the system it is supposed to represent." Just like verification, validation can focus on parts or aspects of the KBS. They note that in this sense testing of i/o behavior is an important validation method.

Evaluation: "which has the goal to assess or measure quantitative and qualitative characteristics of a knowledge-based system and to compare them with expected or desired patterns."

In relating these definitions to classical ones they end up with the following:

Verification: "consists of checking a KBS against its formalizable specifications. It is performed by checking and not by proving KBS properties. It should be considered during the whole KBS life-cycle."

Validation: "consists of assuring that the final KBS complies with the user needs and requirements. Validity can only be partially achieved by verification, since there are informal specifications which cannot be checked by verification. Therefore, specific validation methods are required to check informal specifications. It should be present during the KBS life cycle."

6.1.2.8 Meseguer

In [Meseguer, 1992] it is started from user requirements. These are classified into totally formalizable and partially formalizable, depending on their total or partial translation into specifications.

Verification: "is the process of checking an ES against its totally formalizable requirements"

Evaluation: "considers ES compliance with respect to its partially formalizable requirements"

The author goes on:

> "Combining these definitions, validation appears composed of two parts: verification and evaluation [Laurent 92]. This separation is induced by the existence of two types of requirements, differentiated by a syntactical criterion. In this way, properties that can be objectively checked (by verification) are differentiated from the other aspects that require subjective assessment (by evaluation)."

6.1.3 Psychology

In psychometrics [?] validity is usually contrasted with reliability. Reliability refers to a low degree of measurement error, whilst validity is the degree to which a measuring instrument actually does what it is intended to do.

6.2 Validation of Expertise Models

6.2.1 Proposed Definitions

In line with the opinions one finds in the literature we believe that validation is concerned with the modelling process, whereas verification is concerned with software development. Evaluation is concerned with the end-product and the end-users.

We also agree that validation can focus on different parts or different aspects of the KBS. Thus the term knowledge validation makes sense as meaning "validation of the expertise model that describes the KBS behavior".

Knowledge validation: attempts to ensure that the expertise model that is or is going to be used for KBS development is appropriate. Knowledge validation can be undertaken as a sequence of stages paralleling the different stages (state transitions) of modelling.

Verification: is aimed at ensuring the quality of the process that transforms the product model, and the expertise in particular, into a functional KBS. It is the demonstration of consistency, completeness and correctness of the software at each stage and between each stage of the software development life cycle.

Evaluation: assesses the usefulness and usability of the system. Gaines (1988) defines this as the extent to which the system performs the intended application (its value). Usefulness refers to the utility and benefits derived from the system; usability is associated with the understandability of results, human-engineering aspects, ease of modifications etc. These should be evaluated by users. Evaluation should also be undertaken by experts to determine both the quality of the knowledge-base and of the advice that the KBS provides.

These choices are in line with most of the literature, but conflict with the VALID definitions. Verification is the same as in software engineering.

6.2.2 Validation of expertise models

The following is a list of some of things a knowledge engineer may wish to validate[1]:

- that the correct choice of Interpretation Model has been made
- that the correct refinements have been made to the basic Interpretation Model
- that there is appropriate knowledge in the expertise model
  - structural faithfulness to the source expert(s)
  - behavioral faithfulness (similarity of output or expected performance of the expertise model and the expert(s)
- that models from different experts have been correctly integrated
- the appropriateness of the model

[1] entirely based on input from Christine Tomlinson, LR

  - correct integration of expertise and communication models
  - relevance and adequacy in terms of the objectives of the intended application of problem being solved

The types of evidence that may be used in validation are the following, among others:

1. Content-related - assessing the adequacy of the content by examination of:

   - the domain definitions
   - the adequacy of the elicitation process,
   - that the content elicited is representative of the domain,
   - that non-relevant items have not been included

   The adequacy of the process is evaluated by having independent party assess the extent to which the steps described above are followed properly. By e.g., 'structured walkthrough' - examining the contents of a representation of the KB at different stages of the KBS life cycle. The intention is to find errors, omissions, inaccuracies, contradictions and redundancies.

   Difficulties with content-related evidence:

   - Need a good representation
   - Is a costly and time-consuming process

2. Criterion-related evidence: assessment by demonstrating a relationship between its behavior and some other criterion measure, e.g., a comparison of KBS performance and expert.

   Testing may be predictive e.g., generating scenarios. Or postdictive, where the criterion data has been collected in the part, e.g., the comparison of KBS output against test-cases.

   Test cases should:

   - be non-fictional
   - cover a range of levels of difficulty
   - be comprehensive

6.3 Validation and the Modelling Process

Validity testing should unfold as a sequence of stages paralleling the different stages of developing the conceptual model. The general model as outlined in Chapter 2 puts forward the view of modelling as a process of repeated applications of selected modelling operators.

The selection is based on modelling guidelines. They indicate preferred operators based on task features and the state of modelling. The following is the structure of a modelling guideline as introduced in Chapter 2. One of the slots is the hook for the validation process.

Slots of a modelling guideline:

> **modelling-guideline** <*identifier*>
> **sub-type-of:**
>   <**modelling guidelines**>
> **task feature:**
>   <**model feature - value pair**>
> **expertise model state:**
>   <**expertise model state**>
> **external model state:**
>   <**milestone state**>

operator:
<modelling operator>
operator preference:
<preference indicator>
suggested next operator:
<modelling operator>
result validated by:
<validation procedure>

When an operator has been applied as the result of a recommendation from a modelling guideline then the validation procedure can be used to test, after the fact, whether this choice is still likely to be a good one. As examples of validation procedures the example guidelines in Section ?? mentioned only "expert" or "knowledge engineer". This is, we admit, too shallow a procedure description. Clearly we need to specify a suite of validation techniques which might be applicable at different stages. The guidelines for their usage would then be implicit in the modelling guidelines that recommend them. Appropriate validation techniques should be specified with respect to any of the features that occur in the modelling guideline, and in particular with those that can be expected to have a strong impact on the quality of the product. In addition validation procedures must differentiate between the different scenarios that can be used when applying a modelling operator, for example:

- with respect to the expert
- with respect to a wider population of expertise
- when dealing with multiple expertise, for
  - overlapping expertise (looking for consensus)
  - distributed knowledge (looking at networking knowledge for knowledge creation).

6.4 Validation and Quality Assurance

It is useful to take a wider perspective on validation and to view validation in the framework of software quality assurance. Quality assurance is concerned not only with assuring the compliance of a program with the user requirements at the present, but also with those aspects that influence how well the software will continue to satisfy the user needs in the future [Rushby, 1988b]. Very little work has been done on these topics considering knowledge systems [Barrett, 1990; Rushby, 1988a]. The *Common*KADS Quality Perspective [Williams et al., 1992] is a notable exception and it seems worthwhile to explore the connection between validation and quality assurance in somewhat more depth.

The *Common*KADS Quality Perspective provides a clear framework, largely based in classical software engineering, for building quality into the life-cycle. The CommonKADS life-cycle model [Taylor et al., 1992a] details a risk based approach to management that is heavily influenced by Boehm's spiral model. In this approach management of software development is seen in terms of repeated and recursive appliation of a cycle of events which can be summarized in terms of *review, risk, plan and monitor* activities. Quality is build into this by the definition of *quality characteristics* of the end-product.

Validation is tightly connected to these quality characteristics. Looking at the list of things that one may want to validate for one finds such terms as "appropriate", "correct", "adequate". These are vague terms. As a first step toward a more systematic appraoch to validation we propose to make explicit the meaning of these terms in terms of quality characteristics. The CommonKADS quality viewpoint provides a terminology for this which is based in ISO standards [9001, 1987].

These terms capture in a broad sense the quality of a system with respect to the end-user which is exactly what we need for validation.

A modelling guideline recommends the application of a certain modelling operator when a set of task features are present. In Chapter 3 we have proposed two orthogonal categorizations of task features. One the one hand:

- epistemological,
- pragmatic,
- computational,

and on the other hand:

- related to systems,
- related to purpose,
- related to environment,
- related to grounding

Current thinking is in the direction that the first categorization is related to classes of risks while the second one provides for a language to qualify that risk. The task features that occur in a modelling guideline thus also set the goal of validation. It has to be checked in validation whether the result of the modeling operator adequatly counteract the risk indentified. To do this a systematic analysis is needed of how aspects of the expertise model (complexity, content, choice of method, and so forth) impact on quality characteristics of the end-product. Elsewhere we have proposed a systematic approach toward such an analysis [Duursma et al., 1992].

Part II

The Inventory

Chapter 7

A Task Feature Inventory

This chapter presents the result of identifying relevant task features, on the basis of theoretical and example-focused analysis. The inventory presented is an initial one, intended to be revised and extended by applying the features in analysing practical applications.

Since the main role of the task features is to help in selecting and manipulating components of the library, the feature set must be able to appropriately discriminate between library components. As stated in chapter 3.4, a criterion for the usefulness of a task feature is that it is named in the condition part of a modeling guideline. Hence, the task feature inventory initiated here - as well as the modeling guidelines listed in chapter 8 - must be updated and maintained along with the addition of components into the library.

As described in chapter 3.6, task feature names which match left-hand-sides of modeling guidelines, have a set of predefined values or value types. The actual value of each task feature - for a particular application - is given as an answer to one or more questions associated with that task feature. The inventory of task features presented below is structured accordingly. That is: task features are grouped in a hierarchy of abstract task features, and each task feature has a question associated.

In this chapter, the following general scheme is used to textually describe a task feature:

- Task feature name
  User question for getting the feature value
  Optional answers, corresponding to legal feature values or value types

When realizing this in a modeling support tool, the descriptions will have to be extended as specified by the object structure of task features, presented in chapter 2.3.

In the following sections, the task features we have derived from literature and theoretical analysis, and those derived from analysis of example systems, are presented as separate feature sets. They should eventually be merged into a single set, a process which should be guided by an evaluation of their usefulness with respect to the library.

