Generic Library Facility Guidelines

Walter Van de Velde *

* *Vrije Universiteit Brussel*

Document Id. KADS-II/T1.3/VUB/TR/053/1.0
CEC Reference Draft Deliverable D8.2b
Date Thu Oct 21
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
Paris Research Centre
86/90, rue Thiers
92513 - Boulogne
France
Contactperson: M. Georges
Phone +33.1.49.10.53.98 Fax: 33.1.49.10.06.15
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 BT SE 1
Otto Hahn Ring 6
D-81739 München
Germany
Phone: +49 89 636 40798 ; Fax: 49 89 636 42284
Contactperson: C. Bauer
E-mail: cb@zfe.siemens.de

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

KADS^II

ESPRIT Project P5248 KADS-II

An Advanced and Comprehensive Methodology
for Integrated KBS Development

Generic Library Facility Guidelines

Walter Van de Velde *

* *Vrije Universiteit Brussel*

Synopsis

This document contains guidelines for the use of the Generic Library Facility as it is specified in the Generic Library Facility: Specification. The user guide is devided in a general part and parts specific for the different types of users.

Document Id.	KADS-II/T1.3/VUB/TR/053/1.0
CEC Reference	Draft Deliverable D8.2b
Date	Thu Oct 21
Distribution	Consortium

Contents

0.1 Introduction

The purpose of this document is to provide a description of the user-related aspects of the generic library facility, its purpose and main functionalities, and to provide guidance on how to use these functionalities.

0.2 Motivation

0.2.1 Generic Facilities

Genericity: the old motivation text on generic tools

0.2.2 Model Development

Libraries of different models

0.2.3 A Knowledge Engineering Effort

Custom libraries, exchange, convergence.

0.2.4 Effective Modeling Support

Different types of support : selection - episodes - modeling approaches.

0.3 Overview

Chapter 1

Introduction

In this report we present a structure for the CommonKADS library together with examples of models that can be stored in it. The primary role of the CommonKADS library is to support a knowledge engineer in the construction of an expertise model for a given application. Such an expertise model is but one of a series of models that are being developed in a CommonKADS project[?]. For knowledge based systems development it is, however, a central one. Providing effective support in constructing an expertise model is therefore a major aim of the CommonKADS methodology.[1]

A secondary role of the CommonKADS library is to integrate the experience on modeling from the wider KBS community. Indeed, the idea of a library of modeling components is not unique to CommonKADS. Several researchers base their work on the hypothesis that similar applications share similar models [?; ?; ?; ?]. Achieving a degree of re-usability of models is therefore a common goal of research in knowledge engineering. Different views are around on how this can be realized, i.e., on what should be the nature of the re-usable components and how the modeling process itself can be best characterized. As far as we know ours is the first comprehensive attempt to integrate these different views into a single library.

In light of this integration effort the structure, contents and usage of the CommonKADS library must take into account a variety of insights from the knowledge engineering community. In particular the experiences from previous KADS users (see also [?]) should be considered. For example the notion of inference structure is generally regarded as an important means to a first understanding of the expertise that has to be conceptualized. At the same time it is acknowledged that inference structures hardly ever completely fit, but instead need modification and further refinement. It is interesting to see how modeling approaches other than KADS have lead to similar insights. For example generic tasks [?] hardly ever completely fit a problem either and they have to be refined and changed in all but trivial cases. Another lesson learned from using KADS is that it is useful to create, during the modeling process, alternative views on the model under construction. In particular, an inference (or data-flow) oriented and a control oriented perspective turn

[1] A similar library could exist for any of the other models in the CommonKADS model suite. Many of the considerations in this report carry over to those libraries. Since the expertise model is so central and distinguishes KBS projects from others, it is natural for the project to focus on these.

out to be important. With regard to this, the Components of Expertise framework [?] also advocates the maintenance of different views: that of domain models, of task models and of problem solving methods.

So, experiences from practice in modeling are often similar, even though they come from different modeling approaches. It is the unified CommonKADS framework for describing expertise models [?] that provides the foundation for accommodating a variety of models into a single library. The structure and contents of the library, however, takes these ideas one step further such that alternative approaches to the process of modeling are facilitated. The present document shows how we are approaching this integration task.

The current library is a partial one and it will need to be expanded and maintained throughout the project and afterwards. This is no surprise, given the fact that the CommonKADS methodology presupposes no focus on a particular type of target application. The number of domains and tasks to be covered is therefore enormous. This task is clearly too big to be finished within the scope of the KADS-II project. The main contribution of this report for a knowledge engineer is, therefore, not so much in the detailed library contents that it describes, but rather in making clear what - and what not - to expect from it and how to use it. Further work on the library will focus on filling in the library contents. This and - most importantly - the practical usage of the library will provide the ultimate validation of the proposals that we here put forward.

1.1 Overview

The contents of this report is as follows:

- Chapter 2 discusses a number of basic issues relevant for the design of the CommonKADS library. It then presents the overall design of the library with respect to library contents, library language and library structure.

- Chapter 3 discusses the Conceptual Modeling Language (CML) as proposed in Deliverable 1.1 [?]. The first part presents a summary of the CML entities as it currently stands. The second part discusses problems and choices that are relevant for deciding upon how the different elements from the CML must be represented in the library in order to be of use for supporting the model construction process.

- Chapter 4 presents the set of basic modeling components that must be defined in the library as well as how they are structurally related. Results in this chapter are based on a re-interpretation of the CommonKADS modeling framework, taking into account the pragmatic requirements posed by the library.

- Chapter 5 discusses generic models. Generic models serve as frames or templates in the construction of complete expertise models. They are complementary to the basic modeling components that can be thought of as providing the contents of the models. Generic models are essential for facilitating alternative approaches to supporting the model construction process.

- Chapter 6 describes how the contents of the library can be accessed by a knowledge engineer. In particular, this chapter discusses different routes of navigation through the library and explains how alternative approaches to supporting the knowledge engineer can be realized.
- Chapter **??** concludes the report by summarizing the major achievements of the work presented in this report. In particular, this chapter points out some differences between the current approach and the previous KADS-I framework.

Chapter 2

Overall Design of the Library

This chapter is composed of three parts. In the first part, we discuss the basic issues in the design of the library of expertise models. In the second part, we present the overall design of the CommonKADS library and show how it addresses these issues. The third part expands on preliminary ideas toward support tools for maintaining and using the library.

2.1 Issues in the design of the library

The first question pertaining to the design of the library is:

> *What is being stored in the library?*

There are at least two extreme answers to this. First one could store only complete and ready-made expertise models. Such models would be highly specific for an application class and in principle easy to use, but reasonable coverage would require an enormous amount of such models to be stored. The other extreme is to store only the smallest building blocks of expertise models (knowledge roles, domain entities, task descriptions, control statements, and so forth), as if there were a "shelf" in the library for any type of model element that has been defined in [**?**]. This approach is unlikely to provide much support for a knowledge engineer who would probably get lost in the large number of primitives that can be selected and the huge number of possible combinations that can be made.

It is clear that in fixing the nature of library elements several points have to be taken into account. Care has to be taken that the grainsize is not too small so that components can be meaningfully indexed. On the other hand a larger grainsize typically diminishes the chance that the component be re-usable. Another factor is the degree of guidance in modeling. Smaller components require a lot of modeling still to be done but in themselves don't say anything about how to plan this. This is particularly problematic for expertise modeling since the relative interaction hypothesis [**?**] predicts that there can be any degree of dependency between different parts of a model. So, whatever grainsize one fixes on the library, an additional element of guidance in model construction seems to be needed. In other words, genericity arrises not only from the re-usability of components, but also - if not mainly - from the re-usability of combinations of them.

A second question to be answered is:

In what language are library elements described?

The library must cover a wide range of potential users and of syntactic, semantic, formal and predictive restrictions. To meet these requirements it must allow for various levels of precision and different views according to the user's perspective. There are at least three possible ways to describe expertise models: a *verbal*, a *conceptual* and a *formal* description (from the less structured and precise to the more structured one). It seems useful to store any of those in the library as alternative descriptions to be used depending on the state of modeling, the level of precision needed, or the kind of user. However, it would be problematic to allow descriptions in any of these levels indiscriminately – e .g., to allow an element to be described in a verbal way without a conceptual or formal description. In this way, one runs the risk of having items with essentially the same knowledge content stored as different or unconnected elements, because there is no obligatory link between them. Therefore, it is necessary to choose one of these levels as central, in such a way that the description at the other levels are connected to the description at the central level.

Another question to be considered in this respect is whether the CML as proposed in [**?**] is appropriate as is for the library. The library language must of course be consistent with the CML from DI.1 but it must also fit with the practice of modeling. For example, the incremental construction of large models requires facilities for typing, naming and easy referencing. But there are additional elements that will require an extension of the CML and that will be discussed in more detail in Chapter 3.

Another issue to be considered in the library design is:

How is the library used?

The primary role of the library is to assist in building an expertise model for a concrete application. For this it is not sufficient for the library to simply store elements, but it must also support finding the most useful ones. Usefulness is determined in part by the various epistemological and pragmatic constraints imposed by the application [**?**]. These are the so-called task features[1]. For example, a diagnostic application for systems the behavior of which is only qualitatively known and can be only partially observed will require a considerably different model than when "perfect" knowledge is available. Usefulness is also determined by the restrictions that follow from previous modeling choices. For example, when a problem solving method has been selected a certain type of domain model may be necessary. In summary then the library must be indexed by task features so that informed selection of model elements is facilitated and useful combinations indicated, and so that constraints that follow from modeling decisions are adequately reflected in the alternatives that can be retrieved from the library.

[1] Task features are primarily studied in Task I.5.

An additional point will turn out to be important. The practice of modeling show that it is not always possible in advance to decide upon the role of a library element in a finished model. A typical example is to choose between the status of "task" or "inference". While this may be clear in the final model, it is not initially. It follows that the library can not follow too strictly the category and type structure of a model of expertise as defined in [?]. Certain knowledge elements must have a common "shelf" in the library (e.g. a "task-or-inference") that can reflect the undetermined role status of certain library elements.

