KADS^{II}

Generic Library Facility: Specification

Walter Van de Velde

Document Id.	KADS-II/WPM/TM.8/DD/052/3.0
Date	December 22, 1993
Distribution	Consortium, EEC (public)

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
Pleinlaan 2, B-1050 Brussels
Belgium
Phone: +32 2 641 641 3700 ; Fax: +32 2 641 3729
Contactperson: W. Van de Velde
E-mail: walter@arti.vub.ac.be

This document was produced using the latex wordprocessor

Generic Library Facility: Specification

Walter Van de Velde

Synopsis

This document contains the specifications of a generic library facility for the CommonKADS workbench. The generic library facility is to be used for creating libraries of re-usable models for any of the models in the CommonKADS model set. The project currently only targets libraries of design and expertise models but the generic facility is supposed to be more general than that. Appart from this genericity the library facility incorporates indexing and structuring mechanisms. On the other hand integrity checking, safety, efficiency and many of the features that one would expect for massive storage facilities have not been major objectives.

Document Id. KADS-II/WPM/TM.8/DD/052/3.0
Date December 22, 1993
Distribution Consortium, EEC (public)

Revisions

Author: Walter Van de Velde

Revision date: Feb 3, 1993

Modifications: Initial version 0.1.

Author: Walter Van de Velde

Revision date: May 5, 1993

Modifications: Preparation of version 0.2 for library meeting in Stockholm (May 10, 11). Adaptation of bits and pieces from previous 1.3 work.

Author: Walter Van de Velde

Revision date: Jun 2, 1993

Modifications: Drastic revision following a less bla-bla scheme of sub-system descriptions and functionalities. Distinction between reference, query and operation added.

Author: Walter Van de Velde

Revision date: Jul 27, 1993

Modifications: Only three categories of library entities; orthogonality requirement for modeling components is out; detailed definition of library ontology definition.

Author: Walter Van de Velde

Revision date: Aug 17, 1993

Modifications: Start revision based on feedback from Yann Guerin, email communication dated August 11th

Author:	Walter Van de Velde
Revision date:	Oct 14, 1993
Modifications:	Explicit data model added

Author:	Walter Van de Velde
Revision date:	Oct 18, 1993
Modifications:	Example of design model and expertise modeling. Illustration of modeling operations and adaptation of data model accordingly.

Author:	Walter Van de Velde
Revision date:	December 22, 1993
Modifications:	Update to 3.0. Mainly formatting, adding figures, clarifying paragraphs and references. Version for EEC review.

Contents

1 Introduction

This document specifies the requirements for the Generic Library Facility (GLF). These requirements cover functional and non-functional ones and are user-oriented. The GLF will be implemented as a kernel facility in the CommonKADS workbench. How to do this is not within the scope of this document.

This document is the main reference for the description of the Generic Library Facility from an end-users perspective. It is also the main reference for implementation of the facility. Finally it sets the gold-standard for the validation of the facility.

This document should be updated if the specifications get refined or evolved as a result of design decisions and negotiations among interested parties.

The discussions and specifications in this document are based on previous work on the expertise model library [Aamodt et al., 1992b], the design model [Van de Velde, 1993a] and on task features [Aamodt et al., 1992a]. The author acknowledges the respective members of the project team for their influence and contributions.

2 Requirements Statement

This section describes the problem as it is currently perceived that these specifications are addressing.

2.1 Problem Context and Scope

The aim of the KADS-II project is the development of a comprehensive methodology for the integrated development of knowledge based systems. This includes the development of tools and techniques to support the various interest groups in all aspects of the system's life-cycle.

CommonKADS relies heavily on a modeling view. In a CommonKADS project a series of models are developed that, together, describe the CommonKADS solution to an identified problem. The creation of these models can be done from scratch but the methodology relies heavily on the availability of a series of ready-made abstract or partial models that can be reused in different projects.

> *The problem that the library facility addresses is the storage, management and retrieval of models or partial models and the interface to their usage in the modeling process. This problem is a real one, given that libraries of such models may exist for different models in the CommonKADS model set and that there may be a large collection of such models. Moreover, the models themselves and the dependencies between the different models can be of such complexity that their management and use without a dedicated tool is impractical.*

The KADS-II project also develops an integrated workbench. The purpose of this workbench has been defined to be mainly a demonstrator of important CommonKADS concepts and methods. The CommonKADS modeling view, the CommonKADS life-cycle model and the CommonKADS quality viewpoint are three important aspects that the workbench should demonstrate.

The generic library facility is a support facility for the CommonKADS modeling view. It is intended to support modeling by the provision of libraries of generic and re-usable models or partial models. From the early days KADS has been applying this idea by providing for a library of interpretation models. This was not much more than a tree of generic tasks (labels supposed to be meaningful to the knowledge engineer) with the leaves indicating an interpretation model. In CommonKADS the idea is much extended. The extended size and scope of the library concept calls for a thorough re-thinking of the supporting tool.

2.2 What is needed

At a high level the needs are quite clear:

> *What is needed is a tool that supports the storage and effective retrieval of re-usable CommonKADS models, or more generally, any material that can serve as a stepping stone for the knowledge engineer or KBS developer.*

The library concept of CommonKADS is an extension of the older concept in KADS-I. The following are the main differences and the needs that follow from these:

Model set needs: First re-usable models can be provided not only for the expertise model but for any of the models in the CommonKADS model set. Within the project focus will be on expertise and design models.

What is needed are facilities to create multiple libraries for different model types while, preferably, maintaining a common infrastructure, look and feel. In addition such facilities may anticipate changes in the model set definition. Facilities to maintain the set of libraries (for example merging of libraries) are also desirable. Such needs are also justified by the fact that the construction of an adequate library can not be the result of a single project but the moving target of the wider community of CommonKADS knowledge engineers.

Storage needs: Second the range of re-usable models is being extended to cover just about any meaningful (i.e., indexable) part of a model. Put extreme, the library stores anything that can be meaningfully retrieved. For expertise modeling, for example, the old interpretation models are only one part of the library.

What is needed is a functionality for storing and retrieving a collection of abstract or partial models or parts of them that can be used to model a given application context. In addition, any piece of information that can be instrumental in modeling but is not necessarily a literal part of a model should be handled (problem solving methods are such for the expertise models). The libraries should allow for an organization based on structural (e.g., part-subpart) as well as on taxonomical grounds.

Indexing needs: Third there are several ways to retrieve the models, i.e., the old tree-structure of generic tasks is replaced with (or augmented to) a network of links. Task and model features are used to provide meaningful indices to the library elements.

What is needed are functionalities to annotate library elements with task and model features and to use these for the focussed search and retrieval for useful library elements. Navigation based on features should be largely orthogonal to the structural-hierarchical organization of the library. Such a feature facility should be open ended to be able to accommodate changes in the feature set definition.

Modeling support needs: Fourth the library supports several modeling approaches. Top-down modeling, interpretation model guided, modeling based on configuration of partial models, and so forth: all of these should be supported by the library.

What is needed is a structure of the library based on useful modeling steps so that the storage function of the library is adequately complemented with the modeling support function of the library. These are the needs that follow from the role of the library as a support tool in the modeling process, i.e., a store of modeling approaches and modeling episodes toward a stated objective of modeling an application context.[1]

Inter library needs: Fifth, when exploiting the CommonKADS model set, modeling decisions in one model can and, in fact, should have an impact on other models. Models are created to understand better a problem in a KBS application context and to make decisions about the CommonKADS solution that will be worked out. Thus important guidance is in the implications from one model to another.

What is needed are facilities for links or references between the libraries. In effect and ideally the set of libraries could behave like one huge library. For example, modeling decision in the expertise model can lead to modeling decisions for design. Otherwise no advantage can be gained by using different CommonKADS models.

2.3 Application context

The library facility will be used within the CommonKADS workbench. One can envisage several scenarios of its use by different users (knowledge engineers, library managers, apprentices). There are at least the following users of the Generic Library Facility:

1. CommonKADS methodology specifier

   The CommonKADS methodology specifier takes the decision to develop a library for a certain CommonKADS model. He or she may take additional decisions on what models can be stored and how they should be indexed. He or she uses the Generic Library Facility to create a model specific Library Tool.[2]

[1] It is important to distinguish this from LCM support in the risk driven approach. The risk driven approach leads to the formulation of modeling objectives and is driven by risk-management guidelines. Reaching these stated objectives is done by modeling and supported by the library. It is not risk driven but based on modeling guidelines that exploit task and model features.

[2] It is envisaged that specific user groups develop their own proprietary libraries, for particular types of models. Clearly application development companies and organizational consultants will require different libraries.

2. CommonKADS library manager

   The CommonKADS library manager is responsible for the maintenance of a specific CommonKADS library. There may be such a manager for each type of library. He or she takes decisions on what models will be stored, how they are indexed. He or she keeps overall consistency of the models, avoids redundancy and assures the continued usability and evolution of the model specific library.

3. CommonKADS knowledge engineer

   The CommonKADS knowledge engineer builds models as they are specified in the CommonKADS model set in the course of knowledge based systems project. He or she uses the library as a resource for modelling. He retrieves models or parts of them. He or she combines, adapts and instantiates these models to suite the needs of his or her project. The knowledge engineer may just browse through the library for inspiration.

For each of these users more detailed guidelines can be provided to promote use and convergence of the libraries [Van de Velde, 1993b].

A fourth type of user is the workbench itself, more specifically modeling tools. For example the model advisors and configuration tools may directly interact with the libraries to query and retrieve models. The boundaries between these tools will need to be clarified when the library facility is put into place.

The CommonKADS workbench is a demonstration workbench and the application context is not an industrial one. However, it should sufficiently demonstrate that it can fulfill the needs of an industrial application context. Visibility, security and persistence become problems when the library contains proprietary models. It may be required that certain levels of model details are screened of. This is a problem in an industrial application context but not of our concern in the demonstration workbench. It should be possible to incorporate these as conservative extensions of the demonstration library.

2.4 Strategy and objectives

Our strategy is to consider needs that are common to all types of models [Van de Velde, 1992]. Expertise modeling, organizational modeling, task modeling, design modeling, agent modeling and communication modeling [deHoog et al., 1992] are all instances of modeling. We hypothesize that as far as the library is concerned these different modeling processes can be supported in a general way. This does not mean that we want basically one library. The different libraries should have a common way of working with them. This is why, we claim, they should be implemented on top of one generic library facility.

An extensive preliminary study of the expertise model library has been carried out in the course of the project. The requirements that follow from that study should be taken into account by the present specifications. Also the needs for the design model library, which is the other library one that will be constructed in the course of the project, should be taken into account. The design model library has been studied as well, be it to a lesser extend.

The following are the major objectives of the Generic Library Facility:

1. Support the definition of a model specific library tool for complete, partial and/or abstract models and their support in modeling.