The number of task features will heavily depend on the type of questions associated with them. Whether to go for many yes/no questions, or fewer questions with more options in each, is an issue of workbench design rather than of the methodology as such. The questions below combine several alternatives.

7.1 Task purpose features

- **Goal features**

- Structural ordering.
  Does the system need to (re)order components according to requirements?
  (y/n)
- Selection and combination.
  Does the system need to select and combine components in order to fulfill certain requirements?
  (y/n)
- Comparing behaviour.
  Does the system need to compare actual behavior of an artifact with expected behavior and notify anomalies?
  (y/n)
- Identifying malfunction.
  Does the system need to identify a malfunction in a system?
  (y/n)
- Explanaton creation
  Does the system need to create an explanation for the malfunctioning of a device?
  (y/n)
- Specification
  Is the goal of the task to find a complete specification of a set of 'primitive' components and their relations so as to meet a set of constraints?.
  (y/n)
- Case evaluation
  Is the goal of the task to evaluate a specific case?
  (y/n)
- Collection of components
  Is the goal of the task to collect a set of components?
  (y/n)
- Assistance mode
  What is the role of the system with respect to the environemnt or interacting agent?
  (oracle/advice-giver/teacher/colleague/student)
- Domain adaptability.
  Is it a task of the system to be easily changeable in the case of changes in the task-domain?
  (y/n)
- Explaining decisions.
  Is the system required to give detailed explanations of its decisions - and if so, what is to be explained?
  (no/conclusions/inferencing/strategies)

In the continuation *answers of the (y/n) type is assumed as default*, and therefore not explicitly included in the task feature descriptions. Other optional answer values are indicated.

- **Input features**
  This category of task features describes abstract properties of the task input. Example input features are:

  - **Form of input:**
  - Case description
    Is the input of the task a case description?

- Typical size of the data set
  What is the typical size of the data set?
  ([number])
- Maximum size of the data set
  What is the maximum size of the data set?
  ([number, of attribute-value pairs])
- **Content of input:**
- Input classes
- What is the input, with reference to system and other domain terms?
  ([terms/propositions/free-text])
- Correctness of data
  To what degree is the input data expected to be correct?
  (always/to-a-large-degree/to-some-degree/never)
- Noise in data Is a significant portion of the data expected to be noise?
- Completeness of data
  To what degree is the input data expected to be complete?
  (always/to-a-large-degree/to-some-degree/never)
- **Format of input:**
- Format specification
  What kind of format does the input have?
  (numerical/named-terms/free-text/attribute-values/structured-object)
- **Pattern of input:**
- Frequency of input
  How frequent will input arrive during a problem solving session?
  (at-the-beginning-only/sometimes/often/continously)
- Regularity of input
  To what degree does the form and content of the input vary between and within sessions?
  (always/to-a-large-degree/to-some-degree/never)
- User interaction pattern
  To what degree is the user expected to interact with the system during a session?
  (user-always-in-control/to-a-large-degree/to-some-degree/never)

• **Output features:**
This category of task features describes abstract properties of the task output. Example output features are:

- **Form of output:**
- Binary value
  Is the output of the task a binary value?
- Multiple value
  Is the output of the task multi-valued?
- Typical number of items in the output
  What is the typical size of the output?
  ([number])
- Maximum number of items in the output
  What is the maximum size of the output?
  ([number])

  - **Content of output:**
    What is the output, with reference to system and other domain terms? ([terms/propositions/free-text])
  - Correctness of output
    To what degree is the output expected to be correct?
    (always/to-a-large-degree/to-some-degree/never)
  - Noise in output Is it possible that the output contains noise?
  - Completeness of output
    To what degree is the output expected to be complete?
    (always/to-a-large-degree/to-some-degree/never)
  - **Format of output:**
  - Format specification
    What kind of format does the output have?
    (numerical/named-terms/free-text/attribute-values/structured-object)
  - **Pattern of output:**
  - Frequency of output
    How frequent will output be produced during a problem solving session?
    (at-the-end-only/sometimes/often/continously)
  - Regularity of input
    To what degree does the form and content of the output vary between and within sessions?
    (always/to-a-large-degree/to-some-degree/never)

- **Changes in the task purpose**
  - Multiple purposes
    Does the task purpose ever change within a single problem solving session?
  - May the task purpose change - evolve - over time?

7.2 Task domain features

Subcategories of task features related to the domain:

- **General system characteristics:**
  - System type
    What type of system is the task about?
    (object - a 'thing' in the real world / process - a sequence of events)
  - Object type
    What type of object is the system?
    (natural object/artifact/physical system/ living organism/human being/dead matter/abstract object)
  - Process type
    What type of process is the system?
    (industrial/biological/social/real-time/automatic/operator controlled)

- Knowledge types.

  Types of knowledge needed, or normally used, to accomplish the task (partly based on the classification in [Steels, 1991]. Typically, several of these knowledge types are combined.

- Common sense knowledge needed
  Is common sense knowledge needed to achieve the task?
- Common sense knowledge used
  Is common sense knowledge normally used to achieve the task?
- First principles knowledge
  Is a theory based on first principles (physics, mathematics, logic) used?
- Temporal knowledge
  Is knowledge about time dependiencies (temporal knowledge) used?
- Spatial knowledge
  Is a knowledge about spatial relations (positions, orientations, shapes, distances, connections) used?
- Heuristic, associativ knowledge
  Is heuristic knowledge (if-then associations) used?
- Structural knowledge
  Is structural, hierarchical knowledge about domain concepts (subclass- and instance hierarchies) used?
- Componential knowledge
  Is knowledge about how objects are decomposed into parts used?
  * Structural decomposition:
    Can the system be described in subsystems?
  * Number of decomposition levels
    How deep can the system usefully be decomposed? ([number])
  * Number of sub-systems per level
    How many systems are there (typically) at each level of decomposition? ([number])
  * Component description language Is there a language to describe the behavior of the subsystems at a decomposition level?
  * Input output relations
    Are input-output relations of (sub)system available?
  * Connections between subcomponents
    Is there knowledge about how subparts are connected into subsystems?
- Functional knowledge
  Is functional knowledge (the functions that objects or processes serve) used?
- Behavorial knowledge
  Is behavorial knowledge (how (sub)systems or objects behave) used?
- State knowledge
  Is knowledge naturally tied to particular states of the system?
  * Reference model
    Is there a reference (state) model?
- Causal knowledge
  Is causal knowlegde (causal relations between states of system parts) used?
  Causal explanations
  Can the function of the system be explained in terms of causal relations between states of parts?
- Normal system
  Is knowledge about normal system behaviour used?

  - Abnormal system
    Is knowledge about abnormal system behaviour used?

- Domain knowledge generality

  - Generalized knowledge
    To what degree may the domain be described in terms of general domain knowledge (theories, concepts and relations, heuristic rules)?
    (sufficient/large/significant/none)
  - Specialized knowledge
    To what degree may the domain be described in terms of specific domain knowledge (cases, concrete episodes, examples)
    (sufficient/large/significant/none)

- Domain knowledge strength
  Is the domain knowledge mainly weak, uncertain and incomplete, mainly strong, certain, and proof-capable, or a balanced combination?
  (weak/strong/combined)

- Stability of domain
  How stable is the domain knowledge over time?
  (changes often/changes seldom/stable)

- Boundary of domain
  Has the domain fixed boundaries (classification tasks)?

Features for combined domain-purpose pairs:
Below, examples of domain characteristics related to a particular task type (task purpose) are shown. Task domains are typed into diagnosis (classification) tasks, and design (construction), and a set of features are listed - in question form:

- Domain features specific for the diagnosis task.

  - Causal diagnosis theory
    Is there a causal theory of non-functional (sub)systems?
  - Multiple faults
    Can a symptom be caused by more than one malfunction?
  - Multiple causes for a fault
    Can a malfunction have more than one cause?
  - Multiple explanations for a cause
    Can a cause be explained by more than one explanation?
  - Multiple simultaneous
    Is it possible that more than one malfunction occurs simultaneously?
  - Numberumber of states
    How many states can a component have (or be in)?
  - Number of observables
    How many observables can there be?
  - Number of symptoms
    How many symptoms (fault indicating observables) are there?
  - Filtering of symptoms
    Can symptoms 'easily' be separated out from ther observables?

- Domain features specific for design (construction) tasks.
  - Formal requirements.
    Are there formal requirements that must be satisfied?
  - Available components.
    Are there exisiting components to build from?
  - Number of components necessary.
    How many components are typically involved in the design?
    ([number])
  - Number of possible combinations.
    Is there a large number of practically possible combinations of components?
    (all components combine in some ways/some components combine in some ways/ few components combine/no)
  - Input output definitions
    Are there I/O definitions of the components available?
  - Other component properties
    Are other significant properties (e.g. size, power consumption, weight) available?
  - Side effects
    Are there any important side-effects of the components?
  - Relation between components
    Is a set of relations, or dependencies, between the components available?
  - Requirements of structures
    Are there particular substructures or subparts of the design which have specific and important requirements?

7.3 Task environment features

In this category features of the task environment are specified, which in one way or the other influence the various agents' behavior. The task environment is the organization in which the KBS system has to operate. It includes the user(s), safety requirements, costs and stability of the domain.

- Physical conditions
  - Are there special physical conditions of the task environement, which which will constrain the system's operation?
    ([list of predefined terms]/no)
- Safety
  - Risky tests
    Are any observations/tests more risky or dangerous than others?
  - Dangerous states
    Are any external states more dangerous than others?
  - Unsafe configurations
    Are any potential system configurations unsafe for the environment?
- Costs
  - Costs of data gathering procedures
    Are the costs of the data gathering process important?