A final issue in the design of the library is the following:

> *How are different modeling approaches integrated?*

As described in the introduction, a second role of the library is the integration of different modeling approaches [?; pp.34-36]. For example the inference structure driven approach of KADS-I [?] is different from the method driven approach advocated by McDermott [?]. A modeling approach implies a choice of a type of entry point in the library (e .g., a generic task, an inference structure or a problem solving method) and a paradigmatic approach to modeling (e.g., from inference structure down to domain modeling and up to the task modeling). In order to support these different approaches the library must allow for different access routes into the library. When we view the construction of a complete expertise model as a search process through a space of (possible) expertise models[2], then a second role of the library is to point at interesting steps to be taken in the modeling process. For example, from a given state of the expertise model several refinements may be possible, or the combination of two models into a single new one may be an interesting step to take and the relation between the generic models involved should be documented in the library.

Apart from these four "academic" questions, pragmatic considerations must play an important role in the design of the library. In particular, practical experiences and preferences of KADS users must be taken into account (see also [?]). The following observations can be pointed out:

- Inference structures are generally regarded as an important means to a first understanding of the expertise that has to be conceptualized, independent from the possible control structures that can be defined over it.
- Inference structures hardly ever completely fit, but instead need modification and further refinement. This is facilitated by the meshing of task and inferences (and therefore also of task decomposition and inference structure) in the library.
- It is useful to have, during the knowledge acquisition process, the possibility of creating alternative views on the contents of the expertise model being constructed. The library explicitly supports a data-dependency oriented and a control oriented perspective.

[2]This is the view being taken in Task I.5 that studies task features

The experience of KADS users is relevant because it provides experimental data with respect to the practical usefulness of specific aspects of the CommonKADS methodology that the library should support.

2.2 Basic Design Decisions

In this section we present our basic decisions related to the design of the library. In particular, we discuss proposals for library elements, for the language of the library and for the structure of the library.

2.2.1 The elements of the library

We distinguish between two (disjoint) categories of library elements: the *basic modeling components* and the *generic models* (Figure 2.1).

- The *basic modeling components* are the smallest ones in the library. They are canonical (or *orthogonal*) in the sense that they are one partition of the expertise model (i.e., their contents do not overlap) in elements small enough to provide the desired flexibility by their recombination. This implies a *canonical structure* for every complete model through the necessary relationships between the different types of basic modeling components. For example, an inference function is decomposed into an inference structure, and a control structure is imposed on an inference structure.
- The *generic models* are frames that represent a class of complete expertise models. They capture useful combinations of basic modeling components and serve as templates for the construction of complete expertise models. A generic model is an expertise model frame, complete in form but not necessarily in contents. This notion of generic model provides for any kind of model to support a modeling approach. It can express, for example, Chandrasekaran's generic task as a useful combination of a certain type of task model, domain models, inference structure and control structure.

The basic modeling components are inference functions, inference structures, control structures, domain models, case models, domain ontologies and problem solving methods. They will be described in detail in Chapter 4. Basic modeling components provide a useful level of granularity and sufficient flexibility in modeling. In themselves, however, they do not provide any guidance on what are useful combinations. This guidance is contained in the generic models. The notion of generic model is a straight generalization of Chandrasekaran's generic tasks [**?**]. Just like a generic task a generic model supports modeling because it documents useful ways for bringing together basic modeling components into complete expertise models. The basic structure of a generic model is defined by the canonical structure among basic modeling components. This canonical structure is in a sense the most "empty" generic model.

An interesting aspect to comment in this respect is the question of *redundancy*. While redundancy is extremely undesirable in the category of basic modeling components, this

Figure 2.1: The library contains two categories of elements: basic modeling components and generic models. The basic modeling components are building blocks of expertise models. The generic models are templates for complete expertise models. These two categories of library elements play complementary roles in modeling. (The layout of the library in the figure is not significant.)

is not true for the generic models. To allow flexibility and the use of different modeling methods and strategies, the various generic models may overlap. For instance, an inference structure may appear both as a separate component and as part of one or several generic components like a Chandrasekaran's generic task.

2.2.2 The language of the library

Every library entity (basic modeling component or generic model) can be described as follows (see also Figure 2.2):

1. **Verbal description:** a (partial) knowledge level model of an application in natural language. This is a description in terms of "texts", "quotes" (sequences of words from a text) and "words", and references among these, described by texts themselves. This is like a hypertext language, and some of Shelley's analysis features may be still useful here.

2. **Conceptual description:** a knowledge level model of an application in the "core" CML language for describing knowledge items and structures. The language used is based on an interpretation and extension of the CML [**?**] and is being described in Chapter 3.

3. **Formal description:** a knowledge level model of an aspect of the task reality in the formal language ML^2. We refer to [**?**] for an introduction to $(ML)^2$ and to [**?**] for a more detailed discussion of some of the technical issues.

Figure 2.2: Library elements may be described at a verbal, conceptual and formal level. The conceptual level is the central one. Task features serve as starting points of paths leading into the library.

The most adequate level to tie the three levels in a coherent way is the conceptual level. To use the verbal level would still allow confusion because it has no structure. To use the formal one would require a level of understanding of formal issues that is unrealistic to assume for the average knowledge engineer. The conceptual level is both structured enough to avoid lack of integration and informal enough to be understandable to the average knowledge engineer. Thus, library elements are primarily conceptual components with formal and/or verbal descriptions attached.[3]

[3] The conceptual descriptions are the main focus of attention in this document.

The conceptual language must be consistent with the CML from DI.1 [?] but it must also fit with the practice of modeling using the basic modeling components and generic models (Section 2.2.1). For example, parts of expertise models may be re-usable (and therefore worth while to store in the library) but it may be hard to pre-specify their role in the final model. This is for example the case with tasks and inferences which are sometimes hard to pin down in the initial phases of modeling. In Chapter 3, we will discuss this issue in more detail and propose a useful interpretation and extension of the CML to accomplish these goals.

2.2.3 The structure of the library

The library supports two activities: selection of useful model components and generic models, and progression through useful steps in modeling. This is achieved by a number of structuring principles within the library (see also Figure 2.3):

- **Structure among basic modeling components:**
  the sets of basic modeling components are structured in basically two ways:
  - **Typologies of basic modeling components:**
    basic modeling components within a single class are organized in a series of typologies. Typologies facilitate the construction of an expertise model from a generic model by indicating the refinements and alternatives that the library provides. Typologies can be based on various principles, for example on ontological choices (see Appendix ??)
  - **Constraints from component features:**
    component features document compatibility constraints between basic modeling component. A component feature can indicate component necessity or component exclusion. For example, the choice of an inference may require or exclude the usage of a certain type of domain model. Component features restrict the set of potential combinations of components to those which are structurally possible.
- **Structure between generic models and basic modeling components:**
  a compositional link relates generic models and basic modeling components. The generic model is a frame which restricts combinations of basic modeling components, typically by type restrictions.[4] The compositional link documents the structure of a generic model in terms of its constituent component types.
- **Structure among generic models from modeling operators:**
  the process of modeling that the library should support can be viewed as a search among expertise models in which various modeling operators are applied to the model under construction.[5] Modeling operators are typically parameterized by library elements and exploit the structures among basic modeling components. For

[4] One can envisage different constraints to be represented by the generic model, but before we have seen examples of this we will apply occam's razor to keep things simple.

[5] This view is extensively explored in Task I.5.

example, a specialization operator may exploit a typology of domain models. When one generic model can be obtained from another one in this way, then this is also documented as a relation between them in the library of generic models. To navigate through the library of generic models is to explore potential modeling episodes.

- **Task features as primary guidance in modeling:**
a task feature is any characteristic of an application and its environment that informs the construction of an epistemologically and pragmatically adequate expertise model for it. For example, task features can help in the selection of model components or generic model. More generally task features indicate modeling operations (and model selection is one of them) parametrized by library elements. Thus, the task features are the primary indices for accessing, navigating though and exploiting the library in modeling.

Figure 2.3: 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 (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.)

Structuring the library of generic models according to modeling operators is a powerful means of decoupling the library structure from the structure of the models being stored. In the library of basic modeling components, modeling proceeds based on fine grained operations that are guided by the structural relationships between the different components. In the library of generic models on the other hand, models that are very different, and could be obtained only by a number of construction steps over components can be directly related to one another. For example, the application of a method to a task typically requires to adjust terminology (see Appendix **??**). The relation between a generic model that represents a task - method combination is therefore more than simply compositional. Moreover, modeling operations allow for expressing genuine combinations of complete models, like the integration of a classificatory problem solver and a constructive one into a new model [**?**]. Such operations are not easily expressed using basic components.

2.3 Support tools

Several of the tools which have been defined within the project will make use of the library in one way or the other. A particular tool - the *Library Management Tool* will provide the environment for defining new library components and modifying existing ones. This is the tool of the 'librarian', i.e. the system administrator. Tools for the knowledge engineer which have strong relations to the library include the Conceptual Model Editor (CMED), and the task feature tool - referred to as the Model Selection Tool. The latter helps identifying appropriate library components on the basis of features of the task reality, while the CMED is the major tool for developing the expertise model.

Hence, the CommonKADS Library is not the only KADS-II tool used to build a model, and it cannot build it alone. The elements of the library are to be retrieved to be later used, completed and modified with an editor. Since the conceptual level is the central one, the editor which will be most important for the knowledge engineer developing an expertise model, is the Conceptual Model Editor - the CMED. The library need to play tightly together with this editor, since the role of the library is to supports the modeling process by providing knowledge items - or partial expertise model elements - organized and structured according to its intended use. The library plays an important role within the CommonKads modeling methodology, and it is important that the support tools is designed to facilitate and encourage an effective use of the library.

The mutual dependency and intended between the library and the support tools can be illustrated by making an analogy with an ordinary library. [6] If one needs to make a report, it is necessary to consult various literature sources: books, conference proceedings, journals, etc. To structure and fill one's document, it is possible to use pieces of text from articles or books - usually somewhat modified - and edit them into the document being prepared. The CommonKADS Library can be seen as the ordinary library in this analogy, and the Conceptual Model Editor can be compared to whatever tool used to prepare the document. The library is the support from which we may retrieve pre-prepared components and modify them in order to integrate them into the current version of our product (expertise model or document, respectively). The library and the tools to effectively utilize it help us to avoid "reinventing the wheel", and thus increase both the quality of the end product and the efficiency of the production process.