   This objective targets the CommonKADS methodology specifier as a user (see Section 2.3).

2. Support the construction and maintenance of a model specific library of complete, partial and/or abstract models.

   This objective targets the CommonKADS library manager(s) as a user (see Section 2.3).

3. Support the effective usage of the model specific library for the construction of a project specific model.

   This objective targets the CommonKADS knowledge engineer(s) as a user (see Section 2.3). A scenario to have in mind is knowledge engineering practice.

4. Allow for the demonstration of library support in an integrated way using the CommonKADS workbench.

   This objective targets the KADS-II project team as a user. It enhances communication and coherence within the project, as well as the consolidation and demonstration of its results. A scenario to have in mind is library development work, exchange and demonstration.

These objectives are achieved by the functionalities described hereafter and the additional guidelines that specify the modalities of use of the generic library facility by the different types of users [Van de Velde, 1993b].

Subsequent sections describe the facility that we envisage. In the following section the functional and non-functional requirements are spelled out in detail.

2.5 Assumptions

It is assumed that the different ontologies (data models) for the CommonKADS models are available from the respective model documents. It is also assumed that the conceptual and formal languages for describing these models are available from other tasks in the project.

It is assumed that the contents of the library is available from other tasks. Decisions on what kind of models need to be stored and how exactly these should be indexed are taken for granted.

It is assumed that the library facility should enable library support of the kind as specified in the CommonKADS expertise library document. The basic library design is derived from the expertise model library requirements.

It is assumed that task features as they have been studied and defined in the project are used for purposes of indexing and annotation of the library elements.

It is assumed that the library facilities will be implemented in the CommonKADS workbench as it is being developed within the KADS-II project. Therefore the facility should serve the purposes of a demonstrator, not an industrial standards tool.

It is assumed that the different libraries will be realized as instances of a common generic library facility (this is, in principle, a design decision).

It is assumed that the generic library facility can be used for the creation of libraries that are specific for models that are not among the CommonKADS models. The generic library facility is supposed not to be restricted to CommonKADS models.

It is assumed that the library will be single-user (this assumption holds for the entire workbench).

2.6 Performance needs

If we take as level of performance the degree to which the needs are satisfied, represented by a specific set of values for the quality characteristics [ISO9001, 1987] then we can (loosely) identify the following minimal performance levels:

1. Functionality: the library facility should satisfy the functional requirements as specified in this document. A few advanced functionalities are indicated to be optional.

2. Reliability: the facility should be reliable under normal usage as a demonstration tool. However, total crash security should not be aimed at (in the sense that the system works fail-proof). In any case missing features should generate a meaningful message rather than an error.

3. Usability: it is important the library facility be easy to use and that a common look-and-feel is maintained over its different instances.

4. Efficiency: The CommonKADS libraries will be used during modelling in the CommonKADS workbench. This is not a time critical situations as long as response time is acceptable (in the order of a few seconds). However it can be expected that the library will be consulted several times in one modeling session and that this is often done on the editor's or modeling tool's initiative. The global overhead should he kept small.

5. Maintainability: The maintainability of the library tool is not a major issue, as long as it's specification and design are well documented. The tool itself is for demonstration only.

6. Portability: this in not an issue. The tool will be used on the demonstration workbench and its specification and documentation serves as the basis for porting, not the implementation itself.

3 General description of the Generic Library Facility

3.1 The elements of a library

The objects in a model specific library (i.e., a library for one of the types of models in the CommonKADS model set) are called *library elements*. There are three categories of library elements: the *modeling components*, the *generic models* and the *modeling operations*. These categories are disjoint.

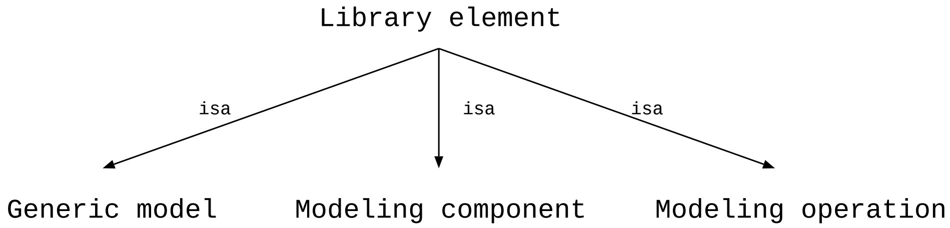

Figure 1: A model specific library contains instances of three disjoint categories of library elements.

- The *modeling components* are structures that are useful for the construction of complete models. They can be genuine building blocks that only need to be inserted in a model, or they can be like modeling methods that specify how to transform a model to yield for example more detail or more contents.[3]
- The *generic models* are frames that represent a class of complete models. They capture useful combinations of modeling components and serve as templates for the construction of complete models. A generic model is a model frame, complete in form but not necessarily in contents. Its structure is thus necessarily the same as the structure of the model type for which the library is specific.
- A *modeling operation* is an operation (relation) over models that transforms a generic model into another one. A modeling operation is typically represented as the application of a modeling operator in the context of a generic model, i.e., to a part of it. Modeling operators are parameterized with modeling components. They provide the necessary coherence between generic models and modeling components.

A model specific library is defined, basically, by specifying what objects in each of these three categories should look like. The specification of the format for each of these categories of library elements is called the *library ontology*. Obviously a library ontology is in part determined by the model definition of the type of model that the model specific library is specific for.[4] The actual content of a model specific library then corresponds to instances

[3] The term 'modeling component' is used instead of 'model component' to emphasize that it does not imply any restriction to syntactic parts of models.

[4] This is definitely so for the generic models, who's structure corresponds to the model's structure. For the modeling components this correspondence is, however, not imposed. It is the role of modeling operations to establish that correspondence.

in each of these categories (Figure 1). These instances are required to respect the library ontology, and the library tool should enforce this as much as possible.

Modeling components and generic models support the role of the library as a store of reusable complete, partial or abstract models or, more generally, stepping stones on modeling. Modeling operations support the role of the library in the process of modeling, i.e., as a store of model construction steps and episodes. They also specify the relation between modeling components and generic models (See also the D1.3 library design). Note, however, that operations are not in any sense active library elements: they do not “do” something. Operations are links between generic models that document that one model can be viewed as the result of some operation applied to another model and possibly exploiting whatever material there is in the modeling components that serve as parameter. Thus, an operation is a modeling step that has been taken and is reusable as a transformation of a model.[5]

The libraries do not support the risk-driven aspects of modelling. Rather it is the creation of a model of a certain type with stated characteristics that is supported. The decision on what that target state should be is risk-driven and taken outside of the library and for reasons that are independent of it. Ones the target state is recognized as a modeling objective the library can support the activity of getting there by providing useful modeling components and modeling operations.[6]

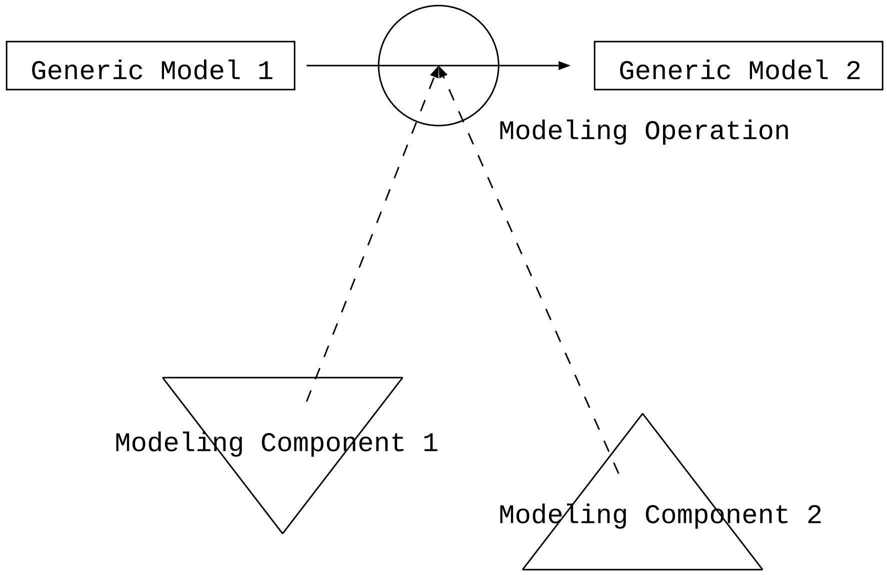

Figure 2: A basic structuring principle derives from the relation between generic models and modeling components through modeling operations.

There is no necessary correspondence between the structure of a modeling component and the structure of a generic model. In other words a modeling component is not necessarily

[5]It would be great to have automatic operations, and maybe simple ones are possible but this is not within the scope of these specifications.

[6]In terms of guidelines, the libraries implicitly contain modeling guidelines, not risk management guidelines.

a syntactic part of the generic model. It are the modeling operations that establish the connection between a modeling component and the part of the generic model in which the operation is applied (Figure 2). This 'trick' has the following advantages:

- it allows much more freedom in the definition of what are deemed to be useful modeling components,
- it allows a richer structure among models, not only based on specialization but on any kind of operation,
- it leaves room for a description format for modeling components that is different from the actual model structure.

3.2 The language of a library

Every library element can be described in either of four ways, or a combination of those. These four ways are the following (Figure 3):

```
Library element
---------------------------------
Verbal description
Conceptual description
Formal description
Feature description
Version (author, date, site)
```

Figure 3: The library elements share a common set of types of descriptions.

1. **Verbal description:** 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. The structure of this description is the same for all types of library elements.
2. **Conceptual description:** a description in a semi-formal language for capturing the entity-relationship structure of a particular library element and the slot-value structure of entities. These structures are defined by a model ontology and are specific for each type of library element.
3. **Formal description:** a description in a formal language. This is typically a refinement of the conceptual description, and we will assume that it preserves the structure of the conceptual description. The formal description language is specific for each type of library element.

4. **Feature description:** a description in terms of syntactic and semantic features, typically expressing compatibility constraints between modeling component and characteristics of the task reality that is being modeled. A feature can, for instance, indicate component necessity or component exclusion. Features restrict the set of potential combinations of components to those which are structurally possible. Feature description languages can be specific for the type of modeling component (see the 1.5 work for a feature description framework).

Of these four descriptions the conceptual description is central because its structure also determines the way one refers to parts of the model (pointing at a part of a model is called referencing. See also Section 5.5). The other descriptions are optional but verbal descriptions are highly recommended for their being easy to understand, while feature descriptions may be heavily exploited in the query mechanism of a library (see also Section 5.7).

3.3 The structure of a library

A model specific library supports the reuse of previous modeling experience. Actual reuse entails two activities: selection of useful modeling components and generic models, and progression through useful steps in modeling. Effective support is being achieved by a number of structuring principles within the library. The basic structure derives from the structure of the models themselves and from the relation between generic models and modeling components through modeling operations, as explained previously. This basic structure is augmented with a hierarchical (taxonomical) organization of modeling components, and a non-hierarchical index structure based on the features contained in the feature description of the library elements.