- Costs of devices
  Are the costs of acquiring and maintaining the data gathering equipment important?
- Costs of solution
  Are the costs of realizing a solution, e.g. producing repairing something, of high importance?
- Costs of time
  Is the problem solving time (real time) an important cost factor?

- User
  - Presentation
    What style of presentation - interfacing - is needed or preferred?
  - Explanations
    What kind of explanations are needed/preferred by the user?
  - User competence
    What is the level of competence of the user?
  - User reasoning
    Is there a particular type of reasoning that the user uses? (theory knowledge/rules/previous episodes/tacit skill/no)
  - Are there any typical user errors? (slips, mistakes)?

7.4 Grounding features

Grounding features are grouped into features related to agents and resources.

- Domain grounding.
  - Measurable data
    Are actual I/O values of (sub)systems measurable by physical devices?
  - Observalble data
    Are actual I/O values of (sub)systems directly observable by humans?
  - Reliability of data-source
    To what degree are the I/O values of the (sub)systems reliable?
    (always/to-a-large-degree/to-some-degree/never)
  - Costs of I/O
    Are the costs of observing and measuring particularly high?
  - Manipulation constraints
    Are there important constraints and limitations to how the system (which the task is about) can be manipulated or affected?
  - Manipulation opportunities
    Are there particular opportunities and potentials for how the system (which the task is about) can be manipulated or affected?

- Inference grounding.
  - Computational features of inference in models:
  - Combinatorial explosion
    Does solving the task easily lead to problems of combinatorial explotion?
  - Computational complexity
    Does solving the task lead to a computationally complex process?

- Relations to human reasoning
  Is the human way of reasoning about the task a plausible model for achieveing computer reasoning?
- Computational capacity of agents:
- Type of inference
  What type of inferences should the agent be capable of? (deductive/abductive/inductive)
- Size of working memory
  Are there general requirements on working memory size?
- Speed of processing
  Are there general requirements on processing speed?

• Resources.

- Time resource
  Are there particular restrictions on the agent's computing time, appart from the environmental time constraint for solving the entire task?
- Computational resources
  Are there requirements on the overall computational costs, e.g of making observations, interacting with user, drawing inferences, etc., in terms of memory (storage) and processor resources?

Chapter 8

An Inventory of Modelling Guidelines

Here follows an initial set of modeling guidelines. These guidelines have been taken form literature and experience in analysis and design. The modeling guidelines should be viewed, basically, as a hierarchical rule set. In the list given here, however, the modeling guidelines are presented as an unstructured collection of rule-like objects, presented as simplified and easier readable forms of the object structure described in chapter 2.

8.1 Task identification guidelines

1. If the goal of the task is to find an explanation for a malfunction of a device

   THEN it the task is a diagnosis task.

   task purpose feature explanation creation for the malfunctioning of a device.

   modeling-guideline *1*
   modeling-goal:
   task identification
   task-features:
   purpose → explanation creation for the malfunctioning of a device
   operator:
   select(diagnosis-task)

2. If the task is a diagnosis task

   AND the system to be diagnosed is a human made artifact

   THEN the task is a Class III diagnosis task.

   task domain feature Type of system.

   modeling-guideline *2*
   modeling-goal:
   task identification
   task-features:
   domain → human made artifact
   expertise-model state:
   selected(diagnosis task)
   operator:

select(Class III diagnosis-task)

3. If the task is a diagnosis task
   AND the device is a organism
   THEN the task is a Class IV diagnosis task
   task domain feature: Type of system.

   modeling-guideline *3*
   modeling-goal:
   task identification
   task-features:
   domain → organism
   expertise-model state:
   selected(diagnosis task)
   operator:
   select(Class IV diagnosis-task)

4. If the goal of the task is to find a evaluation of a case
   AND the input of the task is a case description
   AND there is a reference model
   THEN the task is an assessment task.
   task purpose feature: Finding an evaluation of a case (goal), case description (input).
   task domain feature: Reference model (domain knowledge types).

   modeling-guideline *5*
   modeling-goal:
   task identification
   task-features:
   purpose
   goal → evaluation of a case
   input → case description
   domain
   there is a reference model
   operator:
   select(assessment-task)

5. If the task is an assessment task
   AND the output of the task is a binary value
   THEN the task is an 'acceptance' assessment task.
   task output feature: Type of output.

   modeling-guideline *6*
   modeling-goal:
   task identification
   task-features:
   output → binary valued evaluation
   domain

there is a reference model
expertise-model state:
selected(assessment-task)
operator:
select(acceptance-assessment-task)

6. If the task is an assessment task

   AND the output is a multi-valued evaluation

   THEN the task is a grading task.

   task output feature: Type of output.

modeling-guideline *7*
modeling-goal:
task identification
task-features:
output → multi valued evaluation
domain
there is a reference model
expertise-model state:
selected(assessment-task)
operator:
select(grading-task)

7. If the goal of the task is to find a complete specification of a set of 'primitive' components and their relations so as to meet a set of constraints

   THEN the task is a design task [Brown and Chandrasekaran, 1989]

modeling-guideline *8*
modeling-goal:
task identification
task-features:
purpose
goal
To find a complete specification of a set of
primitive components and their relations as to meet
a set of constraints.
input
→ set of primitive components
→ set of constraints
output → complete specification
operator:
select(design-task)

8. IF the task is a design task

   AND the design requirements are ill specified

   AND there are no known decompositions

   THEN the task is a class I design task.

   task goal feature: Ill specified requirements (input, context design)

   task domain feature: Availability decompositions (context design)

modeling-guideline *9*
modeling-goal:
task identification
task-features:
purpose
input
→ set of constraints is ill specified.
domain → No know decomposition of the artifact.
expertise-model-state:
selected(design-task)
operator:
select(Class-I design-task)

9. If the task is a design task

   AND the are known decompositions of the problem

   AND the failure analysis of the design is hard (i.e to find a repair for a violated design constraint).

   AND synthesis in the design space is especially complicated

   THEN the task is a class II design task.

   task purpose feature [1]: Availability of problem decompositions, failure analysis, synthesis of design space, complexity of design task task.

modeling-guideline *9*
modeling-goal:
task identification
task-features:
domain
→ known decomposition of the artifact.
→ synthesis space complicated.
expertise-model-state:
selected(design-task)
operator:
select(Class-II design-task)

10. If the task is a design task

    AND effective problem decompositions are known

    AND actions to deal with design failure are all explicitly known

    AND there is very little auxiliary problem solving needed

    AND the task is still too complex for 'simple' algorithmic solutions

    THEN the task is a class III (routine) design task.

    task purpose feature: Availability of problem decompositions, auxiliary problem solving, complexity of task.

11. If the task is a design task

    AND the goal of the task is to select a set of components from a fixed set of possibilities

    AND the resulting set can be evaluated

    THEN the task is a class III design task.

    task purpose feature: Selection of a set of components from a fixed set of possibilities.

[1] These features make only sense within the context of the design task

modeling-guideline *10*
modeling-goal:
task identification
task-features:
domain
→ known decomposition of the artifact.
→ actions available to deal with design failure are known
→ very little auxiliary problem solving needed
→ too complex for simple algorithmic solutions.
→ failure analysis is hard.
expertise-model-state:
selected(design-task)
operator:
select(Class-III design-task)

12. If the task is a class III design task

    AND the elements of the goal set can be evaluated independently

    THEN the task is a class III-a design task.

    task purpose feature: Independent evaluation of parts of the task goal.

modeling-guideline *11*
modeling-goal:
task identification
task-features:
domain
→ elements of goal set can be evaluated independently.
expertise-model-state:
selected(Class III design-task)
operator:
select(Class-III-a design-task)

13. If the task is a class III design task

    AND the elements of the goals set cannot be evaluated independently

    THEN the task is a class III b design task.

    task purpose feature: Independent evaluation of parts of the task goal.

modeling-guideline *12*
modeling-goal:
task identification
task-features:
domain
→ elements of goal set can NOT be evaluated independently.
expertise-model-state:
selected(Class III design-task)
operator:
select(Class-III-b design-task)

14. If the goal of the task is create a sequence of activities that has a specified effect

    THEN the task is a planning task.

    task purpose feature: creation of a sequence of activities that has a specified effect.

15. If the task is to combine a number of activities in order to create a sequence with a certain attributes

    THEN the task is a scheduling task.

    task purpose feature: Combination of activities in order to create a sequence with a certain attributes

modeling-guideline *12*
modeling-goal:
task identification
task-features:
domain
→ elements of goal set can be evaluated independently.
expertise-model-state:
selected(Class III design-task)
operator:
select(Class-III-a design-task)

16. If the goal of the task is to find a set of matching pairs on the base of an input list of keys

    THEN The task is a collection task.

    task purpose feature: matching pairs on the base of an input list.

modeling-guideline *13*
modeling-goal:
task identification
task-features:
purpose
goal → find a set of matching pairs on the base of an input list of keys.
input
→ pairs
→ keys
operator:
select(Collection task)

8.2 Method identification guidelines

Method identification guidelines should all be formulated in task specific terminology.

1. If the constraints of the task are completely known in advance.

   And The task goal can be specified by a set of values of attributes.

   And there is a set of partial solutions.

   And there is a set of solution revisions.

   And partial solutions can be evaluated.

   And remedies can be specified for each violated goal constraint.

And there is not a high level of potential conflict for alternative solution extensions.

THEN

RLM method propose and revise is an applicable method.

modeling-guideline *14*
modeling-goal:
method identification
task-features:
purpose
goal → Can be specified by as set of values of attributes
domain
→ There is a set of partial solutions.
→ Partial solutions can be evaluated.
→ There is a set of solution revisions.
→ Remedies can be specified for each violated goal constraint.
operator:
select(Propose & Revise method)

2. If the task can be described in terms of states

   And the begin state and the end state are known

   And there is a list of possible operators from each state

   And the there is a distance evaluation possible at each state of the problem

   THEN

   Means ends analysis is an applicable method for this task definition.

modeling-guideline *15*
modeling-goal:
method identification
task-features:
purpose
goal → Goal can be formulated as a desired state.
domain
→ Start state is known.
→ There is a set of possible operators form each state..
→ There is a distance evaluation possible form each identified state.
operator:
select(Means - Ends Analysis method)

3. If The task can be described in terms of a mapping from many valued symptoms of a system to many valued classes.

   And A large set of quantitative source feature, class pairs is available.