2.4 Summary

In this chapter we have described and motivated the basic design decisions for the CommonKADS model library. They can be summarized as follows:

- The library consists of two disjoint categories of elements: basic modeling components and generic models. The latter can be thought of as providing "form" (frames to be filled), the former as providing elements of "contents".
- Every library entity can be described in a verbal, conceptual and formal way. The conceptual descriptions are central, while the other descriptions - when available - can be accessed from it.
- The structure of the library of basic modeling components is derived from the necessary relationships between these components (type and structural). The library of generic models is structured according to different types of modeling operations over expertise models. These two means of structuring provide complementary guidance.

The remainder of this document focuses on specifying in more detail the basic modeling components, the generic models and the structure that relates these entities. The appendices provide examples of library contents.

[6] The argument that follows applies also for formal level descriptions, with the Formal Model Editor.

Chapter 3

Practical Interpretation of the Common KADS Modeling Language

This chapter discusses the conceptual modeling language that is used in the library in two sections. Section 3.1 presents a summary of the CML as proposed in Deliverable I.1 [**?**]. In order to realize the scheme proposed in Chapter 2 we must repartition the CML entities into a small set of basic modeling components. Section 3.2 shows how this is being done.

3.1 The CommonKADS Expertise Model

A CommonKADS expertise model consists of knowledge in three *knowledge categories*: *domain knowledge*, *inference knowledge* and *task knowledge*. Structure between these categories is realized by the so called *knowledge roles*. Knowledge about constructing a model within this framework is called *problem solving knowledge*. It consists of *problem solving methods* and *strategic knowledge*. The structure that prescribes the allowed form of any expertise model is also shown in Figure 3.1. An expertise model is described as a large componential structure (part-of) of knowledge items from predefined knowledge categories. For each category DI.1. specifies (potentially) a type-hierarchy for the legal representatives of this category, and prescribes the attributes that describe them. In the following subsections we briefly describe each of these constituents.

Figure 3.1: An expertise model is a large componential structure (part-of) of knowledge items from predefined knowledge categories. For each category DI.1. specifies (potentially) a type-hierarchy for the legal representatives of this category. The process of modelling can be viewed as filling this expertise model frame by pushing down the category representatives along the branches of their typologies until they are sufficiently specific.

3.1.1 Domain Knowledge

Modeling the domain knowledge is concerned with representing the knowledge that is specific for the application that the problem solver reasons about. According to [?] domain knowledge can be divided into *meta* domain knowledge and *application* domain knowledge. Application domain knowledge is knowledge about the specific system (object system) that is subject of the problem solving process. Meta domain knowledge describes the vocabulary and the structure of the application domain knowledge. [1]

The entities for modeling the application specific knowledge are:

- domain ontology
- domain model
- case model

The domain ontology specifies the vocabulary that can be used for the construction of domain and case models. The domain model represents general knowledge of the domain, i.e. the knowledge that is independent from the specific system that is reasoned about. Some typical examples are models in terms of causal dependencies between symptoms and diseases in medical domains (causal model) or models that specify structural hierarchies for physical devices (structural models).

The case model represents the information about the specific system that has to be reasoned about. It presents the case specific data for the problem solving process. Four aspects can in this respect be distinguished: (1) initial data (provided before problem solving starts) (2) user input data (provided during the problem solving process) (3) intermediate information (data derived during the problem solving process) and (4) solution (required output of the problem solving process)

3.1.2 Inference Knowledge

By using domain knowledge (case and domain models) an inference is concerned with establishing a relation or a function between the various entities of the domain knowledge. The following aspects are important for specifying an inference (cf. [?]):

- name
- dynamic knowledge roles
  - input role(s)
  - output role
- description (goal)

[1] Referring to meta knowledge as being domain knowledge is a bit unusual because this type of knowledge is independent from the application that is modeled and therefore *not* domain specific knowledge. The meta domain knowledge in fact specifies the knowledge representation that can be used for modeling the domain specific knowledge. We will not discuss this type of knowledge any further in this section.

- static knowledge roles
- realization (or specification)

The name of the inference is merely a label which can be used as a unique identifier for the inference. The dynamic knowledge roles refer to the input and output of the inference, i.e. the domain entities between which the inference establishes a relation or a function (an inference usually has one output role, but may have multiple input roles). The description slot describes the relation between the input and output roles that is realized by the inference. The static knowledge roles refer to the domain knowledge that is used by the inference in order to establish that relation. Finally, the realization refers to a formal description of the inference, i.e. how the required relation between the dynamics knowledge roles is realized using the dynamic knowledge role. It is important to realize that an inference has no control. Both the activation itself as well as how often it must be activated is control knowledge that is imposed upon the inferences by certain tasks (see also task knowledge). In addition, control can not be exercised on the execution *within* the inference itself.

The following points specify additional constraints on defining an inference:

- An inference can be a 'relation' or a 'function' depending on the direction in which the dependencies between the dynamic knowledge roles may be established. In the case of a relation the inference can be used in both ways (non-directional), whereas in the case of a function the inference may only be used to derive the output form the input (directional).
- Whether or not an inference can be realized is determined by the ability to fill in the *realization* slot, in particular whether the required static knowledge roles are available. When this is not possible this means one is actually modeling a task that must be refined into smaller steps that can be realized by the available domain knowledge (see also task decomposition).
- An inference may introduce assumptions, i.e., characteristics of the domain knowledge that are assumed to be true (but not necessarily are). Assumptions are stated in terms of the inference specific terminology and they must be justified by the presence of certain task features of the application that is being modeled. For example, a single fault assumption can only be made if an application is not too critical (a task feature) and may be justified by the feature that time constraints apply.

The notion of an *inference structure* refers to a number of individual inferences being presented together. It presents an overview of the data-dependencies between the different inferences (in terms of knowledge roles) and the inferences that must be carried out. Crucial for an inference structure is that at least one of the knowledge roles (static or dynamic) of each inference is also a knowledge role of another inference.

3.1.3 Task Knowledge

According to [**?**] task knowledge describes how a given goal can be achieved. It is essentially concerned with how inferences can be applied in order to realize problem solving goals. A task can in this respect be thought of as a 'shell' that encapsulates one or more inferences. It specifies:

1. which inferences must be carried out,
2. in what order inferences must be carried out, and
3. how often one inference or a sequence of inferences must be carried out.

Each task is characterized by a *task definition* and a *task body*. A task definition consists of:

- name
- case roles[2]
  - input role(s)
  - output role
- goal (description)

The name of a task is a label by which the task can be identified uniquely. The case roles refer to the input and outputs of the task. The goal (or description) refers to an intentional specification of the relation between the case roles that is realized by the task. The goal description therefore uses the terminology of the case roles.

A task body consists of:

- realization (by means of either:)
  1. an inference
  2. decomposition into subtasks (and/or subgoals ??)
  3. a transfer task
- control (either on:)
  1. on the inference (how often it has to be carried out)
  2. on the order of the subtasks
  3. when to present the task to the user

[2]The difference between case roles and dynamic roles is subtle: dynamic roles are changed by the inference knowledge, whereas the case roles allow the task knowledge to "perceive" these changes in the case model.

There are three ways in which a task can be realized. It may be the case that there is an inference that can make the required deductions in order to realize the goal of the task. If such an inference is not available then the task has to be decomposed into subtasks. Each of these subtasks has then to decide for itself if it can find an inference that can realize its goal. Finally, the task can 'become' a transfer task, i.e. the inputs of the task are presented to the user and the user is supposed to perform the task and present the required output to the problem solver again.

Most typical for a task is the control it imposes in order to realize its problem solving goal. Depending on the realization of the task three different aspects of control are important. When the task is realized by a single inference the control consists of determining how often this inference has to be executed. If the task is realized by decomposition into other task, the control specifies the order in which the subtasks have to be carried out. If the task becomes a transfer task then the control determines when the task has to be present to the user.

3.1.4 Mapping between Knowledge Categories

This section describes how the three knowledge categories as discussed above can be mapped onto each other.

- **Mapping between Inference and Domain Knowledge**
  The mapping between the inferences and the domain knowledge is realized by the dynamic and static knowledge roles. These knowledge roles define how a certain piece of domain knowledge is used by the inference. Firstly, these knowledge roles divide the domain knowledge into static and dynamic use. Secondly, the specific names each of these knowledge roles has (for example: symptom, disease and solution) further define the specific use of each piece of domain knowledge.

  Which pieces of domain knowledge map onto which static or dynamic knowledge roles is specified by the *model terms*. These model terms link a certain entity in the domain knowledge to a certain role of an inference. A typical connection between domain knowledge and inference roles is to map:

  - domain models onto static roles, and
  - case models onto dynamic roles.

  It seems also worthwhile to allow for domain models to perform dynamic roles, but it is impossible for an entity from a case model to perform a static role.

- **Mapping between Task and Domain Knowledge**
  The mapping between the tasks and the domain knowledge is realized by the case roles. These roles define which entities of domain knowledge are input and output for the task. Which pieces of domain knowledge map onto which case roles is specified by the *model terms*. These model terms link a certain entity in the domain knowledge to a certain case role of the task.[3]

[3] Only entities from the case model be inputs and outputs of the task. We can think of examples in which at least part of a domain model is required to specify the task definition. This would require a static

- **Mapping between Inferences and Tasks**
  There is a relation between inferences and tasks. In the definitions given above a part of the task definition is similar to a part of the inference definition. In particular, the case roles (or dynamic roles) which specify the input and output roles are similar for both. However, the *contents* of the case roles (of a task) and the dynamic roles (of an inference) are only the same when the task is realized directly by the inference. Inferences and task differ with respect to the notions of static role, control and realization.