- **Taxonomies of modeling components:**
  modeling components within a single class (as defined by the model ontology) are organized in one or a series of taxonomies. taxonomies facilitate the construction of a model from a generic model by indicating the refinements and alternatives that the library provides. Taxonomies can be based on various principles, for example on ontological choices for describing an application domain.

  To navigate through the library based on the taxonomical structure corresponds to exploring more and more specific versions of a modeling component. For example one can explore varieties of causal models, or varieties of control procedures, or varieties of representations (Figure 4)

- **Classes from component features:**
  component features document compatibility constraints between modeling components or characteristics of a problem situation that can be modelled in a certain way. The stated or implied presence of a feature restricts the library elements from being retrieved. It thus corresponds to an implicit class organization of library elements, where a class corresponds to those elements that have a compatible set of features.

  To navigate through the library based on the class structure derived from features corresponds to exploring the set of modeling decisions compatible with a given set of features. These features can be explicitly stated as part of a query (e.g., one wants an exhaustive procedure for generating solutions) or can be derived from previous modeling decisions (e.g. choosing a causal model requires a causal reasoner).

- **Structure among generic models from modeling operators:**
  A basic structuring mechanism derives from the way in which generic models are related to eachother by means of modeling operations parameterized with modeling components. For 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 (Figure 5).

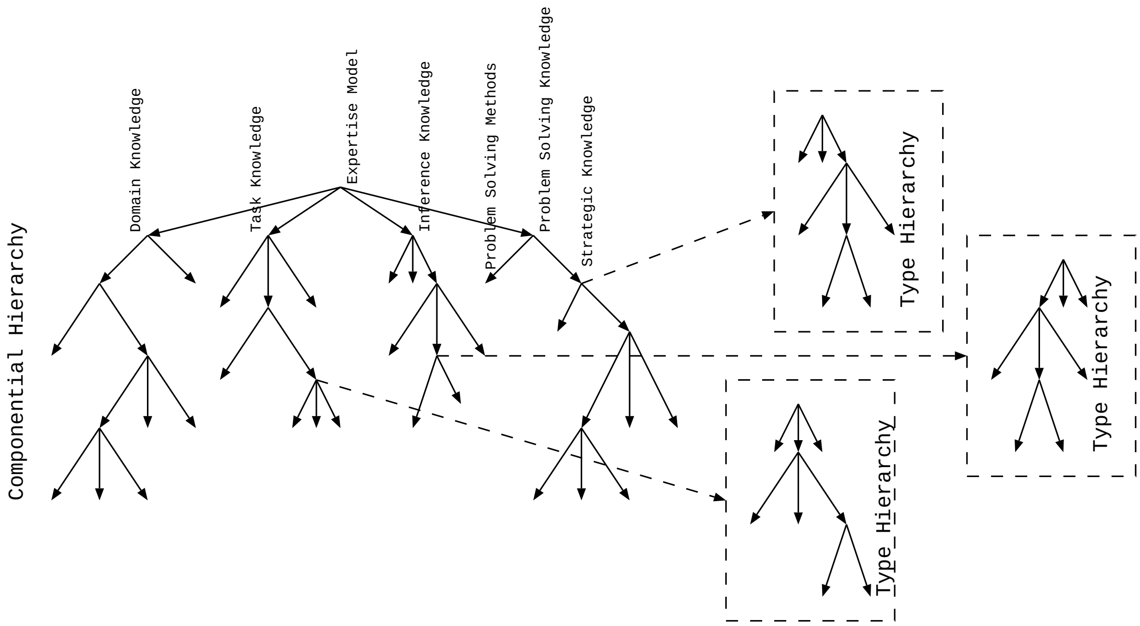

Figure 4: A model is a large componential structure (part-of) of knowledge items from predefined categories. Modeling components contributing to the construction of these parts are organized in type-hierarchies. The process of modelling can be viewed as filling the model frame by pushing down the category representatives along the branches of their typologies until they are sufficiently specific. (figure taken from D1.3).

3.4 From generic library facility to a model specific library

In order to define a model specific library one has to specify the ontology to be used in that library. This amounts to specifying what is the model that the library will be specific for, to specifying what kind of modeling components are deemed to be relevant and useful, and to specifying what are the operations that are used over the models.

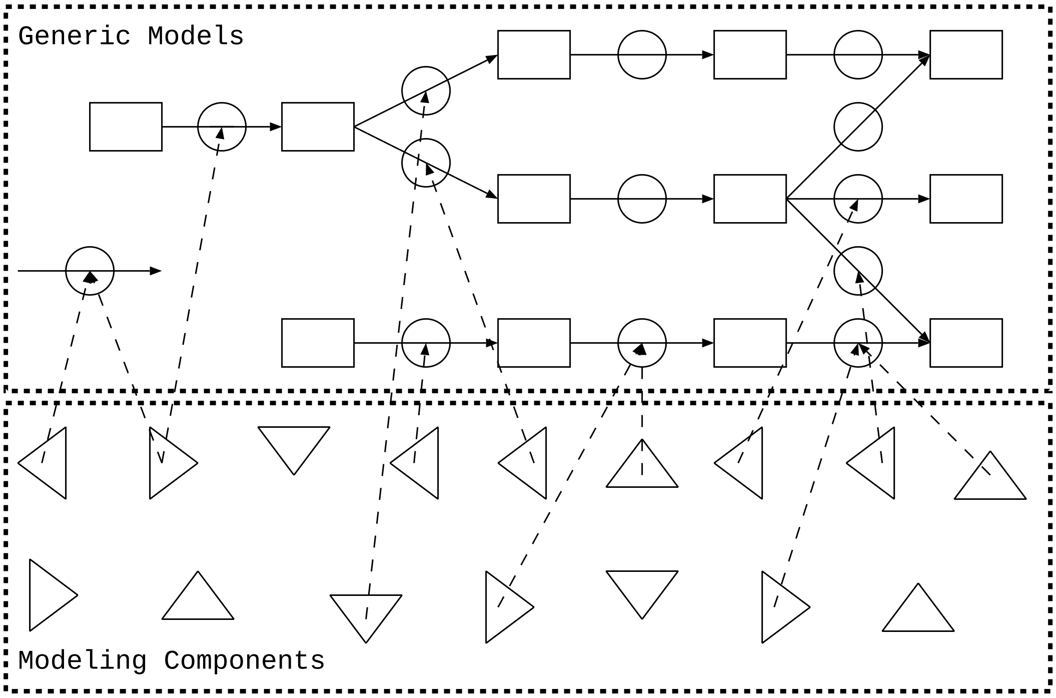

Figure 5: The library can be thought of as a store of generic models and modeling components, or as a store of modeling episodes, sequences of modeling operations.

The definition of the model ontology for a model specific library is closely related but not necessarily identical to the model definition (entity-slot-relationship) of the model, usually defined in the corresponding model definition document. This definition is what we will call the model template.

The model template serves two immediate purposes. The first purpose is to provide the structure of the models, and therefore also the structure of the generic models. Indeed, generic models are models that are complete in form but not necessarily in context. The second purpose of the model template is to serve as the main structure for navigation within a model, what we will call 'reference'. So, essentially, the model template leads to the definition of the structure of the generic models.

Apart from the model template one defines the structure of modeling components. This is done also in a entity-slot-relationship fashion. Again, note that the structure of the modeling components does not need to be a sub-structure of the model template (but it can be, and often will). If it is not then notions of granularity and orthogonality do not make sense and it is not required that modeling components are in any sense the smallest or orthogonal (canonical) elements in the library.

Finally one defines the modeling operations. These are described, like any of the other library elements, in either of the four languages. As explained before and emphasized again the modeling operations are not in any way automatically executed, so there is no need for executable descriptions or specifications of them.

3.5 Terminology

In what follows we use the following terminology:

Generic Library Facility: the generic library tool that will be implemented as a workbench kernel facility to allow the generation and management of the model specific libraries.

Model Specific Library: a library tool for a particular type of CommonKADS model. The tool is generated from the generic library facility.

Library element: any of the objects contained in a model specific library.

Version: for every library or library object, a version is the triple of author, date and where attributes. These attributes are available for every library or library object and are provided with default values when not explicitly given. The defaults are user-name for the author attribute, now for the date attribute and user-site for the where attribute.

4 Data Model

Figure 6 shows the `is_a` hierarchy of the library related objects defined hereafter. Figure 7 shows the `consists_of` relation between the library related objects defined hereafter.

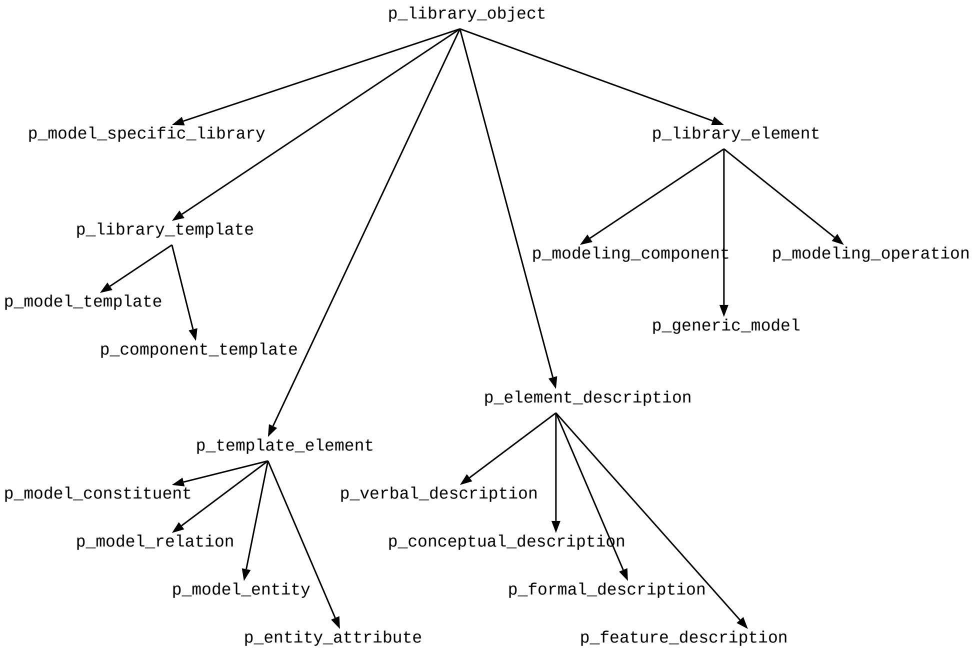

Figure 6: The `is_a` hierarchy of library objects.