   And Similarities among features of a problem imply similarities among the solution of the problem.

   And the Similarities among features of the problem and the solution remain constant over time

   And No deep explanations necessary why a solution solves a problem.

   THEN

   'extrapolate from a similar case' is a possible method for the task.

modeling-guideline *16*
modeling-goal:
method identification
task-features:
purpose
goal → Can be specified as a mapping from many valued symptoms to many valued classes.
domain
→ A large set of quantitative source feature, class pairs is available.
environment
→ Similarities among features of a problem imply
similarities among the solution of the problem.
→ No deep explanations necessary why a solution solves a problem.
operator:
select(Extrapolate form a similar case method)

4. If There is an identifiable and enumerable set of problems.

   And There is an identifiable and enumerable set of problem symptoms.

   And There is a static enumeration of sub-problem, sub-solution pairs.

   And Sufficient problem case symptoms are available and observable to differentiate the possible sub-solutions.

   There is only one solution possible at any time given

   THEN

   'cover and differentiate' is a possible method for the task

modeling-guideline *17*
modeling-goal:
method identification
task-features:
purpose
goal → Can be specified by as set of values of attributes
domain
→ There is a identifiable and enumerable set of problems.
→ There is an enumeration of sub-problem, sub-solution pairs.
→ There is only one solution possible at any time given
operator:
select(Cover & Differentiate Method)

5. If the task is to find an element in a hierarchical structure

   AND branches of the tree can be tested independently

   AND the element is in one branch only

   AND at each level in the tree it can be tested in which branch of the tree the element can be found

   THEN

   'hierarchical focusing' is a possible method for the task.

modeling-guideline *18*
modeling-goal:
method identification
task-features:

purpose
goal → Can be specified as the search for an element in a hierarchical structure.
domain
→ Branches of the tree can be tested independently
→ The element is in one branch only.
→ A each level in the tree it can be tested
in which branch of the tree the element can be found.
operator:
select(Hierarchical focusing method)

Task decomposition guidelines

1. If a task contains activities that are completely independent in time

   THEN these activities can be represented as separate tasks.

modeling-guideline *19*
modeling-goal:
task decomposition
task-features:
purpose
goal → Can be specified as the search for an element in a hierarchical structure.
domain
→ Contains activities completely independent in time.
decompose-in-time(goal)

2. If a task contains set of activities that work on independent objects THEN these activities can be represented as sub-tasks.

modeling-guideline *19*
modeling-goal:
task decomposition
task-features:
purpose
goal →
domain
→ Contains activities that work on independent objects.
operator:
decompose-in-tasks-according-to-objects(goal)

3. If the goal of a task can be split into dependent activities that have a strict order in time

   THEN those activities can be grouped into sub-tasks.

   If a task definition maps on a method but not all the method's requirements can be satisfied by the task environment

   THEN satisfaction of the requirements can be considered sub-tasks of the task.

Task distribution guidelines

1. IF a task cannot be decomposed anymore

   And requires real world activities

   THEN the task should be assigned to a human agent.

modeling-guideline *20*
modeling-goal:
task distribution
task-features:
purpose
goal →
domain
→ Requires real world activities.
operator:
assign the task to human agent

2. IF a task requires no real world interaction
And can be performed by close world reasoning
THEN the task could be assigned to a system agent.

modeling-guideline *21*
modeling-goal:
task distribution
task-features:
purpose
goal →
domain
→ Can be performed in a close world.
operator:
assign the task to system agent

1. **modelling-guideline** *22(mg-uva-1)*
modeling-goal:
task selection
task-features:
purpose → explanation creation
goal → specification of a future system property
input → given system (structure)
output → value of system property
operator:
select-task(value-prediction)

2. **modelling-guideline** *23(mg-uva-2)*
modeling-goal:
task selection
task-features:
purpose
goal → specification of future system behavior
input → a given system (structure)
output → description of system behavior
operator:
select-task(prediction of behavior)

3. **modelling-guideline** *24(mg-uva-3)*
modeling-goal:
task selection
task-features:

purpose
goal → transformation of system structure
input → formal system description
output → modified system (description)
operator:
select-task(system transformation)

4. **modelling-guideline** *25(mg-uva-4)*
modeling-goal:
task selection
task-features:
purpose
goal → specification of a system which will comply with pre-specified behavior
input → informal system description
output → description of (physical) structure component specifications
operator:
select-task(system design)

5. **modelling-guideline** *26(mg-uva-5)*
modelling-goal:
task selection
task-features:
purpose
goal → specification of a system which will comply with pre-specified behavior
input → informal system description set of components global system structure
output → global system structure filled in with components
operator:
select-task(configuration)

6. **modelling-guideline** *27(mg-uva-6)*
modelling-goal:
task selection
task-features:
purpose
goal → make a decision about a case
input → informal case description system model
output → decision class
operator:
select-task(assessment)

7. **modelling-guideline** *28(mg-uva-7)*
modelling-goal:
task selection
task-features:
purpose
goal → verify quality of system output
input → (in)formal case description system model
output → informal system description measures for quality assurance
operator:
select-task(auditing)

8. **modelling-guideline** *29(mg-uva-8)*
modelling-goal:
task selection
task-features:

purpose
goal → explain undesired system behavior
input → informal description of system behavior system model
output → system behavior explanation
operator:
select-task(diagnosis)

9. **modelling-guideline** *30(mg-uva-9)*
modelling-goal:
method selection
task-features:
domain
→ a priori likelihood of hypothesis = known
→ exhaustive hypothesis = true
→ mutually exclusive hypothesis = true
→ conditional probabilities for findings = known
→ conditional independent findings = true
→ domain size = small
expertise-model state:
selected-task(diagnosis)
operator:
select-method(sequential bayesian diagnosis)

10. **modelling-guideline** *31(mg-uva-10)*
modelling-goal:
method selection, search space reduction
task-features:
domain
→ refinement relations between hypothesis can be established
expertise-model state:
task → refinement of hypothesis
method → descend isa hierarchy
select-task(system transformation)
operator:
construct((domain structure, isa-hierarchy, hypothesis)

11. **modelling-guideline** *32(mg-uva-11)*
modelling-goal:
task selection
task-features:
purpose
goal → transformation of system structure
input → formal system description
output → modified system (description)
operator:
select-task(system transformation)

12. **modelling-guideline** *33(mg-uva-12)*
modelling-goal:
refine inference knowledge
task-features:
purpose
goal → transformation of system structure
input → formal system description
output → modified system (description)

expertise-model state:
→ select-task is observe system behavior
→ observation method = system reconfiguration
→ inference structure partially specified
operator:
insert(inference function, transform(system model)))

13. **modelling-guideline** *34(mg-uva-13)*
modelling-goal:
refine inference knowledge
task-features:
domain
→ history data = available
→ history period = long
→ distribution of hypothesis likelihood = uneven
task-model-state:
selected-task(hypothesis selection history data = available)
operator:
insert(inference function,
order(hypothesis-set [2]))
add (likelihood estimates,
–¿ hypothesis)
operator:
select-task(system transformation)

14. **modelling-guideline** *35(mg-uva-14)*
modelling-goal:
method selection
task-features:
domain
→ history data = available
→ history period = short
→ distribution of hypothesis likelihood = uneven
task-model-state:
selected-task(hypothesis selection history data = available)
operator:
select(method, calculate(frequency of correctness))
operator:
select-task(system transformation)

15. If explaining-decisions = conclusions THEN method = conclusion-explanation
16. If explaining-decisions = inferencing THEN method = inference-explanation
17. If explaining-decisions = strategy
THEN method = strategy-explanation

8.3 Additional modelling guidelines from Drilling Fluid example

Below is a small set of modelling guidelines derived from the Drilling Fluid example, described in appendix B. The object structures are here presented in the more extended form, as given by the template defined in chapter 2.

modelling-guideline *36*

sub-type-of:
 method selection
task-features:
 goal
  → learning from experience
  → suggesting a plausible problem solution
 domain
  → no strong domain theory available
expertise-model-state:
 method knowledge is empty
operator:
 select-method(Case based reasoning)
next-operator:
 refine(method)

modelling-guideline *37*
sub-type-of:
 method refinement
task-features:
 domain
  → descriptive domain knowledge available
  → case similarity assessment is semantic
expertise-model-state:
 method knowledge is case based method
operator:
 select-method(Knowledge intensive Case based reasoning)
next-operator:
 combine(case based reasoning, model-based reasoning)

modelling-guideline *38*
sub-type-of:
 domain model selection
task-features:
 domain
  → no strong domain theory available
  → heursic associations available
  → deep domain relations available
operator:
 select-model(combined shallow and deep domain model)
next-operator:
 refine(domain knowledge model)

modelling-guideline *39*
sub-type-of:
 task refinement
task-model-state:
  → we are dealing with a fault finding task
  → we are dealing with a repair task
  → there is interleaved goal achievement
expertise-model-state:
 task knowledge is partially saved
operator:

select-task(combined fault-finding and repair task)

next-operator:

select(method knowledge)

Chapter 9

Using the Inventory in Modelling - A Scenario

9.1 General

This chapter illustrates the use of the task feature framework - including the use of task features and modelling guidelines. This is done by going through some steps in the modelling process, starting from an empty expertise model, selecting initial library components, feeding back to the selection of other task features, and extending and modifying the expertise model.