3.1.5 Problem Solving Knowledge

According to [**?**] an expertise model consists of two major parts: a model of the application knowledge and a model of the problem solving knowledge. Domain, inference and task knowledge can be used to represent the application knowledge. Problem solving knowledge represents knowledge about the problem solving process. It is defined as a description of how to achieve a certain task, characterized by a goal and input/output specifications, through the generation of subtasks (goals) and/or the application of primitive inference actions. More specific: a problem solving method describes a way of decomposing a task in simpler subtasks and specifies control on these subtasks. A problem solving method consists of the following aspects:

- name
- goal
- (task) characterization
- control roles
- subtasks
- control
- method definition
- task expression schema

In the case of a problem solving method that applies an 'primitive inference action' the control roles, subtasks and control are replaced by the 'inference relation used'. Unfortunately the 1.1 deliverable does not further specify the meaning of the different aspects that constitutes a problem solving method.

knowledge role for tasks as well, a role which is currently subsumed by the case model. Of course, it is impossible that the output of a task feed into domain models (by definition!).

3.2 The CML versus Basic Modeling Components

In this section we present our repartition of CML entities into a basic orthogonal set of modeling components. Basically, we argue that similar functional units which are part of different entities from the CML should be singled out and represented in the library as independent units. These functional units then form the basis for the set of basic modeling components that are discribed in more detail in Chapter 4. This re-partition is motivated by the design issues and decisions for the library that have been discussed in Chapter 2:

- First of all we have been arguing that basic modeling components must not be too small so that they can be meaningfully indexed, and orthogonal so that redundancy is avoided. In terms of the componential structure this means that we do not consider as a legal library entity anything which is below a pre-defined (canonical) boundary.
- Second, the practice of modeling show that it is not always possible in advance to decide upon the role of an entity. A typical example is to choose between the status of "task" or "inference". While this may be clear in the final model, it is not initially. In terms of the type hierarchies of expertise model components this means that certain hierarchies have a common root (e.g., data flow element) that can serve as the temporary status of a knowledge item. As modeling proceeds the entity is specialized along the relevant branch until it finds its final place in the expertise model.

Figure 3.2 illustrates what is happening in terms of the original CML structure (compare with Figure 3.1).

Figure 3.2: Basic modeling components define a fixed level of granularity for knowledge components to be meaningful library entities. Moreover, type hierarchies of different knowledge categories are merged by extending them upwards in order to account for the undetermined type of knowledge items in the library.

3.2.1 Separating Data Flow, Decomposition and Control

We made the following choices for establishing the classes of basic modeling components to be stored in the library:

- *Separate out in the library the data-flow aspects from tasks and inferences.*
  Although terminology is somewhat different, the definition of an *inference* is similar to the definition of a *task definition* except for the notion of static roles and specification (realization). We therefore propose to create an independent modeling component that represents the common data-flow aspects of both inferences and tasks. Such a component represents the name of the task or inference, the input and output roles, and the goal or description.[4]

[4] It seems a good idea to unify terminology for task and inference in DI.1 as well.

- *Separately represent decompositions from data flow elements*
  Multiple realizations of a data flow element (inference or task) can be thought of. It is therefore useful to separate in the library the the representation of a data flow element from the representations of the possible realizations (decompositions) of such an element.

- *Separately represent control from (set of) data flow element(s).*
  Both a set of data flow elements as well as single date flow element can be controlled in multiple ways. It is therefore useful to represent in the library the possible control structures independent from the data flows to which they apply.

These choices lead to the following set of basic modeling components that deal with several (orthogonal) aspects of an expertise model:[5]

- *Inference functions* for data flow,[6]
- *Inference structures* for decomposition,
- *Control structures* for control,
- *Domain models, case models and domain ontologies* for domain knowledge,
- *Problem solving methods* for connection (or linkage) (see the following subsection).

3.2.2 Integration of Problem Solving Methods

In [?] problem solving methods are not (yet) a fully integrated part of the CML. Their definition to a large extend overlaps with the definition that is given for representing task knowledge. More specifically, it is not clear how problem solving methods relate to the notions of inference structure and control structure.

For the library we will consider problem solving methods to be the dynamic counterparts of inference and task knowledge. Ideally, a problem solving method would specify:

- how a certain data flow element can be decomposed (or expanded) into a subset of data flow elements at a lower level of detail,
- which control structure is required on the data flow elements, and
- which requirements are imposed on the representation of the domain knowledge.

[5] One could consider that there is also redundancy in the domain modeling, between case and domain models. In order to avoid double modeling there one might introduce a unifying type for both domain and case model. In general, the library issues related to domain modeling are somewhat less worked out.

[6] This notion of inference function is, unfortunatly, not the same as the one used in DI.1 [?]. For lack of a better name we use it consistently in this document. A similar remark applies to inference structures, which in the library play the role of both inference structure in the DI.1 sense and task decomposition.

Not all three aspects must always be present in the definition of a method. Problem solving methods can be domain independent and thus not impose any requirements on the representation of the domain knowledge. Also we can think of problem solving methods that only provide us with an expansion or for that matter with a control structure.[7] Finally, problem solving methods may introduce recursive levels of decomposition, i.e. a configuration of data flow elements and control on these at more than one level of detail.

For the library the notion of problem solving methods may seem to be problematic, because they introduce a certain amount of redundancy with respect to sets of data flow elements and the control structures. The fundamental question is the following:

> *Can we think of problem solving methods that really (dynamically) generate sets of data flow elements and control structures* **or** *can we only describe problem solving methods in terms of their output (and consequently never capture the method itself)?*

For the time being we will adopt the latter[8]: a problem solving method can always be described in terms of data flow elements and/or control structures and/or domain knowledge requirements. We will use the notion of problem solving methods as being the entities that provide the *links* between the modeling components in the library. This choice follows the no-redundancy requirement as well as the integration requirement.

3.3 Summary

In this chapter we proposed a re-interpretation of the CommonKADS modeling framework and language for the purpose of its usage in the model library. This re-interpretation was motivated by the need to partition expertise models in a useful set of basic modeling components. The set of basic modeling components that we propose to store in the library is derived from three (representation) choices:

- Seperatly represent in the library the definition (data-flow) and realization or body parts of inferences and tasks;
- Separately represent in the library decompositions from data flow elements;
- Separately represent in the library control from (set of) data flow element(s).

Based on this, we have put forward a set of modeling components to deal with data flow, decomposition, control and domain knowledge. In addition, problem solving methods are being interpreted as the means to provide linkage between these different components. In this way we have achieved a small set of basic modeling components that satisfy our needs. In the next chapter (Chapter 4) we will further elaborate on the detailed definitions for each

[7] The so called weak methods (cf. [?]) are typical examples of problem solving methods that specify control structures.

[8] The exact interpretation of problem solving methods is a difficult issue to be solved by further Task I.1 work. Our minimal solution to the problem should not be considered as the final word on problem solving methods in CommonKADS.

of the basic modeling components in the library. The generic models that specify frames for putting together complete models from these components are treated in Chapter 5.

Chapter 4

Basic Modeling Components

In this chapter we describe in more detail the basic modeling components that were put forward in chapter 3: *inference functions* for date flow (Section 4.1), *inference structures* for decomposition (Section 4.2), *control structures* for control (Section 4.3), *domain models, case models and domain ontologies* for domain knowledge (Section 4.5) and *problem solving methods* for connection (Section 4.4) (see also figure 4.1). In the last section (Section 4.6) we discuss how the basic modeling components can be orgnanized in a series of *typologies*.

Figure 4.1: Structure of the library: basic modeling components

4.1 Inference Functions

Inference functions are Basic Modeling Components that represent data flow elements. The following aspects are important for defining an inference function:

- name
- dynamic roles
  - input role(s)
  - output role
- goal/description
- assumptions
- typologyreference

The name refers to a unique identifier of the inference function. The dynamic roles refer to the input and output roles of the inference function. The goal/description characterizes the input/output relation that the inference function realizes between the dynamic roles. Assumptions specify characteristics of the inference function with respect to its applicability. Typically they define requirements on the nature of the domain knowledge that the inference requires to work properly. What kind of static role is required by the inference function depends on the assumptions and the typology reference. [1] The typology reference refers to the typology on the basis of which the functionality of the inference is defined. [2] The ontology reference is optional but obviously inference functions that are not specified in terms of a certain ontology are less suited for supporting the knowledge acquisitions process.

During the modeling process inference functions are transformed by the knowledge engineer into inferences or tasks (in the sense of [?]) depending on whether *static roles* can be defined to realize the mapping between the input and the output roles. If the required domain model is not available the inference function will become a task that has to be decomposed into inference functions that can be realised by the available domain models.

4.2 Inference Structures

Decomposition is realised by an inference structure: a set of inference functions that share input and output roles. The relevant aspects of an inference structure are:

- name
- description
- a set of inference functions
- input/output dependencies between inference functions (each inference function from the set has at least one input/output dependency with another inference function from the set)
- assumptions

The name is again a unique identifier for the inference structure. The description refers to a statement in natural language about the characteristics of the inference structure. Each inference function from the set of inference functions that constitutes the inference structure is an inference function as defined in Section 4.1.

The inferences in an inference structure are grouped together in a connected input-ouput relation. This structure can be used to decompose an inference function at a certain level of detail into a set of inference functions at a lower level of detail. [3] The idea here is that the inference functions at the lower level of detail (possibly after a number of decompositions) will be detailed enough for realisation of the inference function by using the available

[1] On the basis of the discussion in Section 2.2.3 one can argue that the typology reference is actually a component feature, expressing component necessity (in this case necessity of a domain ontology).

[2] Depending on the ontology in which the inference function is defined, these functions can be further distinguished into (1) basic (or problem related) inference functions and (2) canonical inference functions (see also Section 4.6). Inference functions that do not refer to a specific ontology will be referred to as library defined inference functions.

[3] May the decomposed inference function be a part of the inference structure that realises the decomposition, i .e. be fully recursive?

domain models. How inference functions can be decomposed into an inference structure is defined by expansion methods (see Section 4.4).

The construction of inference structures may introduce new assumptions which did not apply to the individual inference functions of the structure. The assumptions slot can be used for this purpose.[4]

4.3 Control structures

Control structures are essentially a piece of control defined by a certain control language that includes things like 'while', 'if-then-else', etc. It imposes control on a single inference function or on an inference structure.[5]

Different control structures can be imposed on a single inference or inference structure, which is one reason why we choose to represent control structures as independent library elements. Although we can think of control statements independent of the specific set of inference functions to which they apply it is usually more appropriate to associate the control to a set of inference functions. In terms of library strcuture this simply means that the different control structures are linked by a component feature of component necessity to the inference or inference structure that they apply to. An additional and more fundamental reason why we represent control strcutures as seperate library entities is to accomodate the notion of a control method (see also Section 4.4). Control methods can be highly specific, i.e., pointing directly from an inference structure to a specific control structure. They can also be more general, however. For example the notion of data-driven control or hypothesis driven control are not specific to an inference structure but indicates a fairly specific type of control structures for a whole class of inference structures. These are therefore not control structures but specify ways - though maybe not algorithmicaly - of arriving at specific control structures.