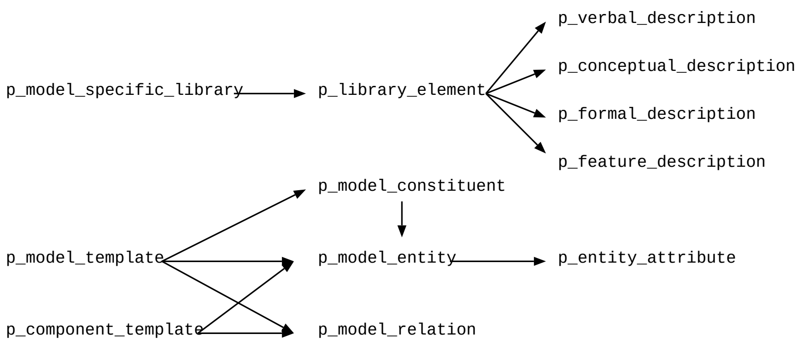

Figure 7: The `consist_of` relation among library objects.

4.1 General

For defining a model specific library we need to distinguish carefully between the library specification and the library contents. A model specific library (p_model_specific_library) is specified in terms of a model template (p_model_template) and a set of component templates (p_component_template). A model template specifies the complete structure of generic models (p_generic_model) while a component template specifies the structure of a modeling_component (p_modeling_component). Model and component templates are similar and we use the term library template to refer to both of them (common type p_library_template). The structure of a model template is illustrated in Figure 8. A library template groups together a set of entities, attributes and relations (p_template_element). It specifies the data model for the model specific library. A library template structures a set of entities (p_model_entity) in a hierarchical way, so that entities can be referred to by a list of symbols and the name. Together these lists form the road-map of the template (road-map slot of p_library_template). Entities can have attributes (p_model_attribute) and can be related by relations (p_model_relation). In a model template the entities can be grouped in constituents (p_model_constituent). Every template element has a conceptual syntax (conceptual-syntax slot of p_template_element), a formal syntax (formal-syntax slot of p_template_element), and a feature syntax (feature-syntax slot of p_template_element) specified. These syntaxes are used for the corresponding description of the corresponding parts of library elements (p_library_element).

The library elements (p_library_element) are the actual contents of a model specific library (p_model_specific_library). Their structure is depicted in Figure 9. All library elements must respect one of the library templates. A generic model (p_generic_model) is a set of descriptions (p_element_description) in one of the four languages (verbal in p_verbal_description, conceptual in p_conceptual_description, formal in p_formal_description and feature in p_feature_description). The descriptions have a reference that locates the described entity according to the road-map of the library template. The road map of the library element (road-map slot of p_library_element) is a pruned version of the road-map for the library template and indicates what parts of the template are actually filled in with at least one description.

Subsequent subsections describe the data model in detail.

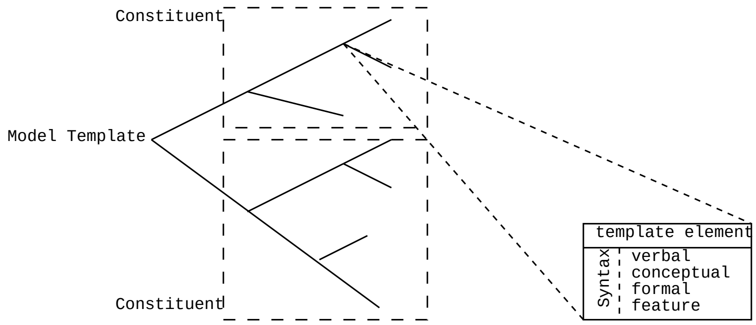

Figure 8: The structure of a model template. A component template is similar but has no constituent structure.

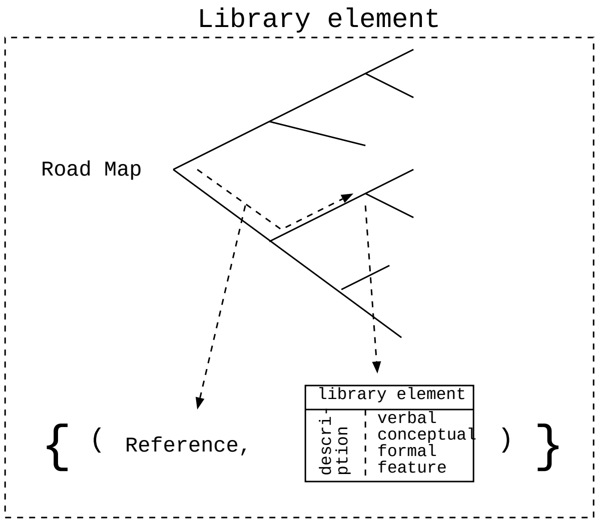

Figure 9: The recursive structure of library elements. The top element collects the entire set of referenced elements and the global roadmap.

4.2 Library object

The library object generalizes all objects in the data model of the generic library facility and the model specific libraries. All these objects have a name and version consisting of author, date and site information.

p_library_object

```
CLASS:          p_library_object
DESCRIPTION:    any object used by the generic library facility or
                a model specific library
SLOTS:
  Name:         Name
  Value type:   symbol
```

```
  Description: the name of the model specific library

  Name:        Author
  Value type:  string
  Description: the author of the model specific library

  Name:        Date
  Value type:  date
  Description: the date and time of creation

  Name:        Site
  Value type:  string
  Description: the origin of the model specific library
```

4.3 Model specific library

p_model_specific_library

```
CLASS:          p_model_specific_library
SUB-CLASS OF:   p_library_object
DESCRIPTION:    a model specific library
SLOTS:
RELATIONS:
  Name:         r_has_model_template
  Type:         p_model_template
  Description: the model template for the model specific library
  Cardinality: 1,1

  Name:         r_has_component_template
  Type:         p_component_template
  Description: the template for a modeling component model of
               the model specific library
  Cardinality: 1,n

  Name:         r_has_operator_template
  Type:         p_operator_template
  Description: the template for a modeling operator of
               the model specific library
  Cardinality: 1,n

  Name:         r_consists_of
  Type:         p_library_element
  Description: an element of the library
  Cardinality: 1,n
```

4.4 Model template

p_library_template

```
CLASS:          p_library_template
SUB-CLASS OF:   p_library_object
DESCRIPTION:    template for library elements
SLOTS:
RELATIONS:
  Name:         r_consists_of
  Type:         p_model_entity
  Description:  an entity in the model template
  Cardinality:  1,n

  Name:         r_consists_of
  Type:         p_model_relation
  Description:  a relation in the model template
  Cardinality:  1,n
```

p_model_template

```
CLASS:          p_model_template
SUB-CLASS OF:   p_library_template
DESCRIPTION:    a model template
SLOTS:
  Name:         road-map
  Value type:   model road map
  Description:  the road map for access to parts of library
                elements that have this template.
RELATIONS:
  Name:         r_consists_of
  Type:         p_model_constituent
  Description:  a constituent of the model template
  Cardinality:  1,n
```

p_component_template

```
CLASS:          p_component_template
SUB-CLASS OF:   p_library_template
DESCRIPTION:    a component template
SLOTS:
  Name:         road-map
  Value type:   component road map
  Description:  the road map for access to parts of library
                elements that have this template.
```

RELATIONS:

4.5 Road map

A road map provides a skeletal hierarchical structure for a model or modeling component. It uses the names of constituents, entities, relations and attributes as terms, as well as references to library elements in other libraries.

There are two types of road maps. Model road maps have a constituent structure. Component road maps do not have a constituent structure. Their syntax is as follows:

```
model-road-map ::= (model-name constituent-structure*)

component-road-map ::= (component-name entity-structure*)

constituent-structure ::= (constituent-name entity-structure*)

entity-description ::= entity-structure
                       | (-> entity-reference)
                       | (-| entity-reference)

entity-reference ::= symbol+

entity-structure ::= (entity-name (slot-structure*)
                       relation-structure*)

slot-structure ::= slot-name
model-name ::= symbol
entity-name ::= symbol
relation-name ::= symbol
slot-name ::= symbol

relation-structure ::= (relation-name entity-description)
```

4.6 Template element

p_template_element

```
CLASS:          p_template_element
SUB-CLASS OF:   p_library_object
DESCRIPTION:    an element in a template (constituent,
                entity, relation or attribute)
SLOTS:
  Name:         conceptual-syntax
  Value type:   bnf-rules
  Description:  the syntax to be used for this element
```

```
               in the conceptual description

  Name:        formal-syntax
  Value type:  bnf-rules
  Description: the syntax to be used for this element
               in the formal description

  Name:        feature-syntax
  Value type:  bnf-rules
  Description: the syntax to be used for this element
               in the feature description
RELATIONS:
```

p_model_constituent

```
CLASS:         p_model_constituent
SUB-CLASS OF:  p_template_element
DESCRIPTION:   a constituent of a model template
RELATIONS:
  Name:        r_consists_of
  Type:        p_model_entity
  Description: an entity in this constituent
  Cardinality: 1,n
```

p_model_entity

```
CLASS:         p_model_entity
SUB-CLASS OF:  p_template_element
DESCRIPTION:   an entity in a model template
SLOTS:
RELATIONS:
  Name:        r_consists_of
  Type:        p_entity_attribute
  Description: an attribute of the entity
  Cardinality: 1,n
```

p_entity_attribute

```
CLASS:         p_entity_attribute
SUB-CLASS OF:  p_template_element
DESCRIPTION:   an attribute of an entity
SLOTS:
RELATIONS:
  Name:        r_attribute_owner
  Type:        p_model_entity
```

```
  Description: the entity that owns the attribute
  Cardinality: 1,1
```

p_model_relation

```
CLASS:         p_model_relation
SUB-CLASS OF:  p_template_element
DESCRIPTION:   a relation in a model template
SLOTS:
RELATIONS:
  Name:        r_source-entity
  Type:        p_model_entity
  Description: the source of the relation
  Cardinality: 1,1

  Name:        r_target-entity
  Type:        p_model_entity
  Description: the target of the relation
  Cardinality: 1,1
```

4.7 Syntax

Verbal description

There is no specific bnf for verbal descriptions. Anything is allowed here that looks like text.

Conceptual description

There is no default syntax for conceptual descriptions. Instead the full syntax from the corresponding model definition document needs to be specified.

There is always the option not to give a conceptual syntax for a library object. In that case, when the library object is used in a generic model or model template, a conceptual description can be created (on demand) which is based on the road map with those subnodes in the map that do have a dedicated conceptual syntax and description replaced by that conceptual description. In this way one can for instance suffice to give a conceptual syntax for the attribute values.

Formal description

There is no default syntax for formal descriptions. Instead the full syntax from the corresponding model definition document needs to be specified.

There is always the option not to give a formal syntax for a library object. In that case, when the library object is used in a generic model or model template, a formal description can be created (on demand) which is based on the road map with those sub-nodes in the map that do have a dedicated conceptual syntax and description replaced by that formal description. In this way one can for instance suffice to give a formal syntax for the attribute values.