Our scenario application is the Hospital Meal Planner (see appendix A for a protocol transscript of this experiment).

9.2 Overview of the hospital meal planning application

This is an example application 'invented' during the work of this task. Below, a functional description is summarized (described in more detail in appendix A).

A decision support system is wanted by the chief-of-kitchen in a general hospital with appx. 2000 patients. He wants active computer assistance with the job to set up the weekly meal plan for the hospital. A meal plan is constructed on each Friday for the following week. He wants the system in order to do his job quicker, but also to compose more optimal meals - i.e. better quality meals with less costs.

Meals are served three times a day, at fixed hours, and named breakfast, lunch, and dinner, respectively. A meal consists of a few dishes. There exists a description of a top level structure of a meal, and lists of possible parts of meals are available. There are hot and cold meals, and lunch is the only hot meal. Drinks are served with all the meals. Modifications of a patient's meal plan during the weak is sometimes necessary due to medical reasons. When new patients come into the hospital, their meals during the rest of the week period are selected according to a simpler procedure.

The goal of the meal planning work is to come up with an assignment of meals to each patient of the hospital on a weekly basis. The meal plan of each patent has to confirm with certain requirements regarding nutritional value, niceness, variability, uniformity of meals, costs, and - of course - diet requirements. The chief-of-kitchen performs the meal planning work in his office, which is a side-room to the kitchen. Here he has easy access to the information he needs, such as what food resources there are, financial budget data, etc., and he can also easily discuss things with the kitchen personel, most notably the chef.

The information which is taken as input to the meal planning work is a list of patients and their diet needs (comes from the medical departments), the hospital's collection of meal recipes, type of food in storage, and food prices. The information which comes out of the process is a weakly meal plan for each patient. This list shall also be sorted according to types of food and drinks, so the kitchen and the medical departments can check whether something needs to be bought. The chief-of-kitchen makes a list of ingredients to be ordered, a list to each of the departments: kitchen, medical, buying. The food inventory management is not regarded as a part of the meal planning process, and is outside the scope of the intended decision support system.

The collection of meal recipes - the cookbook - is also available as a searchable database. The cookbook describes single dishes as well as meals. The kitchen does not always stick to the recipes as described, it sometimes make new combinations. The decision support system should not make new recipies.

The meal planner system shall give support to the chief-of-kitchen by suggesting a weakly meal plan for the hospital. It shall also provide a facility for editing a plan, and be able to check an existing meal plan (for example a plan which has been changed by the user, or a new dish which has been composed) according to further specified completeness and consistency criteria. The system must be able to interact with its user by:

- Allowing convenient entry and editing of patient data. Often, entry of data will be in the form of modifications of the data for previous week.
- Pointing out deficiencies in a plan, and asking the user for additional, or revised, information.
- Justifying its choices. For example, if the user doubt the composition of a dish or a meal, the system shall produce an explanation for its decision. This may be done just by referring to the information and the knowledge used, but it should also be able to show *how* the knowledge was used, and - if possible - also *why* this knowledge or information was used instead of other, i.e a strategic explanation.

The system should work as an independent decision support system, not integrated into the patient journal system of the hospital. However, it should be able to read selected patient data from tape or disk, as well as to access the cookbook database.

9.3 Functional requirements definition

The goal of the meal planning work is to come up with an assignment of meals to each patient of the hospital on a weekly basis. The meal plan of each patent has to confirm with certain requirements regarding nutritional value, niceness, variability, uniformity of meals, costs, and - of course - diet requirements.

The input to the task feature analysis is a functional requirements definition and access to the task reality. The output of the analysis is a set of generic components that can be interpreted as a high level software design.

9.4 Analysis with the modelling guidelines

The analysis starts by applying task identification and task decomposition guidelines. As soon as tasks can be identified the method identification guidelines can be applied.

The task of satisfying the functional requirements can now be decomposed in different ways, bu trying task decomposition in time (decg # 1) or by trying decomposition in terms of independent objects the task work on (decg #2)

- Identify breakfast schedule, lunch schedule and dinner schedule. This is a decomposition in the time dimension. It reduces the complexity by partitioning the search space of possible meals. The disadvantage is that the costs and the nutritial value are computed over weeks respectively. days, thus the recognized tasks (and search partitions) are still mutually dependent.

- Identify the task for the different diet types. The resulting tasks work for different objects (the patients). These tasks are mutually independent. It reduces the complexity by limiting the planning task to a single type of diet. This limits the search space in some extend.

The second type of decomposition seems the best because it actually reduces the search space. The resulting tasks can now be mapped on the task identification guidelines.

9.5 Task identification

The task identification guidelines help to find a label for a set of required (during high level design) or observed (during analysis) activities. The requirement is to make a plan consisting of a set of dishes for a number of patients having one specific diet for the whole week. Here we go in a top down, hypothesis driven way through the guidelines.

This activity cannot be mapped on a diagnosis task (tig #1) because we are not looking for a malfunction or unexpected behavior in a system.

The activity can also not be classified as an assessment task because the we are not trying to find an evaluation of a case (tig #4).

The task can be considered to find a complete specification of a set of ‘primitive’ components and their relations to meet a set of constraints (tig #7). The design requirements are not ill specified and perhaps there are known decompositions. Thus It seems not to be a class I design task. A failure analysis for a proposed meal plan is not especially hard, neither synthesis. Therefore it is not class II design task. There is a fixed (but very large) set of possibilities. The resulting selection can be evaluated. Therefore the task can be classified as a class III design task. The elements of the state cannot be evaluated completely independently. Therefore the most specific classification we can arrive at is a class III-b design task (tig #13).

The task is not to combine a number of activities in order to create a sequence with certain attributes. Therefore it is not a scheduling task (tig #16). The task is also not a collection task (tig #17).

We can conclude that the the activities required can be classified as a class-III-b design task. This generic model entity can now be translated in the specific terminology for the meal planning task.

Meal planning as Class III-b design

Input A set of requirements; these translate into the constraints of the meal planning task, i.e. costs etc.

Output A design that satisfies the requirements; this translates into the weekly schedule that satisfies the constraints.

Goal

9.6 Method identification

The Method identification guidelines help to select a generic model for the task identified. Here we perform again a top down, hypothesis driven search through the modelling guidelines. We formulate the value of a feature and then list in what extend it matches the required task feature of the method indentification guideline. A '+' means a comple match with between the task-reality and the task feature, a '+/-' a near miss and a '-' a mismatch. A mismach does not mean automatically that the method is not applicable, beause a task can be added to make the required knowledge available (task decomposition guideline #3).

9.6.1 Propose & Revise (mig #1)

1. the constraints are known in advance (+)
2. there is not a set of partial solutions (-)
3. there is not a set of solution revisions (-)
4. partial solutions can be rejected but not confirmed (+/-)
5. remedies can be specified for each violated goal constraint (+)
6. the level of conflict is 'reasonable', i.e. the expert cook is able to solve it. (+).

It appears that this method can be applied if we can find a set of partial solutions and a set of solution revisions. The partial solutions can perhaps be retrieved from the expert cook. The knowledge engineer can try to identify a set of solution revisions.

9.6.2 Means ends Analysis (mig #2)

In order to evaluate the applicability of Means Ends analysis the problem must first be mapped onto the elements of the state space metaphor, which are states, operators, goal state and start state [Duursma, 1992a]. Means ends analysis adds an extra requirement; the availability of a distance metric between any state and the goal state.

1. The task can be formulated in terms of states, where states represent partially finished menu's. (+)
2. Addition of a dishes to a menu can be seen as operators working on the states. (+)
3. The starting state is an empty menu of the week. (+)
4. The goal state is a menu that satisfies all the cost, diet, nutritial value constraints. (+)
5. The distance metric can be considered the percentage of the meals of the weak covered, but this takes not into account whether all requirements have been satisfied in this extend as well. Therefore the distance metric needs to be formulated in a way that takes the fullfillment of the other requirements into account as well. This is quite difficult. (-)

A problem with the application of means ends analysis to perform this activity is that there is no elegant way to model the constraints on the solution. But in principle the model seems applicable.

9.6.3 Extrapolate from a similar case (mig #3)

Extrapolate from a similar case is a quantitative case based approach. Its traditional application task is 'sizing' (see [McDermott, 1988; Duursma, 1992c]).

The dish selection example can in fact be translated to a quantitative description. Each menu can be specified in terms of amounts of mony, nutritional values and amounts of ingredients specified by diets.

1. A source case can be a patient with a certain diet and a certain insurance connected with the solution consisting of a week schedule. (+)
2. A set of source cases can be retrieved from the expert (the cook). (+)
3. Similarities among the diet, insurance, preference of a patient imply similarities over the weekly meal plan. (+)
4. the Simlarities between the cases and the solutions are not completely constant over time. If food prices change over th eyear than the solutions for the source case will change as well (+/-)

5. No deep explanations are necessary why a certain weekly schedule has been proposed. (+)

Apparently the 'extrapolate from a similar case' approach is also applicable on the meal planner example.

9.6.4 Cover & Differentiate (mig #4)

Cover and differentiate tries to cover a set of symptoms with a set of explaining theories, and then tries to find symptoms to differentiate between the theories by asking for additional symptoms.

1. The problem states of the meal planner can be considered the different possible weekly schedules. However, it is practically impossible to provide an enumeration of these. (-)
2. The possible symptoms can be the patients, with diets, types of insurance contracts etc. The types of patients are enumerable, but not the exact amount of them. (+/-)
3. A static pre-enumeration of all explanations maps in the meal planner on a static enumeration of all weekly schedules. This is possible per patient like in the previous item. This might however conflict with the criteria of variable weekly schedules (not a boring repetition). (+)
4. There is information to differ different weekly schedules. But there is not always a best solution because of the variety required. (+)
5. One candidate weekly schedule is applicable at any time per event (patient). This is not the case; several weekly plans may be applicable (-).