4.4 Problem Solving Methods

The connections in the library are realized by the problem solving methods. At least three types of methods can be defined:[6]

- Expansion methods
- Control methods
- Role refinement methods

[4]A criteria that could be imposed on the construction of inference structures is to require that only inference functions with the same ontology reference may used in a single inference structure (=canonical inference structures). This requirement would prohibit the construction of inference structures with inference functions at different (arbitrary) levels of detail. It would be premature, however, to enforce that restriction at this point.

[5]Control structures correspond to the notion of *task structures* in KADS-I.

[6]In order not to violate the orthogonality assumption, and based on the discussion in Section 2.2.1 complete methods are moved to the category of generic models.

An expansion method associates an inference function at a certain level of detail to an inference structure at a lower level of detail and as such realises decompositions of inference functions. The relevant aspects of an expansion method are:

- name
- description
- inference function (to which the method applies)
- inference structure (the method delivers)
- mapping between input/output roles (of the inference function to input/output roles of inference functions in the inference structure)
- rational (of the method)

The name is again a unique identifier of the expansion method and the description is a statement using natural language about the method. The inputs and outputs of the expansion methods are described in terms of the inference function to which it applies and the inference structure that it delivers. The method must also specify how the input and output roles of the inference function that is decomposes can be mapped onto the input and output roles of the inference functions in the inference structure.

A method is not an arbitrary connection between an inference function and an inference structure, instead it is based on the features of each of these entities. In other words the expansion method is based on the characteristics of its input and its output. The rational is verbal description (or possibly a formal) of the reason 'behind' the method. It provides an account of why the expansion delivered by the method is an useful and/or interesting one.[7] Multiple expansion methods may of course be applicable to a single inference function.

A control method associates an inference function to a control structure.[8] The important aspects this method are:

- name
- description
- inference structure (to which the method applies)
- control structure (the method delivers)
- rational (of the method)

The definition of a control method is essentially similar to the definition of the expansion method. Although they differ of course with respect to their input and output.

[7] The 'description' and the 'rational' slot differ in the sense that the latter does incorporate an account of why the method is useful.

[8] A control method may also apply to a single inference function.

Finally, there are methods that specify data refinement. These methods are a further specification of the mapping between input/output roles of one inference function to the input/output roles of another inference function. Usually the mapping is taken care of by expansion methods (see also definition of expansion methods).

4.5 Domain Knowledge

Representation of the domain knowledge essentially follows the definitions discussed in the previous chapter (Section 3.1.1):

- Domain models
- Domain ontologies
- Case models

The connection between domain models and domain ontologies on the one hand and inference functions and inference structures on the other hand is realized by assumptions and the notion of typology reference. In particular, the latter specifies which domain models can be used for realising a certain inference function.

4.6 Typologies of Basic Modeling Components

In Chapter 2 we have introduced typologies as the means of structuring a class of basic modeling components. Typologies facilitate the construction of an expertise model from a generic model by indicating the refinements and alternatives that the library provides. Typologies can be based on various principles. For example the descriptions of the different basic modeling components that we have introduced in this chapter provides for a straightforward hierarchy based on a simple (frame-based) notion of specialization and inheritance. But other, more sophisticated typologies can be envisaged, for example based on ontological considerations. Table 4.1 lists two typologies on inference functions:

Extension	Bibliographic References
Canonical Inference Functions	this report, [?]
Basic Inference Functions	this report, [?]

Table 4.1: Two examples of typologies on basic modeling components.

Canonical inference functions (typology): The KADS-1 approach used a set of knowledge sources with a canonical character, that were in fact borrowed and adapted from KL-ONE. We can adapt the same typology to the KADS-II framework, generating the *canonical inference functions*. As the name says, they are a typology of inference functions that are defined by ontological distinctions. We discuss canonical inference functions in more detail in Appendix **??**.

Basic inference functions (typology): The same typology used to define the basic tasks in KADS-1 can be used to define a typology of inference functions, that we call correspondly the *basic inference functions*. Again, naming issues are open, but the names have to keep their correspondence to stress the common typology behind them. We may see the basic tasks as "filled" in basic inference functions, or redefine the basic tasks as a sum of a basic inference function, an inference structure and a control structure (i.e., a generic model). For the discussion on basic inference functions we refere the reader to the discussion on basic tasks in Appendix **??**.

Clearly other typologies can be derived along similar lines. For example CommonKADS abandoned the idea of using a KL-ONE ontology as the only possible basic ontology. Canonical inference functions as defined above are relative to the underlying representational ontology and a similar exercise can be done for any other ontological choice.

4.7 Summary

In this chapter we have described in more detail the basic modeling components and their relations (see also Figure 4.1):

- inference functions
- inference structures
- control structures
- domain models
- case models
- domain ontologies
- problem solving methods

In addition we have shown how typologies of such components allow for extending the library.

Chapter 5

Generic Models

As mentioned in chapter 2, the library consists of two basic types of entities: basic modeling components and generic models. The basic modeling components were discussed in Chapter 4. In this chapter we focus on the generic models. We will initially discuss what generic models look like (Section 5.1) and describe how this part of library is structured (Section 5.2). Section 5.3 expands on the relation between generic models and basic modeling components, and gives examples of classes of generic models that have been described in the literature. Several of the appendices in this document discuss such classes of generic models in more detail (role-limiting methods, problem solving methods for construction tasks, basic tasks), albeit not yet in a fully integrated way. The design of the generic model part of the library is illustrated in Figure 5.1.

Figure 5.1: The library consists of partial (but in form complete) expertise models described at various layers of precision and organized in useful partial model development graphs.

5.1 The form of a generic model

The goal of modeling with the library is the construction of a complete and coherent expertise model. Generic models must serve templates and platforms for this process to start from. It is natural to think about generic models as partial (but in form complete) expertise models. The form of a generic model is therefore the following (directly borrowed from the decomposition of the expertise model as described in [**?**; Figure 6]. The further structuring of the frame (a kind of "tree of knowledge") follows the specification of [**?**] as discussed already in Chapter 3 and illustrated in Figure 3.1. Every generic model in the library is a partial instantiation of this frame. For some generic models only one slot may be filled in. For others all slots may be specified.

Basically there are two reasons for using only complete expertise model frames as generic models in the library. One is the relative interaction hypothesis [**?**; Section 2], the other is flexibility in modeling [**?**; Section 6]:

- The relative interaction hypothesis is one of the basic principles underlying the CommonKADS modeling approach. It has mainly implications for what counts as a generic model. The interaction hypothesis was introduced by Chandrasekaran to indicate that knowledge form and use could not be meaningfully separated. This hypothesis implies that the lowest meaningful level of granularity for generic models consists of a package of domain, inference and task knowledge. On the other hand the KADS-I view was that each of these three knowledge categories could be modeled relatively independently of each other. In CommonKADS the idea is that these are two extremes of a spectrum of possibilities and that the methodology must be able to accommodate any level of interaction between different knowledge categories. Consequently no implications follow for the level of granularity of a generic model. A model is generic if it is a useful platform to start the construction of a complete expertise model. Such a platform is now viewed as a partially instantiated expertise model frame.

- The KADS-I view is that the inference structure is preferred as "the initial or pivotal element in conceptual modeling"[1]. The Generic Task approach [?] would rather take task knowledge as the entry point. In Steels' Components of Expertise [?] domain knowledge plays a more prominent role in the initial phases of modeling, after which the focus may shift on task knowledge and decomposition knowledge. Clancey's NEOMYCIN [?] shows the importance of task structures while his notion of inference structure, as exemplified in the heuristic classification model [?], seems closer to taking the domain structure as a starting point. McDermott's analyses of problem solving [?] is centered around problem solving knowledge though it is closely coupled to task knowledge. But, no matter where one starts from it is necessary to fill in at least task, domain and problem solving knowledge (at least with inference and action methods).

It follows that it is not possible to make claims about the level of granularity of meaningful generic models, neither about the nature of the generic models that serve as entry points into the library. A single expertise model frame that can be instantiated to various levels of specificity accommodates these consequences. For example a KADS-I interpretation model becomes an expertise model with the inference structure and control structure filled in. In CommonKADS a variety of generic models co-exist [?], for example role limiting methods, instantiated problem solving methods, generic tasks, and combinations of these. The advantage is that the methodology is not restricted to one "modeling paradigm" but accommodates a variety of modeling approaches. Providing for a certain type of generic model implies a bias toward a certain modeling practice, namely one in which the generic model is taken as the appropriate entrypoint in the modeling activity. Our approach avoids this bias.[2] It is true, of course, that coping with this variety requires more sophisticated modeling tools.

[1] Note by Joost Breuker, September 20, 1991

[2] There are other advantages of a totaly different nature. The fact that the library of generic models consists of expertise frames that match the full CML definition implies maximal expressiveness and transparency of library elements as well as an interface to modeling support tools that requires no additional translation. Incremental modeling and structural guidance (missing pieces are clear from the model) are additional advantages.

5.2 Modeling operations as library structure

What are the relations between the generic models? As argued in Section 2.1 an answer to this must be found by taking a pragmatic stance: how will the library be used? It was already pointed out above that the CommonKADS methodology is supposed to be more flexible with respect to the process of modeling itself. This does not imply that the contents of the library should be different depending on how you model. It does imply that the way in which the library is traversed and the model composed from partial ones is different.