Feature description

There is a default syntax for feature descriptions.

```
feature-description ::=
  DOMAIN: single-feature-description+
  PURPOSE: single-feature-description+
  ENVIRONMENT: single-feature-description+
  GROUNDING: single-feature-description+
  EPISTEMOLOGICAL: single-feature-description+
  PRAGMATIC: single-feature-description+
  COMPUTATIONAL: single-feature-description+
  LIBRARY: single-feature-description+

single-feature-description ::= ( feature-name = feature-value )
                               | attribute

attribute ::= symbol | text

feature-name ::= symbol

feature-value ::= symbol | text
```

Feature descriptions can be matched; if the value is text then the feature does not play a role in the matching. Features can be switched on or of for matching depending on their category.

4.8 Library element

p_library_element

```
CLASS:         p_library_element
SUB-CLASS OF:  p_library_object
DESCRIPTION:   a library element
SLOTS:
  Name:        reference
  Value type:  symbol or list of symbols
  Description: the reference for this library element.
               It is the name if the element is a generic model,
               modeling component or modeling operator. Otherwise
```

```
               it is the reference to the element in the scope
               of the encompassing generic model, modeling component
               or operation to which this element is related by the
               r_part_element relation.
RELATIONS:
  Name:        r_consists_of
  Type:        p_verbal_description
  Description: the verbal description of the library element
  Cardinality: 1,1

  Name:        r_consists_of
  Type:        p_conceptual_description
  Description: the conceptual description of the library element
  Cardinality: 1,1

  Name:        r_consists_of
  Type:        p_formal_description
  Description: the formal description of the library element
  Cardinality: 1,1

  Name:        r_consists_of
  Type:        p_feature_description
  Description: the feature description of the library element
  Cardinality: 1,1
```

4.9 Generic model

p_generic_model

```
CLASS:         p_generic_model
SUB_CLASS_OF:  p_library_element
DESCRIPTION:   a generic model
SLOTS:
  Name:        road-map
  Value type:  model-road-map
  Description: the internal reference structure of the element
RELATIONS:
  Name:        r_specialization_of
  Type:        p_generic_model
  Description: a more general instance on which the definition of this
               one is based. All slots and relations are inherited.
  Cardinality: n,1

  Name:        r_has_model_template
  Type:        p_model_template
```

```
  Description: the model template for the generic model
  Cardinality: 1,1

  Name:        r_element_part
  Type:        p_library_element
  Description: the library elements describing a part of the model.
               The references of these parts are compatible with the
               road-map of this library element.
  Cardinality: 1,n
```

The model template of a generic model is the same as the one for the model specific library for which it is defined.

4.10 Modeling component

p_modeling_component

```
CLASS:         p_modeling_component
SUB_CLASS_OF:  p_library_element
DESCRIPTION:   a modeling component
SLOTS:
  Name:        road-map
  Value type:  nested list of symbols
  Description: the internal reference structure of the element
RELATIONS:
  Name:        r_specialization_of
  Type:        p_generic_model
  Description: a more general instance on which the definition of this
               one is based. All slots and relations are inherited.
  Cardinality: n,1

  Name:        r_has_component_template
  Type:        p_component_template
  Description: the component template for the modeling component
  Cardinality: 1,1

  Name:        r_element_part
  Type:        p_library_element
  Description: the library elements describing a part of the model.
               The references of these parts are compatible with the
               road-map of this library element.
  Cardinality: 1,n
```

4.11 Modeling operation

Modeling operations are essentially relations between generic models. They are described as applications of a modeling operator, parameterized by modeling components. The various description slots of the operation library elements allow for ample flexibility in the definition of the operators.

p_modeling_operation

```
CLASS:         p_modeling_operation
SUB_CLASS_OF:  p_library_element
DESCRIPTION:   a modeling operation
SLOTS:
  Name:        context
  Value Type:  list of symbols
  Description: the part of the generic model on which this
               operation works

  Name:        road-map
  Value type:  nested list of symbols
  Description: the internal reference structure of the operator.
               Specifies the names and types (in terms of name of a
               component template) of parameters to the operator.
RELATIONS:
  Name:        r_specialization_of
  Type:        p_modeling_operation
  Description: a more general instance on which the definition of this
               one is based. All slots and relations are inherited
               unless overridden.
  Cardinality: n,1

  Name:        r_model_template
  Type:        p_model_template
  Description: the model template that should be respected by the
               generic models that this operation relates. The value
               of the context slot must be meaningful with respect to
               this template.

  Name:        r_from_model
  Type:        p_generic_model
  Description: the source generic model or generic models that
               provide the context of the operator
  Cardinality: 1,n

  Name:        r_to_model
  Type:        p_generic_model
```

```
  Description: the target generic model or generic models
  Cardinality: 1,n

  Name:        r_parameterized_by
  Type:        p_modeling_component
  Description: the modeling components that are parameters
               to the operator
  Cardinality: 1,n
```

4.12 Descriptions

p_element_description

```
CLASS:         p_element_description
SUB-CLASS OF:  p_library_object
DESCRIPTION:   an element description
SLOTS:
  Name:        Value
  Value type:  according to syntax slot
  Description: the actual description
```

p_verbal_description

```
CLASS:         p_verbal_description
SUB_CLASS_OF:  p_element_description
DESCRIPTION:   a verbal description
SLOTS:
RELATIONS:
  Name:        r_describes_element
  Type:        p_library_element
  Description: the library element being described
  Cardinality: 1,n
```

p_conceptual_description

```
CLASS:         p_conceptual_description
SUB_CLASS_OF:  p_element_description
DESCRIPTION:   a conceptual description
SLOTS:
RELATIONS:
  Name:        r_describes_element
  Type:        p_library_element
  Description: the library element being described
  Cardinality: 1,n
```

p_formal_description

```
CLASS:         p_formal_description
SUB_CLASS_OF:  p_element_description
DESCRIPTION:   a formal description
SLOTS:
RELATIONS:
  Name:        r_describes_element
  Type:        p_library_element
  Description: the library element being described
  Cardinality: 1,n
```

p_feature_description

```
CLASS:         p_feature_description
SUB_CLASS_OF:  p_element_description
DESCRIPTION:   a feature description
SLOTS:
RELATIONS:
  Name:        r_describes_element
  Type:        p_library_element
  Description: the library element being described
  Cardinality: 1,n
```

4.13 Example

Below are a few examples that illustrate the intention of the above data model. At some points the examples are not complete. Descriptions are treated as slots, rather than as relations.

4.13.1 Library template

Figure 10 shows the design model as it is defined in the CommonKADS model set. A simple model specific library could correspond to the following instance (currently empty):

```
Instance of p_model_specific_library
  Name: my-design-library
  Author: Walter Van de Velde
  Date: Tue Oct 19 12:04:32 1993
  Site: Arti17, VUB AI-Lab, Pleinlaan 2, Brussels
  r_has_model_template: design-model-template
  r_has_component_template: computational-architecture
```

The author, date and site slots exist for all instances of p_library_object. They are not repeated in the descriptions below. The above library allows for the storage of generic

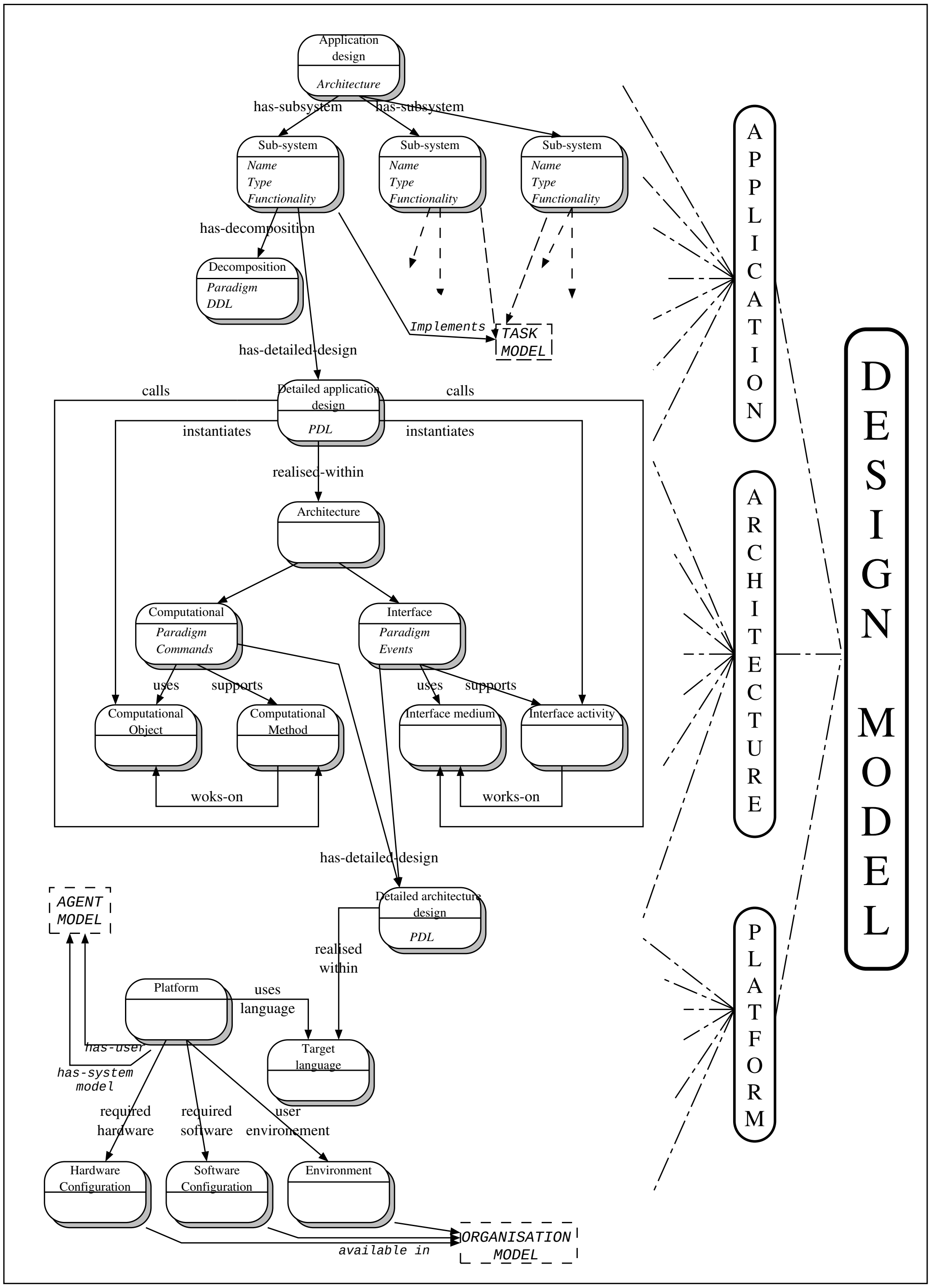

Figure 10: The CommonKADS design model.

models that respect the design-model-template and of modeling components that respect the component template computational-architecture. Of course a realistic library should have more categories of components but for purpose of illustration this one will do.