9.7 Evaluation of selected generic components

In the previous section we applied the model selection guidelines in order to select a set of generic components. It appeared that of the few models available two were 'interesting' solutions;

- **Propose & Revise**

  P&R is the computational approach to the problem; solutions are actually created by a constraint satisfaction algorithm out of pre-shaped, abstract solutions provided by the expert. The amount of pre-shaping necessary depends on the level of conflict between the constraints, the size of the search space and the hardware available. This generic model seems suitable enough for further elaboration, i.e. a complete mapping and translation, like in [Duursma, 1992c]. An advantage of this approach is that the solutions always based on up-to-date data, which make adaptation to a changing environment easy, as long it is within the limits of the pre-shaped solutions.

- **Extrapolate from a similar Case.**

  ESC is the re-use and adaptation of previous solutions approach to the problem; solutions are not created out of their constraints, but previous cases are re-used and modified if necessary to fit the new problem. All requirements of this approach mapped on the meal planner domain, thus this model seems very suitable. The only problem might be when the environment changes in such extend that the old solutions fail to be applicable. This can happen for instance when prices of certain ingredients vary according to season. Then specific cases for specific seasons should be provided. If unforeseen changes in the environment happen; then the system should go back to a learning mode in which the the expert has to provide new, updated cases.

Chapter 10

Conclusion and Future Work

The goal of Task I.5 is to come up with a method, including knowledge engineering strategies and supporting tools, for how task features can be used to identify and utilize library components in building an expertise model. The objective is to come up with an inventory of such features, and a set of guidelines for how they can help to make the best possible use of the *Common*KADS library in developing knowledge-level models.

Underlying the present work is a model of modelling as search. This provides us with a powerful framework and a terminology for discussing task features. In particular the notions of 'task reality', 'modelling operator', 'modelling guideline' and 'task feature' are related to eachother in a coherent framework. This has been described in Part I of this document.

We have started to fill in this framework with concrete task features, modelling operators and modelling guidelines. The inventory in Part II of this document is, though surely incomplete, a rather extensive list of practical modelling support. To bring these results to the end-user and to integrate them properly into the overall *Common*KADS framework the following remain 'hot topics':

Modelling methods: task features have to be used mainly in the context of the *Common*KADS library. The work on library and task-features has been going on in parallel and the both tasks have influenced eachother greatly. The library is often perceived to be complex and we need to demonstrate that it isn't really. In particular we need to capture modeling methods into sets of modelling guidelines at a higher level of abstraction.

Tool support: use of large sets of modelling guidelines is only possible with adequate tool support. We have implemented and demonstrated a first modelling support tool. We feel, however, that modelling support should be more tightly integrated with the workbench facilities of libraries and model editors. The whole modelling framework that we use in this deliverable and in [Duursma et al., 1992] is applicable to any model in the *Common*KADS model set so that generic support can be envisaged. This would greatly increase the scope of the workbench [Van de Velde, 1992]

Validation: work on validation and verification has been less emphasized but it is clearly important. Now that the work on the library [Aamodt et al., 1992], the model set [deHoog et al., 1992], the life cycle [Taylor et al., 1992a; Taylor et al., 1992b] and quality assurance in *Common*KADS [Williams et al., 1992] is stable a new approach toward validation seems feasible. n particular we have argued that it seems worthwhile to explore the connection between validation and quality assurance in somewhat more depth.

References

9001, I. (1987). Quality systems - model for quality assurance in design/development production and servicing.

Aamodt, A. (1990). A computational model of knowledge-intensive learning and problem solving. In *Wielinga et. al: Current trends in knowledge acquisition (Proceedings of EKAW-90).* IOS Press.

Aamodt, A. (1991). *A knowledge-intensive approach to problem solving and sustained learning.* PhD thesis, University of Trondheim, Norwegian Institue of Technology, Trondheim, Norway.

Aamodt, A., Bredeweg, B., Breuker, J., Duursma, C., Lockenhoff, C., Orsvarn, K., Top, J., Valente, A., and de Velde, W. V. (1992). The commonkads library. ESPRIT Project P5248 KADS-II, Deliverable D1.3 KADS-II/T1.3/VUB/TR/oo5/1.0, ECN, SICS, SIEMENS, UvA, VUB.

Adrion, W., Branstad, M., and Charniavsky, J. (1982). Validation, verification and testing of computer software. *ACM Computing Surveys*, 14(2), 159–192.

Ayel, M. and Laurent, J.-P. (1991). *Validation, Verification and Test of Knowledge-Based Systems.* Wiley Professional Computing. Wiley, Chichester, UK.

Barrett, B. (1990). A software quality specification methodology for knowledge based systems. In *Proceedings of the Workshop on Knowledge-Based Systems: Verification, Validation and Testing, AAAI'90.*

Belis, I. (1992). Taak analyse tool: Een hulpmiddel voor het ontwerpen van kennis systemen. Master's thesis, VUB AI laboratory.

Bensabat, I. and Dhaliwal, J. (1989). A framework for the validation of knowledge acquisition. *Knowledge Acquisition an International Journal*, 1, 215–233.

Brender, J., Larsen, H., Lopez, B., Meseguer, P., Nonfjall, H., Plaza, H., Vaudet, J., and Vesoul, P. (1989). Kbs validation: State of the art. Technical Report Deliverable 1, VALID Project.

Brown, D. C. and Chandrasekaran, B. (1989). *Design Problem Solving - Knowledge Structures and Control Strategies.* Research Notes in Artificial Intelligence. Pitman, London.

Caekenberghe, S. V., Steels, L., Bogemans, D., and Gilbert, F. (1991). *KAN reference manual.* VUB AI laboratory, Pleinlaan 2 Brussels. AI Memo 91-5.

de Velde, W. V. (1991). Definition task i.5. ESPRIT Project P5248 KADS-II, Task Definition TI.5 KADSII/T1.5.VUB/TD/002/1.0, VUB.

deHoog, R., Martil, R., Wielinga, B., Taylor, R., Bright, C., and de Velde, W. V. (1992). The commonkads model set. Technical Report ESPRIT Project P5248 KADS-II/WP1-2/TR/UvA/0040/4.0, The KADS-II Consortium.

Duursma, C. (1992a). Interpretation models and problem solving methods. Vrije Universiteit Brussel, AI-Lab.

Duursma, C. (1992b). Interpretation models and problem solving methods. Technical Report AI-Memo 91/11, VUB AI lab, Brussels, Belgium.

Duursma, C. (1992c). Role limiting methods for the cm library. Technical Report AI-Memo 92/4, VUB AI lab, Brussels, Belgium.

Duursma, C., Schreiber, G., Terpstra, P., and de Velde, W. V. (1992). Operationalising knowledge models. ESPRIT Project P5248 KADS-II, Deliverable D1.2.3 KADS-II/T1.2/VUB/TR/004/1.0, UvA, VUB.

Green and Keyes (1987). Verification and validation of expert systems. In *Proceedings of Western Conference on Expert Systems*, 38–43. Anaheim, CA.

Hoppe, T. and Meseguer, P. (1991). On the terminology of vvt. In Jenkins, P. and Plaza, E. (Eds.). , *Proceedings of the European Workshop on the Verification and Validation of Knowledge Based Systems*, Jesus College, Cambridge, UK.

Klinker, G., Bhola, C., Dallemagne, G., Marques, D., and McDermott, J. (1991). Usable and reusable programming constructs. *Knowledge Acquisition*, 3, 117–136.

McDermott, J. (1988). Toward a taxonomoy of problem solving methods. In Marcus, S. (Ed.). , *Automating Knowledge Acquisition for Expert Systems*. Kluwer Academic Publishers.

Meseguer, P. (1992). *Validation of Multi-Level Rule-Based Expert Systems*. PhD thesis, Universitat Politecnica de Catalunya.

Motta, E., Rajan, T., Domingue, J., and Eisenstadt, M. (1990). Methodological foundations of keats, the knowledge engineer's assistant. In Wielinga, B. (Ed.). , *Current Trends in Knowledge Acquisition*. IOS Press.

Rushby, J. (1988a). Quality measures and assurance for ai software. SRI Project 4616 SRI-CSL-88-7R, Stanford Research Institute.

Rushby, J. (1988b). Validation and testing of knowledge-based systems. how bad can it get? In *Proceedings of the Workshop on Knowledge-Based Systems: Verification, Validation and Testing, AAAI'88*.

Schrooten, R. (1992). Selection of generic models. Technical report, VUB, Licentie Thesis.

Sommerville, I. (1989). *Software Engineering*. Addison Wesley, Wokingham, UK. 3th Edition.

Steels, L. (1991). Commet: A componential methodology for knowledge engineering. CONSTRUCT Report D/WP2/2, VUB AI-Lab.

Suen, C., Grogono, P., Shinghal, R., and Coallier, F. (1990). Verifying, validating and measuring the performance of expert systems. *Expert Systems with Applications*, 1(2).

Taylor, R., Bright, C., Menezes, W., and Groth, J. (1992a). Life-cycle model. volume one: Principles of the lcm. Technical Report ESPRIT Project P5248 KADS-II/T2.1/TR/TRMC/010/1.0, Touche Ross & Lloyd's Register & Cap Gemini Logic. Deliverable D2.2, volume 1 0f 2.

Taylor, R., Bright, C., Menezes, W., and Groth, J. (1992b). Life-cycle model. volume two: A guide to the commonkads development process. Technical Report ESPRIT Project P5248 KADS-II/T2.1/TR/TRMC/010/1.0, Touche Ross & Lloyd's Register & Cap Gemini Logic. Deliverable D2.2, volume 2 of 2.

Tomlinson, C. (1991). Definition proposals for validation and verification. KADS-II technical Note T1.5.5.