The modeling process itself (as it is being studied in Task 1.5) can be viewed as a search process in which a model development graph is being build up. At each moment the CommonKADS expertise model frame reflects the status of knowledge level modeling. Modeling operators operate on expertise models. There are a number of such operations:

- specialization (e.g., specializing into a specific kind of diagnostic task)
- instantiation (e.g., filling in the contents of a domain model)
- combination (e.g., applying a role limiting method to a task)

This taxonomy can and should be refined, e.g. according to the type of specialization being done or on what knowledge category it works. The point is that the library must contain useful partial paths of the modeling process so that, once you get in, it guides the application of operators by offering different options and alternatives. In some cases, as illustrated in Figure 5.2, a complete model development graph can be obtained by a few initial selection steps (based on task features) and a final instantiation step (for example to fill in the domain model). In practice several model construction operations will need to be done outside of the library, of course.

Figure 5.2: The structure of the library supports modeling by providing a set of useful partial model development graphs. These document the relation between generic models that can be obtained from one another by the application of modeling operators.

By making the modeling operations explicit we want to achieve a level of guarantee that all the necessary steps have been taken to ensure internal consistency of the model. A modeling operation is more than a single addition to a model. The intention to apply a modeling operation generates an "agenda" of tasks to be done to bring the model into a new internally consistent state. The application of a decomposition method (present in the problem solving knowledge) to a task (present in the task knowledge) will influence the domain, task and inference structures of the global model.

The structure among generic models thus allows for documenting the genesis of a generic model. For example, it could be documented that a model of systematic diagnosis is, for example, the result of applying a generate-and-test method to a hierarchical decomposition model. It is now for instance possible to trace back the origin of an inference structure as an "expanded" version of another one by the application of a decomposition method instead of an inference method. This gives us the effect of hierarchically structured inference structures for free. Also it is possible to move easily back and forth during the modeling process, for example having retrieved a fairly complete model of which only the domain organization is not quite right, it is possible to decompose it and recover the useful parts as the basis for a new combination step.

(Add arguments that this structuring is really more powerful than structure among basic components, e.g. for superimposing models).

5.3 Classes of Generic Models

It is interesting to notice that the two ways of extending the library (adding typologies or adding generic components) correspond to two ways of grouping components. On the one hand, generic models are made by defining, basically, meaningful sets of different basic components (e.g. one inference structure and one control structure). On the other hand, typologies of components are made by defining meaningful sets of the same component.

In the literature one finds several classes of generic models: interpretation models [**?**], role-limiting methods [**?**], generic tasks [**?**] and so forth. Usually these generic models are tight to particular modeling approaches. It is tempting to introduce typologies of generic models, just as we did for basic modeling components. However, since generic models are made by basic ones, such typologies would probably refer (explicitly or not) to typologies of basic components. Therefore we do not define typologies of generic models since it would not add any power to the library (in fact, it could restrict it). We find it useful however, to refer to *classes* of generic models, since they can be related to notions from the literature (such as generic task). For guiding modeling however, the boundary of a class should not be "visible". At this point the knowledge engineer does not care what class of generic model he/she is working with.

Some classes of generic models are listed below.[3]

Basic tasks Although the KADS-1 interpretation models have been shown to have certain limitations, they have also shown to be very useful. The basic tasks here proposed are the correspondent to the interpretation models in the old KADS-1 library [**?**]. The name is used because they are based on exactly the same ontological (functional) distinction that generates the basic inference functions.[4] As generic model, they are a set of an inference function, an inference structure and a control structure. As a typology, not every set of these basic elements is allowed, but only some. The ontological distinction that generates the typology is system-centered. The detailed discussion of basic tasks is made in Appendix **??**.

Role Limiting Methods The discussion provided in [**?**; **?**] gives us a basis to include role limiting methods as proposed by McDermott [**?**] in the library.[5] For what we

[3] In the following description, many aspect relating to domain issues are not dealt with in detail. This should be completed later on, as soon as CML domain issues are better explained and resolved.

[4] The name may be somewhat misleading. Maybe *system-centered tasks* or *problem-related tasks* could be better; another option could be to maintain the old name (interpretation models).

[5] The study is particularly interesting because domain issues were addresses in detail.

could derive from these sources, role limiting methods are composed by two expansion methods and two control methods (that apply in succession), and *many* features, both about task and domain. It is still open whether they are also a typology. Role limiting methods as generic models still have to be detailed and discussed, and the present definition is only initial. It is interesting to notice, however, that one differentiation factor between role limiting methods (other than the two step refinement in terms of our library) is their strong emphasis on domain issues. Since these are not treated in detail in this document yet, the full potential of role limiting methods is unfortunately hidden in our simple interpretation.

Complete methods A complete method is a combination of an expansion and a control method. It applies to an inference function and delivers both an inference structure and a control structure. Following the definition of expansion and control methods it is clear that a method must be characterized as follows:

- name
- description
- expansion method (specifying the inference function to which the complete method applies and the inference structure the method delivers)
- control method (specifying control structure the method delivers and the inference structure[6] to which the control applies)
- rational (of the method)

Complete methods may have an extended rational from their respective sub-methods (if it makes sense to make them explicit).

Generic tasks We may implement generic tasks in basically two ways. In the first, they would be complete tasks (i.e. a inference function + an inference structure + a control structure) at a certain (pragmatic) level of abstraction. In the second, they would also contain domain knowledge, or at least certain domain models or structures. If the first definition is assumed (which is less likely), we could regard basic tasks as defined above as particular types of generic tasks, defined with the help of a typology. Following the basic bibliographic reference in [**?**] the latter interpretation is more likely. Again, this interpretation has to be carried out in more detail, and an extensive discussion has to be included later on.

Fields The notion of field refers to a typology of application domains that can be regarded as the counterpart of the typology of basic tasks (inference functions) in the domain division. The concept of field is a new one[7]. It is still unclear what is their exact nature but as a first definition, we propose that fields are a set of ontologies and domain models. The latter is derived from the former. A full discussion of the issue is still lacking, and will be added in future refinements of this document.

Table 5.1 summarizes the examples of classes of generic models in the library.

[6] the inference structures of the expansion and control method should of course be the same.

[7] The notion of field has been proposed by Joost Breuker within KADS-II

Extension	Definition	Bibliographic References
Basic Tasks	Inference function Inference structure Control structure	[?]
Role Limiting Methods	Inference functions Expansion methods Control methods	[?]
Complete Methods	Expansion Methods Control Method	this report
Generic Tasks	Inference functions Expansion methods Control methods	[?]
Fields	Ontologies Domain Models	this report

Table 5.1: Examples of classes of Generic Models in the CommonKADS Library

The above examples indicate that we tacitly assume that any expertise model can be rewritten as a structure of basic modeling components. However, this does not mean that such a translation is easy or simple. Sometimes the whole theory these components are inserted in has to be reinterpreted in CommonKADS terms in order to allow such a translation. Given the proliferation of terms and theories in the field and the lack of unification between them, this is probably a very difficult task.[8] Therefore, a major effort in the re-interpretation of other modeling approaches and modeling components is necessary to enhance the CommonKADS library. The aim of this effort must be to document as much as possible the compositional links between a generic model and the basic modeling components that are at the basis of it.

5.4 Summary

In this chapter we have elaborated on the second type of library entities: the generic models. We have argued that generic models are needed in order to accommodate the relative interaction hypothesis as well as for integrating different approaches to modeling. Of course generic models are not independent of basic modeling components. In fact the latter are often needed to further instantiate (fill) the former. In addition we have argued in favor of documenting the structural relations between a generic model and the basic modeling components that constitute its parts. In addition structure over generic models is provided by various modeling operators that represent in the library structure useful paths of modeling and document the origins of a generic model. Finally, we have shown how generic models can be used to capture different classes of generic models from the literature.

[8]The problems of integrating Components of Expertise [?] into KADS clearly illustrate this point.

Chapter 6

Use of the Library

So far in this report, we have dealt with the structure of the library trying to maintain as much as possible a high degree of independence of issues related to their use. In this chapter, we will deal with some of these issues. However, we still try to maintain independence from approach specific aspects. That is, we will discuss some issues that relate the structure to the use, but the discussion will be done independently from the specific modeling method used. As much as we claim that with the proposed structure for the library it is possible to implement different modeling components, we claim that with the tools we provide here to specify how to use the library it is possible to use different modeling methods. This maintains a complete independence of the modeling approach – both structure and use, both modeling components and modeling methods.

Presently, the chapter is not completely structured, but has more a series of sections presenting some concepts or discussing some specific aspects. Also, some aspects here discussed are more of the scope of tasks I.1, I.5 and I.6, or involve of other aspects or subtasks within task 1.3.

6.1 Accesses and Routes

The library is structured to provide support for its use. This use consists of the retrieval of library components through navigation on the library items. We distinguish *routes* in this navigation. Some routes are structurally supported, and also indicated or prescribed to the user. Other ones are possible, but not recommended. The analogy with a real library would be to go directly to the shelves without searching before in the catalogues: it is allowed, but not recommended or supported. Within these two extremes – completely guided routes and completely unsupported ones – many alternatives may be used in the design of the library.

To support specific routes in the use of the library, we will distinguish structural *accesses* to the library. They are collections of components which are particularly useful as starting points to routes. We also distinguish between *primary* and *secondary* accesses. The primary ones are in some sense prescribed; i.e., recommended by the methodology. The secondary ones are accesses which are starting points to routes that are not recommended but still supported at some extent, and therefore may also be used. This distinction reflects a distinction (preference) between different routes (related at a certain extent to modeling methods).

6.2 Task Features and Component Features

In KADS-II, the basis for providing guidance and structure in the modeling process is the use of *features*. The details of their use is part of the task 1.5 work, but some aspects are relevant to be commented here.

In principle, to retrieve and use a component of the library means also to assume a certain set of constraints and premises that are intrinsically attached to the component. For instance, to use an expansion method 'generate and test' (and its associated inference structure), it has to be possible to test the hypotheses. This generic assumption is related to domain issues, and may need to be detailed in domain terms – i.e., it may be necessary to specify in more details which domain assumptions are consequence of this more generic one.

While retrieving a component in the library and adding it to a model being constructed, the user should have in mind that also some restrictions (features) have to be obeyed. The final conceptual model will consist of various conceptual knowledge items *and* a group of features. These features may of course be discarded at the end, since they are in principle only useful during the modeling process. However, taking into consideration the possible need for these features in later phases (e.g. the construction of the design model), it is more natural to keep track of them at least as a means of documentation.