Below is the instance of p_model_template. The consists-of relations group the constituents, entities and relations. After the name of the target instance, between brackets is the class that the target element is an instance of.

```
Instance of p_model_template
  Name: design-model-template
  Road-map:
    (design-model                     ;; model name
       (application                   ;; constituent
          (application-design (application-architecture)
                                      ;; entity and slots
            (has-subsystem            ;; relation
              (subsystem (name type functionality)
                                      ;; entity and slots
                (has-decomposition    ;; relation
                  (decomposition (paradigm DDL)))
                (has-detailed-design
                  (detailed-application-design (PDL)
                    (calls (-> computational-method)
                                      ;; internal pointer
                           (-> interface-activity))
                    (instantiates (-> computational-object)
                                  (-> interface-medium))
                    (realized-within (-> architecture))))
                (implements
                  (-| task-model))))) ;; external pointer
        (architecture
          (architecture-design
            (consists-of
              (computational-architecture (paradigm commands)
                (uses
                  (computational-object))
                (supports
                  (computational-method ()
                     (works-on (computational-object))))
                (has-detailed-design
                  (detailed-architecture-design (PDL)
                    (realized-within (-> target-language))))
              (interface-architecture (paradigm events)
                (uses
                  (interface-medium))
                (supports
                  (interface-activity ()
```

```
                    (works-on (interface-medium))))
               (has-detailed-design
                 (-> detailed-architecture-design)))))
       (platform
         (platform-design
           (uses-language
             (target-language ()))
           (user-environment
             (environment ()
               (available-in (-| organization-model))))
           (required-software
             (software-configuration ()))
               (available-in (-| organization-model))))
           (required-hardware
             (hardware-configuration ()))
               (available-in (-| organization-model))))
           (has-user
             (-| agent-model))
           (has-system-model
             (-| agent-model))))
 r-consists-of: application-design (p_model_constituent)
                architecture-design (p_model_constituent)
                platform-design (p_model_constituent)
 r-consists-of: application (p_model_entity)
                sub-system (p_model_entity)
                decomposition (p_model_entity)
                detailed application design (p_model_entity)
                architecture specification (p_model_entity)
                computational architecture specification
                (p_model_entity)
                computational object (p_model_entity)
                computational method (p_model_entity)
                interaction architecture specification
                (p_model_entity)
                interface medium (p_model_entity)
                interface activity (p_model_entity)
                detailed architecture design (p_model_entity)
                platform (p_model_entity)
                target language (p_model_entity)
                user (p_model_entity)
                hardware configuration (p_model_entity)
                software configuration (p_model_entity)
                environment (p_model_entity)
 r_consists_of: has-subsystem (p_model_relation)
                implements (p_model_relation)
                has decomposition (p_model_relation)
```

```
                 has detailed design (p_model_relation)
                 realized within (p_model_relation)
                 instantiates (p_model_relation)
                 calls (p_model_relation)
                 uses (p_model_relation)
                 supports (p_model_relation)
                 has detailed design (p_model_relation)
                 works on (p_model_relation)
                 uses (p_model_relation)
                 supports (p_model_relation)
                 has detailed design (p_model_relation)
                 works on (p_model_relation)
                 realized within (p_model_relation)
                 uses language (p_model_relation)
                 required hardware (p_model_relation)
                 required software (p_model_relation)
                 has user (p_model_relation)
                 has system model (p_model_relation)
                 available in (p_model_relation)
                 available in (p_model_relation)
                 available in (p_model_relation)
```

The entities, attributes and relations are themselves defined as objects. For example:

```
Instance of p_model_constituent
  Name: platform-design
  r-consists-of: platform (p_model_entity)
                 target language (p_model_entity)
                 user (p_model_entity)
                 hardware configuration (p_model_entity)
                 software configuration (p_model_entity)
                 environment (p_model_entity)

Instance of p_model_entity
  Name: computational-architecture-specification
  formal-syntax: Z-syntax
  r_consists_of: paradigm (p_model_attribute)
                 command (p_model_attribute)

Instance of p_model_attribute
  Name: paradigm
  r_attribute_owner: computational-architecture-specification

Instance of p_model_relation
  Name: uses language
  r_source_entity: platform
```

```
  r_target_entity: target-language
```

The model template describes the structure of the generic models. Similarly the template for a modeling component called computational architecture is defined. Remember that the component templates don't have a constituent structure.

```
Instance of p_component_template
  Name: computational_architecture
  Road-map:
    (computational-architecture  ;; component-name
      (computational-architecture (paradigm commands)
        (uses
          (computational-object))
        (supports
          (computational-method ()
             (works-on (computational-object))))
        (has-detailed-design
          (detailed-architecture-design (PDL)
            (realized-within (-> target-language)))))
  r_consists_of: computational-architecture
                 (p_model_entity)
                 computational-object (p_model_entity)
                 computational-method (p_model_entity)
                 detailed-architecture-design (p_model_entity)
  r_consists_of: realized-within (p_model_relation)
                 instantiates (p_model_relation)
                 uses (p_model_relation)
                 supports (p_model_relation)
                 has detailed design (p_model_relation)
```

4.13.2 Modeling Component and Library Element

Here is an example of a modeling component of the category computational-architecture:

```
Instance of p_modeling_component
  Name: standard-blackboard
  Reference: standard-blackboard
  r_has_component_template: computational-architecture
  Road-map:
    (computational-architecture
      (computational-architecture (paradigm commands)
        (uses
          (computational-object))
        (supports
          (computational-method ()
```

```
            (works-on (computational-object))))))
  Verbal-description: "the standard blackboard architecture as
    described for instance in Velthuysen:92."
  Formal-description: (the Z specification as in Velthuysen:92)
  r_element_part: standard-blackboard-object
  r_element_part: standard-knowledge-source
  r_element_part: standard-blackboard-cycle
```

Note that the road map is a substructure of the road-map of the template. It shows that the part-elements of the standard-blackboard library element do not describe the detailed-architecture-design. The standard blackboard architecture as a whole has no conceptual description but there is a formal specification in Z available. To find more detail one can follow the road-map to find, for example, a computational object called standard-knowledge-source among the parts:

```
Instance of p_library_element
  Name: standard-knowledge-source
  Reference:
    (computational-architecture uses computational-object)
  Road-map:
    (computational-object
      (has-condition (condition))
      (has-action (action))
      (has-area (blackboard-area)))
  Verbal-description: "a knowledge source has a condition and action
    part. The condition applies to a designated part of a blackboard and
    returns true or false without side affects.  The action part acts on
    a designated part of a blackboard and produces a side effect."
  r_element_part: ...
```

Note that this is a library element, and not a modeling component. As a consequence it is only accessible from a modeling component and itself not usable as an argument in modeling operations. If one wants this then knowledge source must have its own component template declared.

It is unlikely that this element has a formal description, since the knowledge source is "empty". Specializations of this one could have a specific condition and action filled in which would mean that there is some description available in the library elements with reference `(computational-object has-condition condition)` among those related as `r_element_part` to the more specific knowledge source.

4.13.3 Modeling Operator

Design Modeling We now turn to the definition of the most simple modeling operator. It simply inserts a modeling component in the right location of a generic model. This is only possible if the model and component templates are compatible, of course.

```
Instance of p_modeling_operator
  Name: select-computational-architecture
  Context: (design-model
             architecture-design
             consists-of
             computational-architecture)
  r_model_template: design-model-template
  r_parameterized_by: computational_architecture
  Verbal-description: select a computational architecture within
    which to realize the application specification
```

This operator is easy enough to define and makes the necessary link between a category of modeling components (in this case computational-architecture) and the generic models for the design model. On its own this operator does not relate explicit generic models. It documents that, whenever one wants to work on the element `(design-model architecture-design consists-of computational-architecture)` of a design model, then this may be a useful operation to do. More specific guidance is documented in the following instance:

```
Instance of p_modeling_operation
  Name: blackboard-selection-#12
  Specialization-of: select-computational-architecture
  r_from_model: diagnosis_design
  r_parameterized_by: standard-blackboard
  Feature-description:
    epistemological: multiple sources of evidence at different levels of
      abstraction need to be combined.
```

This modeling_operation recommends a blackboard for an existing diagnosis_design in the library. It mentions no explicit r_to_model, so all it says is that it may be useful to select that computational architecture. However, it does not actually give the new generic model where the blackboard is inserted. In this case, a tool may do the job for the user. An alternative that the library provides is to give an explicit r_to_model. If it is then chosen to apply the operation then the refined generic model can be retrieved from the library.

Expertise Modeling The CommonKADS expertise model has the notions of task and inference. The expertise model library has the notion of function. Functions are described as follows:

> "Functions are components that we still do not know whether they will be decomposed or not. If they are, they are transformed into task definitions; if not, into inferences. For this reason, a function will contain a set of slots which is the union of the sets of slots of the task definition and inferences. In this way, the transformation operation will consists basically of suppressing some of the slots of the function."

This is an example of the notion of modeling operator as the library supports it. "Interpret as task" takes a generic model and interprets a library component (function) as a task in that generic model. Similarly, the operator "Interpret as inference" takes a generic model and interprets a library component (function) as an inference in that generic model.

```
Instance of p_modeling_operation
  Name: interpret as task
  context-reference: (expertise-model task-knowledge task)
  r_parameterized_by: function
  verbal description: "removes the operation-type and
     static-role slot from the function and insert it at
     the place of the task"
```

As before for the design model, this operation can be specialized in various ways to give the appropriate level of support to the knowledge engineer.

5 Functional specification

5.1 Generic Library Facility Architecture

The generic library facility has the following sub-systems: an ontology editor and a library manager. These allow for the creation of a model specific library.

Ontology Editor: creates a library ontology by defining the structure and syntax of the generic models, the modeling components and the modeling operations. In particular it supports the creation of the library template and the component templates (Section 5.2).

Library Manager: supports the creation of a model specific library from a library ontology. Manages the coexistence of several model specific libraries and their combination in a single one (Section 5.3).

A model specific library, once created, is manipulated by the following subsystems: a library browser, a reference sub-system, an element inspector and a query handler.

Library Browser: general tool for walking through a model specific library based on names alone. General tool for inserting and removing elements in a model specific library and, more generally, for managing the library contents (Section 5.4).

Reference sub-system: retrieves elements or parts of them from the library based on reference in the form of a path (Section 5.5).

Element inspector: The element inspector allows one to look in detail at the shape and contents of a library element. The navigation within a library element is based on the road-map of the template and on the different descriptions that the library element may have (Section 5.6).