Van de Velde, W. (1992). Re-orienting the commonkads workbench. Esprit project p5248 kads-ii, wp1-wp3 working paper, VUB.

Williams, D., Tomlinson, C., Bright, C., and Rajan, T. (1992). The commonkads quality viewpoint. Technical Report ESPRIT Project P5248 KADS-II/T2.2/TR/LR/0040/1.0, The KADS-II Consortium.

Appendix A

Description of the Hospital Meal planner Case

This chapter describes an example application that has been used for focusing the discussions in TI.5. The example has been used to identify task features, and hence provided input to the bottom-up analysis of task features classes and how they may point to modelling items in the CommonKads library.

A.1 Original Example

Suppose you are running the kitchen of a hospital. You have to cook dinners for patients with different diets expressed in what they should NOT have. But they all should eat in a healthy way. Moreover you want to make nice meals (not annoying). Finally it should be as cheap as possible.

There is a list of parts of dishes available (boiled potatoes, spaghetti, salad, yoghurt), there is a list of patients and their diets and here is also a list of classes of ingredients the patients should not have (like sugar, salt, fat).

The Task is to build a system that has has input the number and types of patients and as output a schedule of breakfasts, lunches and dinners for a whole week.

A.2 Extended Example

Here is an extended version of the example, based on the transcribed protocol, included in chapter 3. This example also contains some information which is not explicitly given in the protocol. This additional information is partly information which was given before the protocol recording was started, partly based on intelligent guessing in order to fill in some holes. In a real situation, of course, such holes would have lead to a new interaction with the expert, as part of the normal protocol analysis procedure. The purpose of the description given here is to serve as a starting point for the derivation of task features as potential pointers to CommonKads modelling components. The protocol analysis procedure as such is therefore not emphasized in this example. In order to distinguish information given in the transcribed protocol form that in the elaborated textual description, reference numbers pointing to lines in the protocol are included in the text.

KAHMP - A Knowledge-Based Assistant for Hospital Meal Planning

The Level Egdelwonk hospital is a Belgian general hospital with approx.. 2000 patients [46]. The chief-of-kitchen of this hospital wants a computer system that can help him [1] with the difficult and time consuming job to set up the weekly meal plan for the hospital. A meal plan is constructed on a Friday for the following week. Currently, it takes almost a whole day to work out the complete meal plan [45]. Hence, he wants this system in order to do his job quicker, but also to compose more optimal meals - i.e. better quality meals with less costs.

Meals are served three times a day, at fixed hours, and named breakfast, lunch, and dinner, respectively. A meal consists of a few dishes [27]. There exists a description of a top level structure of a meal [29], and lists of possible parts of meals are available [26]. There are hot and cold meals [31], and lunch is the only hot meal [31]. At breakfast two dishes are served: Bread, and a desert - normally fruit [32]. At lunch - which is the main meal - three dishes are served: A starter, the main course, and a desert [33]. The dinner is a light meal with two courses: Bread with cheese or meat, and desert [34]. Drinks are served with all the meals.

Modifications of a patients meal plan during the weak is sometimes necessary [4] due to medical reasons. When new patients come into the hospital, their meals during the rest of the week period are selected according to a simpler procedure.

The goal of the meal planning work is to come up with an assignment of meals to each patient of the hospital on a weekly basis. The meal plan of each patent has to confirm with certain requirements regarding nutritional value, niceness, variability, uniformity of meals, costs, and - of course - diet requirements.

Nutritional value is a measure of general healthiness of a meal, and is based in the contents of nutritional ingredients [18,19] such as proteins, carbohydrates, vitamines, fat, etc. in various food types. It is an absolute measure, not related to the patient [21]. Total nutritional value is the sum of the nutritional value of the ingredients [20]. Such values are defined in health standards of the Belgium Health Association [14], and expressed in terms of minimum and maximum amounts of various nutritional ingredients per. day [14,15].

Niceness of a meal is an indication of taste and visually appetizing features. Taste is a subjective matter, and is evaluated from the chief-of-kitchen's personal experience. Visual appeal is related to a single dish, and determined by colour, texture, and preparation of the food. Niceness is related to the overall meal schedule [17], not (only) a single meal.

Variability expresses nutritional value over time. There are two time periods for which variation is assessed: A day and a week.

Uniformity (similarity of meals [44]) is a requirement which may pull in the direction opposite to variability. This requirement is motivated by the efficiency - and hence the costs - of the kitchen procedures. So, satisfying the variability requirements for each patients should be done with as few types of dishes as possible.

The costs has to confirm with budgets on a monthly basis [13].

Diets are restrictions on what a patient can eat. Diet requirements has to be satisfied, and has precedence over all other requirements.

The chief-of-kitchen performs the meal planning work in his office, which is a side-room to the kitchen. Here he has easy access to the information he needs, such as what food resources there are, financial budget data, etc., and he can also easily discuss things with the kitchen personnel, most notably the chef. The chief-of-kitchen is a former experienced cook [2], who was appointed to his present position [2,3] two years ago. He therefore also has some management knowledge [3].

The information which is taken as input to the meal planning work is a list of patients and their diet needs [4] (comes from the medical departments [8]), the hospital's collection of meal recipes [22], type of food in storage, and food prices. The information which comes out of the process is a weakly meal plan for each patient [5]. This list shall also be sorted according to types of food and drinks, so the kitchen and the medical departments can check whether something needs to be bought. The chief-of-kitchen makes a list of ingredients to be ordered [5], a list to each of the departments: kitchen, medical, buying [6,7]. The food inventory management is nor regarded as a part of the meal planning process, and is outside the scope of the intended decision support system.

The collection of meal recipes - the cookbook [22] - is also available as a database. The cookbook describes single dishes as well as meals [38]. For example, it shows how to cook potatoes, make mayonnaise, apple pie, etc. [25], and also what to serve with some ingredients (e.g. potatoes) [39]. It also contains dish associations such as the niceness of a particular dish. The kitchen does not always stick to the recipes as described, it sometimes make new combinations [23]. The decision support system should not make new recipes [24]. The cookbook should, at least, be accessible by the decision support system, possibly a part of it [40]. The cookbook is updated every two years [41].

EXAMPHLE shall give support to the chief-of-kitchen (i.e. the user) by suggesting a weakly meal plan for the hospital. It shall also provide a facility for editing a plan, and be able to check an existing meal plan [36] (for example a plan which has been changed by the user, or a new dish which has been composed [36]) according to further specified completeness and consistency criteria. The system must be able to interact with its user by:

- Allowing convenient entry and editing of patient data. Often, entry of data will be in the form of modifications of the data for previous week.
- Pointing out deficiencies in a plan, and asking the user for additional, or revised, information.
- Justifying its choices. For example, if the user doubt the composition of a dish or a meal, the system shall produce an explanation for its decision. This may be done just by referring to the information and the knowledge used, but it should also be able to show *how* the knowledge was used, and - if possible - also *why* this knowledge or information was used instead of other, i.e a strategic explanation.

A successful system will each day ensure that correct dishes are suggested for each patient, i.e. no diets restrictions are violated [9], that the meals are kept within the budget [10], and that the meals are healthy [12] and nice [11].

The system should work as an independent decision support system, not integrated into the patient journal system of the hospital. That is, EXAMPHLE will not have direct access to the patient files [42], or the inventory and storage database [43]. However, it should be able to read selected patient data from tape or disk. The management and maintenance of the cookbook database will not be handled by the EXAMPHLE system.

A.3 The Protocol Transcript

```
     ? Where will the system be used    ?
 1   > It will be used by the head of the kitchen.

     ? Is the user an expert?
 2   > Has been a cook but has been promoted.

     ? Doe he understand the domain?
 3   > yes and has also some management knowledge

     ? Information about the patients?
 4   > Every week a list of patients with their diets. Modifications
       may occur during the week.

     ? What is the output?
 5   > I give the cook a plan what he should
       (recipe) cook every day in the next week. And I also make a list of
       ingredients to be ordered.


     ? Order list is for different part of the organization?
 6   > Yes for the

     ? Three departments
 7   > Kitchen, Medical (Care), Buying.

 8   > Patients list from medical staff.

     ? How do you know the schedule works; what makes it a good schedule?

 9   > 1. Every day the correct dishes (i.e. no diets are violated).
10   > 2. Limited budget, so keep within the budget.
11   > 3. Nice meals (it's a good hospital).
12   > 4. Healthy food.

     ? Elaborate on the constraints:
```

? Budget on a week, year?
13 > On monthly basis.

? How can you judge the variation of a meal.

? Elaborate on the goal; nice and healthy. Goal is to schedule meals
that are within the budget, enough variation ..

? How would the expert define a healthy meal.

? What does one wants to achieve?

? What is nice and healthy.

14 > Healthy is standards of Belgium health association; expressed in
minimal and maximal amounts of ingredients per day. Categories
of food ingredients. S

? Should be present within one meal.

15 > Should be present within one day.
16 > Ingredients? Composition of meals?

? Types of ingredients. Nutricial value.

? Niceness is coupled to the schedule?
17 > Yes.

18 > Nutricial Values.
? NV is property of ingredient?
19 > Yes.

20 > NV of a meal is the sum of the ingredients of the meal. The way of
preparing is not of relevance here.

? Is relative to patient or absolute?
21 > Absolute.

22 > There is also a book of recipes.

? Do you stick to this book?
23 > We make also new combinations.

? Do you expect the system to compose new recipes? Do you want to
check this?

24 > I don't want the system to make new recipes.

? The recipes somehow to constrain composition of ingredients.

25 > There are recipes to make certain type of ingredients; like
potatoes, mayonnaise, apple pie.

? Grain size. I think we have to stick to complete dishes as sub part
of a meal.

26 > Lists of parts of a meal are available.

27 > A meal consists of a few dishes.

28 > Are we going to have the system also combining spaghetti with
potatoes (as side dish) (expert in trouble).