Task features have been defined in Task I.5 as any characteristic of an application and its environment (the "task reality") that is relevant in selecting an appropriate model or model components for it. These task features have been grouped as features relating to the domain, the task purpose, the environment and to grounding. The set of basic modeling components that is being used in the library suggests another class of features, namely *component features*. These component features connect or link components of the library by relations of:

(mutual) component necessity When a library component can only be used when some other is also used.

model necessity When some component can only be used if certain features and/or elements are present in the conceptual model. The typical case is domain knowledge availability.

(mutual) component exclusion When the use of a library component excludes the use of another library components.

model exclusion When the use of a library component excludes the presence of other features and/or elements in the conceptual model.

See also figure 6.1.

Figure 6.1: Typology of library component features

The *component necessity* and *component exclusion* features are particularly helpful in guiding the navigation through the library. For instance, the library should keep a trace of a consultation to verify mutual constraints and advise (positively or negatively) the retrieval of components using the features of the components already retrieved.[1] On the other hand, the *model necessity* and *model restriction* features are particularly adequate to help the user verifying whether or not a certain library component is compatible with the model being constructed. While the *mutual* features refer to constraints within the library, the *model* ones refer to restrictions external to the library – i.e., in the conceptual model being constructed.

Since some of the library components overlap in knowledge content, it seems that their features will too overlap to some extent. For instance, an expansion problem-solving method that points to a certain inference structure in the library will probably have many features in common with the inference structure itself.[2] More generally speaking, at any situation in which the choice of a component necessarily implies the choice of another (even if implicitly), the features of the second will be also in the first.

6.3 Operations on the Expertise Model

Since the use of the couple CMED/CKL concerns the process of building a conceptual model of expertise, it should be clear which *operations* can be done with respect to this model. These operations will reflect on different classes (types) of operations in the use of the couple CMED/CKL.

A first aspect to be specified is how (i.e., in which situation) will the user start to use the library. Since the construction of a conceptual model starts from the leaf tasks in the *task model*, we can assume that there is a task already defined (although only in general terms) when the model of expertise starts to be built. In other words, this task definition is the necessary connection between the task model and the model of expertise; the endpoint of the first and the start point of the second.

Therefore, the user will start the use of the CMED/CKL with an empty model of expertise with a single task. What should be filled is the task body of this main task (i.e., its inference structure and the control structure over it) and the domain knowledge necessary to operationalize the task (both filling domain knowledge roles and primitive inferences). These two needs correspond to the divisions in the library, and give us a basic distinction between two types of operations: the *task division operations* and the *domain division operations*. This distinction is not vacuous, but reflect an inherent different nature of the two divisions.[3] Note, for instance, that there is a mapping between domain and task/inference knowledge (provided by the domain roles), but there is no distinction between inference and task (at least in the library). Consequently, it seems appropriate to establish and use

[1] It is a design decision whether or not to effectively forbid or oblige the user or only kindly advise.
[2] It is even debatable whether they can have different features at all.
[3] See, for instance, the discussion about knowledge roles in D1.1

the two divisions proposed to the library also in the CML editor.

The inference and control structures are *layered*, i.e., they are constructed in many levels which provide details about how parts of the upper level will be accomplished. Therefore we can distinguish two other sub-types of operations in the task division: *single-level* and *multi-level* operations. Single-level operations work based on knowledge differentiation, i.e., they modify and adapt an existing element of the model[4], given some specific modeling needs. Multi-level operations are the ones which generate domain elements in a different level – by either decomposing (expanding) or assembling elements of one level into elements of another.

The types of operations on the conceptual model are summarized in figure 6.2.

Figure 6.2: Types of operations on the library

6.4 How to Decide upon an Operation

In this section we will discuss the guidelines to the use of the CMED/CKL – in other words, how to decide upon an operation. As we have already mentioned, this guidance is primarily provided by *features*. Features are being used to guide in the application of operations on models. This guidance is provided by the so called *modeling guidelines* that have been proposed in Task I.5 work. The modeling guidelines can refer to either task or component features. It can also be conditional over the state of the model it is being applied to and the state that is being aimed at. When applicable, a modeling guideline gives a qualified recommendation for an operation that is itself parameterized over library elements. For example a modeling guideline may recommend to apply a method to a task, but it may also recommend not to apply it. The modeling guideline is completed with a way to verify the appropriateness of the operation (validation procedure). In summary a modeling guideline has the following attributes (see KADS-II/T1.5/WP/VUB/002/1.0 for a more complete treatment of modeling guidelines):

```
modeling guideline
   task-features                       <some task features>
   model-features                      <some model features>
   current-state                       <a model state>
   target-state                        <a model state>
   external-state                      <a milestone state>
   operator                            <a modeling operator>
   library-component                   <some library-components>
   preference                          <a preference indicator>
   validated-by                        <a validation procedure>
   suggested-next-operator             <a modeling operator>
```

Operations make intrinsically different modifications in the set of features of the model. Modeling guidelines can help keeping track of the (hidden) implications of each operation.

[4] Which can be also components of the library, but not necessarily so.

The typical example is the retrieval of a library component. When this operation is done, the features of the component being added are appended to the list of features of the model. By analogy, deleting an element implies that its related features have to be removed from the list. However, it will be difficult to keep track of this list of features when a modification operation is performed (particularly in the case of inference structures), for instance over a library component that was previously retrieved. Some of the features may refer to the overall element, and it is very difficult to know if they are still valid or necessary when the standard library component is modified, or if other features should come into place. The idea of connecting the features to the operations themselves could be used to solve this problem. For instance, when a canonical inference is deleted from a canonical inference structure, the library could, activated by the operation, verify which modification have to be done on the set of features of the working model because of the operation. At the same time, the kind of features that could change might be stored in the operation itself. A weaker version of this idea would be to store in the operations only the possible changes, in a way which indicates to the user which features might be affected without modifying them directly. This looks more coherent also, since the knowledge about which types of assumptions and restrictions are affected by an operation is more intrinsic to the operation itself than the specific change in the library elements.

Another kind of knowledge that can be used to decide upon an operation is a *modeling strategy*. This is the overall approach to knowledge modeling, reflected in the way the library is used. It is doubtful whether or not it is possible (or even within the scope of the project) to provide an intelligent overall guidance to the user in the form of a both active and passive intelligent help system (see, for instance, [**?**], it seems reasonable to embed these possible strategies in the library in some way. They would probably be represented as *plans*.

In summary, the information about how to decide upon an operation would be stored as:

- task features
- component features
- model state
- modeling strategies and plans

6.5 Entering the Library: Basic Tasks and Fields

The major difference between CommonKADS Library of Re–Usable Models and its predecessor –the KADS–I Library of Interpretation Models[**?**] is that it contains also domain models, besides models of the reasoning strategies, i.e. the CommonKADS Library of Re–Usable Models has a task division which contains the reasoning models, and a domain division filled with basic models of domain knowledge (see Chapter 3).

The access to the models in the KADS–I Library of Interpretation Models was provided by a taxonomy of Generic Tasks. This taxonomy was based upon an 'ontology' derived from Clancey's view that a KBS is a 'model of some system in the world' [**?**; **?**]. From this notion the KADS–I project derived a pseudo–'ontology' in which the Generic Tasks reflected the notion of '*what can happen to a system model*', or, what functions (operators, procedures) can be applied to a system model. In the CommonKADS Library of Re–Usable Models we will maintain the same 'ontology', but we will extend it now to domain models as well. For the domain division the taxonomies are simply based upon the question *what types of 'systems in the world' (system models) can be distinguished.*

6.5.1 Task Division routes

Both questions have further implications. These implications create the refinement **routes** in the CommonKADS Library. As stated in Chapter 3, *inference functions* as basic components of the description of reasoning strategies can be refined by *expanding* them by applying expansion (p.s.) methods. Expansions answer the question what we mean by some function, by intentional definitions, i.e. one can say that a function or task is redefined, rather than decomposed.[5] The inference structures that result from expanded inference functions can be *elaborated* by applying control (p.s.) methods, which add control to the structures, thus resulting in task structures. The expansions and elaborations create the stepwise refinement routes in the Task Division of the CommonKADS Library. In general, access and refinement are accomplished by checking whether specific features are present in the domain of analysis. These features can be part of the model –e.g. in diagnosis whether the solution is a malfunction or a defective component–, or can be assumptions that are implied by selecting a model –e.g. the presence or constructibility of a taxonomy of malfunctions. When these assumptions refer to components in the other division of the CommonKADS Library the (p.s.) method is called a *role–limiting method* [?]. Because the grain–size level of the CommonKADS Library is not fixed[6] the routes may re–enter. An inference function may be expanded by a basic task. For instance, the specification of expected values (norms) in diagnosis may be expanded to 'prediction of behavior', which is a basic task. For that reason it seems also more adequate to use the term expansion rather than decomposition for the refinement steps.

Besides stepwise refinement routes, there are also packaged refinement routes. As tour–operators may offer all–included tours, some p.s. methods –in particular the role–limiting methods– provide in one go a more or less complete model of expertise for some (sub)task. These compound p.s. methods stand for what is traditionally understood by a problem solving method.

The task–division of the CommonKADS Library terminates in *canonical inference functions.* These make up the regular interface to the domain division, because the (re–)definition of of these functions in terms of meta–domain concepts, or to be more precise, in terms of the most abstract ontologies of domains. The most abstract domain ontologies

[5] This looks as a neat distinction, but the consequences look strongly the same. (Re–) definitions consist of other terms. The notion of 'other' means that if recursion or cyclic definitions are forbidden, a definition is a composition, and the 'definiens' (i.e. the thing to be defined) is decomposed in other terms. From the point of view of looking back from the result, decomposition and redefinition is the same; from the point of view of dynamic problem solving the two are not the same, because it makes more sense to talk about 'redefining a problem in new terms' than to say that one 'decomposes a problem', because by itself a problem has no parts other than what is given and what is asked.

[6] Except for primitive inference functions.

are vocabularies for knowledge representation. In other words, the intentional semantics of the inference functions ends by referring to entities that belong to the most abstract ones in the Domain–Division.

An example of canonical inference functions and their derivation from knowledge representation terms, is the KADS–I Library of Interpretation Models typology of knowledge sources [?]. In the CommonKADS Library different typologies may be used. The one developed in the KADS–1 project is revised and presented in Appendix **??**. It is the current default, but new ones may be created in the course of further 1.3. activities.