Query Handler: general tool for accessing a model specific library based on queries for partially specified library elements (Section 5.7).

Subsequent subsections give a more precise description of the functionalities that are required for each of these sub-systems.

5.2 Ontology Editor

5.2.1 General

The Ontology Editor creates a library ontology. The library ontology is the specification of what type of objects there are in each of the three categories of library elements (generic model, modeling component and modeling operation). This is done by defining the structure and syntax in each of the different description languages.

5.2.2 Data Model

The data model of the ontology editor is the p_library_template and p_template_element (see Section 4).

5.2.3 Functional Requirements

Define model template (*functionality of Ontology Editor*):
define the model template (p_model_template) for the model specific library.

Define component template (*functionality of Ontology Editor*):
define a component template (p_component_template) for the model specific library.

Define operator type (*functionality of Ontology Editor*):
defines an operator type for the model specific library.

Below these primary functionalities are further decomposed.

Define library template (*functionality of Ontology Editor*):
define a library template which is either a model template (p_model_template) or a component template (p_component_template).

The following are the sub-functionalities required for defining the templates:

Create model template (*functionality of Ontology Editor*):
creates a model template (p_model_template) with a given name and version. Name must be specified. Version is automatically created and can not be modified. Road-map is a list with as only element the name of the template. No further relations are established.

Create component template (*functionality of Ontology Editor*):
creates a model template (p_component_template) with a given name and version. Name must be specified. Version is automatically created and can not be modified. Road-map is a list with as only element the name of the template. No further relations are established.

Copy library template (*functionality of Ontology Editor*):
creates a new named library template as a copy of a named one, and copies of its related objects. Version is the only distinction between them that matters.

Create model constituent (*functionality of Ontology Editor*):
creates a model constituent (p_model_constituent) with a given name and version. Name must be specified. Version is automatically created and can not be modified. No further slots or relations are established.

Create model entity (*functionality of Ontology Editor*):
creates a model entity (p_model_entity) with a given name and version. Name must be specified. Version is automatically created and can not be modified. No further slots or relations are established.

Create model relation (*functionality of Ontology Editor*):
creates a model relation (p_model_relation) with a given name and version. Name must be specified. Version is automatically created and can not be modified. No further slots or relations are established.

Create entity attribute (*functionality of Ontology Editor*):
creates a entity attribute (p_entity_attribute) with a given name and version. Name must be specified. Version is automatically created and can not be modified. No further slots or relations are established.

Specify template structure (*functionality of Ontology Editor*):
takes a template, a reference path and a named template element. A reference path is a list of names of template elements that were created before. Updates the road map of the template. If the reference path is incompatible with the current road-map then a warning is issued and the template remains unchanged. Otherwise also the template element is added to the r_consists_of relation with the template. All necessary entities are also added to the appropriate constituent, attributes to the appropriate entities.

Check Library Template (*functionality of Ontology Editor*):
checks whether a named library template is syntactically complete. This means that the created objects are effectively used in the template, that they are of the right type, and entities belong to at most one constituent. Missing elements or errors are reported in text format on a log_file with name “¡library-template-name¿.liblog”

Specify Language (*functionality of Ontology Editor*):
specifies the syntax to be used for any of the description languages that describe library entities.

Specify Conceptual Language: takes a reference to a template element and a syntax. Puts the syntax in the conceptual-syntax slot of the template element.

Define Formal Language: takes a reference to a template element and a syntax. Puts the syntax in the formal-syntax slot of the template element.

Define Feature Language: takes a reference to a template element and a syntax. Puts the syntax in the feature-syntax slot of the template element.

Define operator type (*functionality of Ontology Editor*):
specifies what operators can be used in the model specific library. At least the canonical operators must be specified.

Create operator type (*functionality of Ontology Editor*):
creates an operator type (p_modeling_operation) with a given name and version, and for a given model template. Name and model template name must be specified. Version is automatically created and can not be modified. Model template must exist and is installed in the r_model_template relation. Otherwise error message and no operation.

Specify operator context (*functionality of Ontology Editor*):
takes an operator name and a context (list of symbols). Finds operator with the given name. Checks consistency of context with the model template of the operator. Installs context in the context slot of the operator. No other modifications. If inconsistency detected then error message.

Specify operator template (*functionality of Ontology Editor*):
takes an operator name, and associations between a parameter name and a component template name. Corresponding operator and component templates must exist, otherwise error. Updates the operator road-map slot accordingly.

5.2.4 Interface Requirements

The library template is basically an entity relation diagram. A graphical editor is most suitable for this. In addition there must be a way to import the data model of any of CommonKADS models as the basis for the definition of a model template.

It is useful to have easy graphical ways to combine templates or to selects parts of them and copy them. For example, the modeling components will often be very similar to parts of generic models. The template for a component should be rapidly constructed from the model template. The other way around, a model template can be based on a combination of previously defined component templates.

5.3 Library manager

5.3.1 General

The library manager allows for the creation of a series of libraries for different CommonKADS models and the maintenance of that set.

5.3.2 Data Model

The data model of the library manager is the p_model_specific_library, p_library_element, p_library_template and p_element_description (see Section 4).

5.3.3 Functional Requirements

Define Model Specific Library (*functionality of Library Manager*):
creates a model specific library (p_model_specific_library) for models that satisfy given model and component templates and description languages.

The following are the sub-functionalities required for that:

Create Library (*functionality of Library Manager*):
creates a model specific library with a given name and version. Name must be specified. Version is automatically created and can not be modified. No further relations are established.

Specify Generic Model Type (*functionality of Library Manager*):
establish r_has_model_template relation between a named library and a named model template.

Add Modeling Component Type (*functionality of Library Manager*):
establish r_has_component_template relation between a named library and a named component template.

Add Modeling Operator Type (*functionality of Library Manager*):
establishes r_has_operator_template relation between a named library and a named operator template.

Functionality: list libraries: list libraries that match a pattern of attributes that were used when creating them.

By giving only the name one checks for the existence of a library with that name. By giving only the name of a template one retrieves all libraries for a certain type of model.

5.3.4 Interface Requirements

The interface to the library manager is a window and menu-oriented facility to call the generic library functionalities. Very much like a card browser.

List of libraries: The interface display should have a pane that can display a list of model specific libraries. Every entry must mention the name of the library, the model type and the date when last changed.

The ordering can be chosen and is by model-type by default. Ordering by model-type means as a three-level hierarchical list where the first level orders the model-types alphabetically, the second level orders the library-names alphabetically, and the third level orders the libraries-dates chronologically in inverse order (most recent above).

Entries in the list are mouse-sensitive and one can be selected (highlighted) at the time. Double-clicking an item

Library attributes: The interface display has a pane listing all the attributes of a library: name, model-type, author, date when created, date when last modified, where, and possibly some statistics about size, number of models and so forth.

When a library is selected in the list of libraries then the attributes of that value are displayed as the library attributes.

When library attributes are selected and changed then this means that a pattern for filtering the list of libraries is given. In that case the display in list of libraries should be updated.

Libraries menu: a menu with three choices: new library, list all and merge-libraries.

New library: clears all attribute values in the library attributes display. Puts default values for all but name and model-type attributes. Lets user edit name and model-type. When done an ¡ok¿ button (otherwise disabled) creates the new model specific library by calling the appropriate functionality. At that point new-library operation is finished and it is added to the list of libraries.

List all: lists all libraries known to the generic library facility in the list of libraries.

Merge libraries (optional): asks for selection of two libraries, asks for a name and calls merge libraries functionality with these arguments.

When the generic library facility is quited, it must make sure that all the libraries that it knows about are properly closed.

5.4 Library Browser

5.4.1 General

The library browser allows to look at the contents of a model specific library from a birds-eye perspective. It can list the names of library elements, possibly restricted to a particular type. It allows for the selection of one of those elements, purely based on the fact that they are listed. The library browser is also dedicated to insertion of library elements, deletion of library elements and for maintaining a general view of its contents. A more targeted access facility is provided by the query subsystem. Accessed objects can be inspected by the element inspector but should be, more realistically, passed on to a dedicated tool for the type of model that the library is specific for.

5.4.2 Data model

The data model for the library browser is the p_model_specific_library and the library elements p_library_element.

5.4.3 Functional Requirements

Insert a library object (*functionality of library browser*):
add a named object to the library. Only possible in write mode. Update lists. Ask for replacement if duplicate names. If no then keep previous version(s).

Delete a library object (*functionality of library browser*):
remove a named object from the library. Only possible in write mode. Update lists. For deletion references must be removed first (keep reference counts?).

Set scope (*functionality of Library Browser*):
takes a model-specific-library and a name. The name must be the name of a p_library_template (either a p_model_template or a p_component_template). All browsing is restricted to elements that have the named template.

List generic models (*functionality of library browser*):
display an alphabetic list of library elements within the scope. When different versions are available duplicate name in the list.

List modeling components (*functionality of library browser*):
display an alphabetic list of library elements within the scope. When different versions are available duplicate name in the list.

List modeling operations (*functionality of library browser*):
display an alphabetic list of library elements within the scope. When different versions are available duplicate name in the list.

Find generic model (*functionality of library browser*):
highlight generic model(s) specified by a configuration of attributes as in a card browser and within the scope.

Find modeling component (*functionality of library browser*):
highlight modeling component(s) specified by a configuration of attributes as in a card browser and within the scope.

Find modeling operations (*functionality of library browser*):
highlight modeling operations(s) specified by a configuration of attributes as in a card browser and within the scope.

5.4.4 Interface requirement

It should be possible to view library contents by the element names and selectively for each of the three categories of library elements in separate lists and to find any of those elements based on the name.

It should be possible to switch between read and write only modes. In write mode with a user and password for write operations. It should be possible to insert and delete elements in write mode.

5.5 Reference sub-system

5.5.1 General

A reference to the library is of one of the three following types:

Reference by name: retrieves a library element with a given name from the library.

Reference by path: retrieves a library element that is associated through one or a sequence of relations with a given or named library element.

Slot reference: retrieves the value of a named slot of a library element which is given, or accessed by name or path.

These are the primary accesses to the library and treat it as a collection of library objects, structured only by their names and model or component structure.

The scope of search in the library can be set to a subset of library objects with a given template. In this way one can search for only certain modeling components or certain generic models.

5.5.2 Data model

The data model for this subsystem are the p_library_element, p_library_template, and their specializations or parts.

5.5.3 Functional Requirements

Set scope (*functionality of reference sub-system*):
takes a model-specific-library and a name. The name must be the name of a p_library_template (either a p_model_template or a p_component_template). All search for retrieval of a referenced library element are restricted to elements that have the named template.

Find library element (*functionality of reference sub-system*):
takes a model-specific-library and a reference. Returns the referenced element if it is retrievable, otherwise notifies user that element is not found.