? Is there a top level structure of a meal
29 > yes, there is such structure.

? A meal is a frame with slots that can be filled with different
dishes
30 > What is a slot.

? First dish, main dish etc.

? Are there constraints of combinations of parts?

31 > There are hot meals cold meals. Breakfast and dinner are in
principle just bread. Lunch is the only hot meal.

? AH there are three types of meals.
32 > Breakfast has bread and a drink.
33 > Lunch has starter, main, sweet.
34 > Dinner is bread with cheese or some meat, some fruit etc.

? And now

35 > Combinations do not really matter for niceness.

? Refocus on What system should do.
? You want the system to suggest a schedule. Do you want to put in
some information and then get something out?

36 > I want to be able to check every type of combination the system
proposes. I want to evaluate the composition. I wan tot have nice
meals, thus I want to be able to check.

? If you alter it you have to check again the criteria?

37 > We have also this cook book. Only new combinations need to be
evaluated.

? Your cookbook provides dishes, not meals.

? Is it about meals or dishes.

38 > It provides both.

39 > Provides how to cook potatoes, and also What to serve with potatoes.

? The cookbook should be in the system?
40 > I assume so

```
    ? Does the cookbook ever change.
41  > Update every two years.


    ? Should the system be couples to the central system of the hospital.
42  > No.

    ? Planning, storage has that to be linked?
43  > No we have a separate Database system.

    ? He wants to check generation of new combinations. So it is not just
      a simple what if facility. You want to be able to edit what the
      system comes up with and then calculate again.

    ? Writes up subtasks, generate, check , edit.

    ? The problem of this example is that we now too much of it.

    ? The problem is that we have no good enough idea of the goal; what
      makes this problem.

    ? Limited capacity of kitchen and cook.

44  > Yes, I forgot, it should be as cheap as possible but also as much as
      similar dishes as possible. Otherwise the cook gets mad.

    ? Why build this system
45  > Saves me one day a week; to save money, better quality check.

    ? How many patients
46  > 2000
```

A.4 Task Analysis

A.5 Analysis: Starting situation

A.5.1 Task analysis

In this section we present a task analysis of the dish selection example.

Kitchen running involves the management, planning and actual work of the Hospital kitchen.

1. meal planning
   (a) get requirements
      i. get diets
         Goal Get the diets for coming week.
         Input None
         Output Number of diets for each of the days in the coming week. This is a paper list that has been prepared by dietian in collaboration with doctors.
         A. get nr of low salt diets. This information is retrieved from the diet list by counting the number of crosses. It is handled by the kitchen administration.
         B. get nr of low calory diets. idem low salt.
         C. get nr of normal diets. idem low salt.

ii. get food stock.
Goal Get a list of food in stock.
Input None
Output A list of articles and their 'best before' dates.

iii. get diet budgets.
   A. get normal diet budget.
   B. get low salt diet budget.
   C. get low calory diet budget.

(b) Select recipes
   i. select breakfast
      A. select first dish
      B. select second dish
      C. select drink
   ii. select lunch
      A. select starter
      B. select main lunch dish
      C. select dessert
      D. select drink
   iii. select dinner
      A. select main dish
      B. select dessert
      C. select drink
   iv. select snack

2. administration.
   (a) compute total nutritional value
   (b) divide budget
      i. divide budget between diets
      ii. divide budget between over days
      iii. divide budget over meals
      iv. divide budget over courses
   (c) compute total nutritional value
   (d) divide nutritional value
      i. divide nutritional value between diets
      ii. divide nutritional value over days
      iii. divide nutritional value over meals
      iv. divide nutritional value over courses
3. get food prices.
4. Food preparation assessment.
   (a) Pre cooking Preparation time assessment.
   (b) Cooking time assessment.
   (c) Vulnerability of food.
   (d) Ease of serving.
5. Preparation task distribution
   (a) Assign preparation of Normal diet
   (b) Assign preparation of low salt diet
   (c) Assign preparation of low calory diet.
6. Actual preparation and cooking.
7. Ingredients ordering.
8. Adjustments according to feedback

A.5.2 Object Analysis

Object *Daily Meal*

Attributes Nutritional value: Sum of the nutritional values of the parts,
Cost: Sum of the costs of the parts.
Parts: Breakfast, Lunch, Dinner, Snack.

Object *Breakfast*

Attributes Nutritional value: Sum of the nutritional values of the parts,
Cost: Sum of the costs of the parts.
Parts: breakfast entry, breakfast main, drink.

Object *Lunch*

Class Meal.

Attributes Nutritional value: Sum of the nutritional values of the parts,
Cost: Sum of the costs of the parts.

Relations

Parts Lunch starter, Lunch main course, Lunch dessert, Lunch drink.

Object *Breakfast*

Class Meal.

Attributes Nutritional value: Sum of the nutritional values of the parts,
Cost: Sum of the costs of the parts.

Relations

Parts breakfast entry, breakfast main, drink.

Object *Diet*

Class

Attributes Diet type: one of low_salt, low_calory, normal.
limits: List of limits to certain nutritional values (?)

Object *Recipe*

Class

Attributes Dish type: one of first dish of breakfast, second dish of breakfast, main course lunch etc.
Ingredients: List of ingredients required by the meal.
Nutritional value: List of nutrition/quantity pairs.

Object *Patient*

Class

Attributes Diet: The type of diet, one of normal, low salt, low calories.
Insurance class: The type of insurance, A, B or C.
Preferences: The types of dishes the patient likes or not.

A.5.3 Task Feature analysis

Here we analyze a few tasks to see whether we can come up with some useful task features.

A.5.3.1 Task Analysis

Task *select breakfast drink*

agent cook

input (role: Requirements) Available drinks, Diet of Patient, Cost of Drink, Type of Insurance of patient, Nutritional value of drink, Nutritional value of rest of Meal, Preference of Patient.

output a drink

goal
1. Selected drink is an element of the available drinks
2. Nutritial value of drink together with other food that is part o the breakfast is within diet limits.
3. Cost of the drink is together with all other food costs within the limits of the day for the Insurace.
4. Drink is within preference limits of the patient

Strong requirements (may not be violated): Diet of patient. (This should be related to the goal of the task).

dependencies
1. Max nr of possible drinks: 10
2. Max nr of Diets: 3. (primitive)
3. Types of insurance: 3 (primitive)
4. Nutritional value of meal: (derived)
5. Nutritional value of drink: 1 (primitive)
6. Preference of patient: (primitive).
7. type of selection criteria: Complex dependencies.

regularity high. (a patient usually takes the same drink every morning, only after special events this might change.

Although an expert will have made a task-decomposition, there is no obvious task decomposition of this task for the knowledge engineer. A first characterization of the task can however be made:

- Goal element is an element of a pre-listed set.
- Goal element has to satisy a number of requirements
- The requirements are dependent on the selection of the element.

It appears that the task is dependent on the cost of the food of the complete week and the nutritial value of the other food of the day.

When analyzing the other diet selection tasks, we see that they have similar constraints and that they are all interdependent; no task can be solved without solving also the other tasks. Therefore it is not possible to apply a different problem solving method on each of the different sub-tasks.

Thus the 'select-drink' task cannot be solved in this decomposition. Another decomposition need to be tried. At the 'day' level the costs of the meals can all be determined. At the 'meal' level the nutritial value of the meals can be determined. If we apply the task recognition guidelines we can see that the day planning task can be recognized as a design class III b task.

Consequently the task we are going to investigate is

Task *Make day plan*

agent: cook & system

input Available drinks, Nr of Patients, Diet of Patients, Cost of Drink, Type of Insurance of patient, Nutritional value of drink, Nutritional value of rest of Meal, Preference of Patient.

output A day meal plan (see object description for definition).

goal
1. Selected day meal is within cost range.
2. Selected meal is within nutritial value range.
3. Meal is within preference limits of the patient

A.5.3.2 Method Identification

The task class has now been recognized; The task to be investigated is 'day meal plan' because this task is the smallest entity that can be executed independently of all other tasks.

Now we will check the assumptions of the P&R approach;

The assumptions as described in [Duursma, 1992c] are

- Design can be specified by values for a fixed number of attributes.
- 'Procedures can be specified to determine the starting attributes and their starting values, most likely values and most preferred values' (original).
- There is an enumerable set of design constraints.
- 'For each design constraint remedies in terms of modifications to the attribute values of the design can be specified' (original).
- 'There is not a high level of potential conflict for alternative design extensions' (original).

We can now try to satisfy these assumptions;

- The 'design' is the day meal in the meal plannning task. It is specified by the dishes and drinks to be served in each of the three meals of the day.
- The initial or most likely values of attributes can be taken form the expert (the cook) because he is currently performing the task. It is not very clear how 'likely' is still acceptable for the method.
- The constraints of the meal are enumerable, but they are all relative (this makes it a Class III b design task).
- The level of potential conflict is (again) relative. It cannot be seen always in advance whether the ingedients will be so expensive that no solution is possible in the future. Currently the task is at least solvable because also the expert is able to solve it.

We can conclude that the P&R method seems applicable. It is not clear whether the two relative notions 'level of conflict' and 'likely initial values' can be satisfied. These method assumptions should be described less ambiguously.

A.6 Desired situation

Here a new task decomposition need to be made and an assignment of tasks to the new agents.

A.6.1 Task Design

Here we cannot give a complete new task decomposition because that involves more work than the scope of this example allows. We give a few examples.

Task *Select Breakfast drink'*

agent System or Expert cook.

responsibility agent.

Dependencies

1. Possible drinks: retrieved from food database.
2. Diet: Retrieved from patient database.
3. Insurance type: Retrieved from patient database.
4. Nutritional value of meal: Computed by counting the nutritional value of the other dishes of the day (dependency).
5. Nutritional value of drink: Retrieved from
6. Preference of patient: (primitive).

type of selection criteria: Complex dependencies.