6.5.2 Domain Division routes

The components of the Task–Division consist of inference and task–structures. Those of the Domain Division are (pre–) ontologies. An ontology is the distinctive vocabulary of a domain. Because terms in the ontologies can be relations -besides the typical objects, an ontology may be structured, and stand for abstract 'tops' of domain structures.

For the Domain–Division the refinement routes are less evident. Fields, which are the access to the domain division, are selected on the basis of their type (features). A field reflects one or more domain theories as discussed in Deliverable 1.1. [?]. Domain models form the next refinement step. A domain model is a functional view on a domain theory [?]. The ontologies of domain models may be subsets of those of domain theories (fields), but at the moment we have not enough evidence to say that this is the case in general (see below on problems with defining domain models).

In the Domain–Division of the CommonKADS Library the refinement route can be stated as follows:

field → domain model → (canonical) inference function (support role)

In the Task Division of the CommonKADS Library, components are defined in terms of other, more detailed components which are finally grounded in ontologies via canonical inference functions. In the context of Deliverable 1.1. this ontology would be called 'basic model structure' (or: '(domain) model (structure) schema') (see figure 7 [?]), which is more or less of what is meant by the basic, conceptual vocabulary of a knowledge representation formalism (see further). Ontologies are part of the Domain Division, therefore, the canonical inference functions are the natural interface between both divisions. In models of expertise the linking between the inference layer and the domain layer consists of the *domain mappings* of roles and functions. In the domain–independent definitions of canonical inference functions the mapping refers to highly abstract knowledge representation formalisms. In terms of Deliverable 1.1. (version 3), these domain–independent ontologies are 'meta–domain knowledge'. In this way, there is a grounding of the Task–Division of the CommonKADS Library. Conceptual entities can be grounded in unambiguous semantics in two ways: by reference to *computational* realizations (e.g. [?]), or by translation and validation into (logical) formalisms. Research in AI on knowledge representation is concerned with both routes. Therefore, we recommend that typologies of canonical inference structures are to be based on validated and well understood knowledge representation ontologies. Note also that the recasting of the components of the CommonKADS Library, expressed in CML, into FML (Formal Modeling Language) in the next stage of the 1.3. Task, has also this validation role.

In principle, the knowledge representation ontology to be used for grounding the CommonKADS Library should be the meta–domain knowledge as presented in Deliverable 1.1., Sec. 6.2, i.e. the CML for the domain layer. As far as this one is established (see Sect. A.1. of D 1.1. (version 3), it overlaps with the one presented in Appendix **??**.

Ontologies for domain models will be expressed in CML domain layer terms. However, for domain models and fields also other means may be available. The ontology of a domain theory may have also a *calculus*, which combines a formal and computational grounding. For instance, a number of calculus for time, space, causality, mechanics, etc. may be found in the literature. This is not only the most valid grounding, but it also supports the knowledge engineer, because he may get the description of a (specialized) inference engine for free. Wherever possible, the library should contain references to these calculus. It should be noted that the grounding of the ontologies only in terms of the domain CML is a weak and minimal one. A calculus is the most valid foundation one can have for the ontologies in the library.[7]

[7] There are some problems with the notion of domain models that need to be resolved. Domain models are functional partitionings of domain theories. Steels presents as examples: functional, causal, teleological, structural, topological, etc. [**?**]. This functional perspective poses problems in the CommonKADS Library, because the functional viewpoint is in fact part of the Task–Division. If we want to have only 'interaction' in Chandrasekaran's sense between domain knowledge and the functional perspective at the level of canonical inferences we should avoid these functional perspectives in the Domain–Division. One may argue that this is not much of a problem, because one may use straightforward function–dependent links between the two divisions: a kind of shortcuts, where some inference function directly pre–supposes some domain model, as e.g. in role–limiting methods. However, the real problem is then that a model with the same name and role (function) may have completely different ontologies (definitions). For instance, a causal model in medical diagnosis has or may have a completely different ontology, than a causal model for troubleshooting audio–equipment. Moreover, the same ontology may be used for different functions. A causal model and a teleological model may have the same ontology –in an application: the same knowledge base–, but may only differ in the reasoning perspective. For instance, it rains to make the plants grow vs. plants grow because it rains. Because generic (re–usable) domain models may neither have a unique ontology, nor an exclusive one, domain models may not be the right components for the CommonKADS Library. This does not mean that the notion of domain models may not be very useful in constructing models of expertise. On the contrary, and this notion should be used in partitioning application domain knowledge, but one may doubt whether they can exist as 'meta–domain knowledge' elements in the CommonKADS Library. In the 1.1. Deliverable, Sec.6.2, domain models are part of the application domain knowledge. However, the ontology of models is assumed to be part of the meta–domain knowledge. At least in Steels' view such generic role of domain models is not intended [**?**], p. 39–40.

What domain models of the same function *may* have in common are relation terms. For instance, causal models should have some causal relationship explicitly represented, or derivable. Causal relations come in large varieties, but their meaning is at least a non–domain dependent topic of research in AI. Therefore, we may propose to replace the notion of domain model as a refinement step in the Domain–Division by the notion of *relation*. In KADS–I a similar view was held but it went never beyond subsumption and aggregation ('part–of') relations. See for more and better details [**?**], p. 45–46.

6.6 How to Specify a Modeling Approach

To understand how to implement or specify a modeling approach or a modeling method, we should first make clear what they are – more specifically, what such things contain and define. Of course, this is a matter of hard and hot discussion in the AI community (and particularly in the Knowledge Representation and Acquisition communities). Therefore, we will only show our point of view over the subject: we do not want to enter that debate, but only use some concepts to aid our (local) task.

In the first place, it is important to make clear that we are dealing with *knowledge* modeling approaches and methods. All approaches we are analysing have a common goal, which is to produce knowledge-based systems through the construction of a certain number of knowledge level models.[8]

A knowledge modeling approach can define a number of things:

- A *knowledge theory*;
- An *ontological view* on knowledge based on this theory;
- A *knowledge representation scheme* based on this pair theory/ontology;
- A number of *knowledge level models*;
- A number of *knowledge modeling components*;
- One or more prescribed ways to use these components in order to build the desired models – i.e., a number of *modeling methods*;

Only some of these concern us, since the library only involve or support some of them. The two last items are the ones we should pay attention. In the first place, the knowledge modeling components are developed on top of a knowledge representation scheme (sometimes a language), and they represent a partition on the knowledge to be modeled into a number of items. This partition does not represent the same ontological distinctions made by the representation scheme itself, neither (commonly) have a strong theoretical status. they are a more pragmatic partition, conceived to help the modeling process. Using the terminology proposed by Lenat and Guha [?], the knowledge representation scheme is epistemologically adequate, while the knowledge modeling components reflect a highly pragmatic preoccupation. That is, the first is designed to be as much as possible *independent* of the modeling process itself (i.e. its use), while the second has the opposite design concern. Of course, between these two extremes many levels can exist, with various schemes or sets of components at different levels of epistemological and pragmatic adequacy.

In summary, the modeling components are, within the context of a modeling approach, the tool which is designed to be adequate for the pragmatic concerns – i.e., their efficient and practical use in the modeling process. The use itself has also to be designed, and this is done with the *modeling methods*. They give a more thorough specification of how

[8] Knowledge level as proposed by Newell [?].

the modeling process may (or should) be done. In particular, they specify how to use the modeling components to build the desired knowledge-level model. That is the relation between the modeling components and the modeling methods within the context of a knowledge modeling approach.

For what refers to *a* library only (not necessarily *our* library), to specify or implement an approach it is necessary to specify the base knowledge representation scheme, the set of modeling components and the modeling methods. Since our library imposes a knowledge representation scheme (the pair CML/FML), the first step is fixed, and only the latter two remain. Thus, to specify a knowledge modeling approach in the library, it is necessary to:

- Specify the set of modeling components used in the approach;
- Specify the modeling methods used in the approach;
- Interpret the modeling components in terms of Common KADS CML, and more specifically in terms of the library basic components;
- Use this interpretation to adapt (if necessary) or simply re-specify the modeling methods to Common KADS terms.

One interesting open question is the lack of a language or framework for the specification of modeling methods. Until now, although some of them have been formally treated or presented, the AI community is far from a consensus of how to describe a modeling method. The work presented here suggest some results in this direction, but a full blown study is certainly out of our scope. Evidence is growing that the effective usage of generic models is far from being solved. In [**?**] essential problems with selecting and using generic components are pointed out, based on practical experience:

1. The discrimination problem.
   Several generic models may apply. How to select the most appropriate one?
2. The overgenerality problem.
   Selected model is too general to give useful support. How to arrive at a more specialized model?
3. The modification problem.
   Selected model is not directly applicable. How to modify it?
4. The terminology problem.
   Selected model is described in a strange vocabulary. How to map it onto the ontology of the application?
5. The risk problem.
   Selected model may turn out to be wrong. How to minimize the costs of selecting a wrong model?

Based on these problems a set of guidelines for a knowledge modeling methodology is being suggested [**?**]:

- Combine bottom-up and top-down analysis.
  Bottom-up analysis may provide useful pointers to, and initial assessment of, generic models.
- Multiple selection criteria are needed.
  In addition to knowledge availability (the initial criterion used), other criteria suggested are: domain complexity, the expert's representation of the domain, information reliability, real world interaction, domain stability.
- Provide composable models.
  Task features should also point to schemas of combinations for generic models.
- Integrate modeling and model validation.
  Check expectations of real world behavior indicated by selected models.
- Provide for an extensible library, with annotations based on experience.
  Experience from using the library must be captured.
- Explicitly document the assumptions of generic models.
  Underlying assumptions should be used in the model selection process.
- Allow for partial models as end product.
  Weaken the role of the knowledge level model as a complete intentional specification.

The library as it is being proposed provides for the facilities to deal with these issues.

6.7 Summary

In this chapter we have discussed a number of issues related to the effective usage of the library. We have elaborated on the concepts of task feature, model feature and modeling guideline. We have also put forward some ideas on how to specify modeling approaches and have pointed out some points of current concern in current work on using generic models as a basis for a development methodology.