If the reference is a symbol then the library element with that symbol as name and within the scope of search is returned. Otherwise notifies user that element is not found in that scope.

If the reference is a list then the reference is matched is with the active template and used as an access to library elements or to a slot.

5.5.4 Interface Requirements

The interface consists of one window, much like a card browser.

5.6 Element Inspector

5.6.1 General

The element inspector allows one to look in detail at the shape and contents of a library element. The navigation within a library element is based on the road-map of the template and on the different descriptions that the library element may have.

5.6.2 Data Model

The data model for the element inspector is the p_library_element, p_element_description and their specializations.

5.6.3 Functional Requirements

Functionalities to get information associated to a library element:

Class of element (*functionality of element inspector*):
returns the class of a library element. The type can be one of the three basic types of library elements.

Type of element (*functionality of element inspector*):
returns the type of an element within a given taxonomy. Any of the library elements can be organized in taxonomies. We assume only one type.

Specializations of element (*functionality of element inspector*):
the list of specializations of an element in a given taxonomy.

Show verbal description (*functionality of element inspector*):
returns verbal description of the library element. This only gives the verbal description of the element itself, if any.

Show conceptual description (*functionality of element inspector*):
returns conceptual description of the library element. This gives the conceptual description of the element itself, if any is available.

Show formal description (*functionality of element inspector*):
returns formal description of the library element. This only gives the formal description of the element itself, if any is available.

Show feature description (*functionality of element inspector*):
returns feature description of the library element. This only gives the feature description of the element itself, if any is available.

Functionalities to get information derived from an element's structure:

Virtual conceptual description (*functionality of element inspector*):
only applicable if no conceptual description is available. Returns a conceptual description of the element that is constructed based on the road-map and the conceptual descriptions of the parts of the element. The description is re-constructed every time it is asked for and never saved.

Virtual formal description (*functionality of element inspector*):
only applicable if no formal description is available for the element itself. Returns a formal description of the element that is constructed based on the road-map and the formal descriptions of the parts of the element. The description is re-constructed every time it is asked for and never saved.

Show road map (*functionality of element inspector*):
displays the road map as a hierarchy of symbols. Current element's reference is highlighted (somehow, depending on interface medium used).

Select element component (*functionality of element inspector*):
retrieves a library element that is accessible from the active road-map. This can be done graphically, if possible, by clicking in the road map of the current element being inspected.

Functionalities to create and modify elements:

Select class of element (*functionality of element inspector*):
selects the class of a library element. The type can be one of the three basic types of library elements.

Select type of element (*functionality of element inspector*):
selects the type of an element within a given taxonomy. Any of the library elements can be organized in taxonomies. We assume only one type. As a consequence the corresponding library template is selected.

Edit verbal description (*functionality of element inspector*):
edits verbal description of the library element. This only applies to the verbal description of the element itself, if any.

Edit conceptual description (*functionality of element inspector*):
edits conceptual description of the library element. This only applies to the conceptual description of the element itself, if any is available.

Edit formal description (*functionality of element inspector*):
edits formal description of the library element. This only applies to the formal description of the element itself, if any is available.

Edit feature description (*functionality of element inspector*):
edits feature description of the library element. This only applies to the feature description of the element itself, if any is available.

5.6.4 Interface requirements

Basically this is a window with five panes and an editable box. Title is the name of the object. Four panes correspond to the four possible descriptions. A small editable box displays the name of the element and this can be changed at all times. The interface calls find element on the name in that box. If an element is returned its descriptions, if available, are displayed in the corresponding pane. Conceptual and formal pane have a button labelled "virtual" that re-constructs the virtual description if possible any time it is hit. The fifth pane is the roud-map, in tree-shape, horizontal. The current focus on the library element is being highlighted. One can change focus by clicking on the corresponding node.

When a new element is being accessed based on the current element's road map then the previous element is retained on a stack that is being displayed (elements by reference). At each moment the active road map highlights the elements that have been inspected and are therefore to be found and inspected again from the stack.

Editing of an element's descriptions can call on specialized editors if these are available for the specific languages being used. For example specialized design model or expertise model editors are available. However, these may not be applicable to any defined modeling component so that it is recommended that a general default editor (textual) be available for all types of descriptions.

5.7 Query Handler

5.7.1 General

The purpose of the library is to support in knowledge engineering by finding generic models, modeling components or modeling operations with specified characteristics. Thus, a query is a partial description of an element from any of these three classes, or a combination of those. The result of the query is a stream of library elements that match the description.

5.7.2 Data model

The data model for the query handler are the library elements (p_library_element), their specializations and respective road-maps.

5.7.3 Functional requirements

Any of the three categories of library elements can be queried for using the same basic functionality. A query is based on a combination of road-maps and partial descriptions. Road maps indicate the type of library element that one is looking for, and indicate the part that one is interested in. Partial descriptions express additional constraints on the possible answers to the query. The answer is a stream of library elements that match the road map and satisfy the additional constraints.

Select query template (*functionality of the query handler*):
selects a library template from those known to the model specific library. Used as the query template. For example one can choose the template for a design model, for a computational architecture, for a specialization operator.

Specialize query template (*functionality of the query handler*):
specializes (or generalizes) a reference in the query template and updates the template accordingly. For example one can choose to specialize the template of computational architecture to blackboard architecture.

Inspect query (*functionality of the query handler*):
invoques the element inspector on the query template and allows to modify it in all its descriptions. However only the template, conceptual and feature description will be used for executing the query.

Retrieve elements (*functionality of the query handler*):
finds in the model specific library all the library elements that match the query template, the conceptual description and the feature description. The query template is matched if the library element extends it. The conceptual description is matched if it is specialized by the library element. The feature description is matched if it is logically compatible.

5.7.4 Interface requirements

The query interface may resemble very closely the interface of the element inspectors. Ideally the library works as a completion facility over partially specified models in the element inspector. Upon querying the model that matches the description is used to complete the query.

The interface for selecting and modifying the query template is a tree shaped graphical display of a template. Nodes are mouse-sensitive. Clicking can expand or close a node. It can also invoque a selection menu with specializations and generalizations of templates for that node. Selection invoques the specialize query template functionality. The inspect query interface is the library element inspector.

6 Scenarios

This section outlines, very briefly, three scenarios, one for each type of user of the generic library facility.

6.1 The CommonKADS methodology specifier

The CommonKADS methodology specifier specifies a new library for the generic model.

1. Specification of a model template for a CommonKADS model. This is done on the basis of a given model definition, e.g., for the design model.

2. Specification of a number of modeling component templates for the design model. These can be based on parts of the model template or not.

3. Specification of a number of modeling operators to use the modeling components in modeling.

4. Define a model specific library based on the model template, the component templates and the modeling operators.

The model specific library is now ready for use.

6.2 The library manager

The library manager inserts new library elements in an existing library.

1. Find library for the given type of model, e.g., the design model.
2. Add modeling components
3. Add generic model
4. Add operator instance that uses the modeling components
5. Add target generic model
6. Close the library

The library manager collects libraries from different sites.

1. merge libraries
2. browse new library
3. save new library
4. remove old libraries

6.3 The knowledge engineer

1. The knowledge engineer browses an existing library.
2. The knowledge engineer queries for a suitable model component
3. The knowledge engineer queries for a model transformation
4. The knowledge engineer creates a personnel library

7 Operational Requirements

The various functionalities of the tools and subsystems should be constrained according to the type of user. These requirements are at present not further detailed and are not crucial for the demonstration purposes of the workbench.

8 Miscellaneous notes

Note on implementation: the r_specialization_of relation on library_elements applies only to modeling components, generic models and modeling operations. For them some inheritance scheme must be provided where all slots and relations must be inherited unless they are overridden.

Note on implementation: the road map is not uniquely determined by the model definition. It may be useful, however, to have a functionality that interprets the data model of one of the CommonKADS models as a road-map.

Note on implementation: it is possible, according to this model, to have a modeling component that is related by r_part_element to a generic model or modeling component. The second case would seem strange and should be avoided. The first case, however, allows for structure sharing between a generic model and a modeling component, provided that the road map of the modeling component is compatible with (i.e., a sub-tree of) the road-map of the generic model. As this will often be the case, it may be advantageous to exploit this in the implementation.

Note on implementation: a generic model or modeling component can be reduced to textual format which is easy to save and edit on file. Only the following 6-tuple must be saved.

1. The first element contains the name and version information.
2. The second element contains the road map of the element
3. The third to sixth element contains the different descriptions in the right syntax.

Once a tool knows what templates are begin used (say, as documented in the header of the file), then it can interpret such a 6-tuple as a library element. Textual format for saving is a good idea so that conventional editing can be used to maintain and exchange library contents.

Note on implementation: descriptions could be implemented in principle as slots of library elements. However, it may be useful to allow for sharing of descriptions among different objects. For example if only the formal specification of an architecture is refined then the verbal description may remain the same. In that case the two modeling components that represent computational architectures may share the same verbal description.

Note on functionality: in an advanced implementation one would want a functionality to merge two libraries for the same model-type. This is useful when the development of libraries is going at several sites, a very likely situation. The merger of two libraries results in a new library for the same model type with a specified new name. The resulting library behaves the same for reference and query as the parent libraries. Name clashes are allowed in which case the library objects are returned as two different versions. If the versions are the same then the models are also assumed to be the same and only one is kept. The merger of taxonomies with the same name on top is done by union of sub-types. Sub-types with the same name are merged into one sub-type with two different versions if these are different.

References

[ISO9001, 1987] 9001, I. (1987). Quality systems - model for quality assurance in design/development production and servicing.

[Aamodt et al., 1992a] Aamodt, A., Benus, B., Duursma, C., Tomlinson, C., Schrooten, R., , Top, J., and Van de Velde, W. (1992a). Task features and their use in *Common*Kads. Esprit project p5248, AI-Lab, Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium.

[Aamodt et al., 1992b] Aamodt, A., Bredeweg, B., Breuker, J., Duursma, C., Löckenhoff, C. L., Orsvarn, K., Top, J., Valente, A., and Van de Velde, W. (1992b). The *Common*Kads library. Esprit project p5248, AI-Lab, Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium.

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

[Van de Velde, 1992] Van de Velde, W. (1992). Re-orienting the *Common*Kads workbench. Esprit project p5248, VUB AI lab, Pleinlaan 2 Brussels.

[Van de Velde, 1993a] Van de Velde, W. (editor) (1993). Design model and process. ESPRIT Project P5248 KADS II KADS-II/M7/VUB/RR/064/1.1, AI-Lab, Vrije Universiteit Brussel, Brussels, Belgium. with contributions from Cuno Duursma and Guus Schreiber.

[Van de Velde, 1993b] Van de Velde, W. (1993). Generic library facility: guidelines. KADS-II report KADS-II/WPM/TM.8/DD/004/1.2, Artificial Intelligence Laboratory, Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels.