Configuration Support Environment Functional Specification

Ronald Schrooten
Cuno Duursma
Vrije Universiteit Brussel
AI Laboratory
Pleinlaan 2
B-1050 Brussels
email: {ronald,cuno}@arti.vub.ac.be

Document Id.	KADS-II/DM7.4/VUB/RR/004/1.0
CEC Reference	Deliverable M.7.4, Version 1
Date	May 31, 1994
Distribution	Consortium

KADS-II Consortium

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

Cap Gemini Innovation
Paris Research Centre
86/90, rue Thiers
92513 - Boulogne
France
Contactperson: M. Georges
Phone +33.1.49.10.53.98 Fax: 33.1.49.10.06.15
E-mail: mari@capsogeti.fr

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

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

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

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

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

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

Siemens AG ZFE BT SE 1
Otto Hahn Ring 6
D-81739 München
Germany
Phone: +49 89 636 40798 ; Fax: 49 89 636 42284
Contactperson: C. Bauer
E-mail: cb@zfe.siemens.de

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

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

Free University of Brussels
Artificial Intelligence Laboratory
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

Configuration Support Environment Functional Specification

Ronald Schrooten , Cuno Duursma
Vrije Universiteit Brussel
AI Laboratory
Pleinlaan 2
B-1050 Brussels
email: {ronald,cuno}@arti.vub.ac.be

Synopsis

This deliverable contains opportunities for a configuration approach and specifications for an environment that supports the configuration of KBS design with *Common*Kads. First an analysis of the overall design process is given. Tasks in KBS design most suited to the application of the configuration approach are identified. These tasks are modelled in some depth and various problem solving methods are matched on them. The idea of using design guidelines for the configuration of a design is introduced. Configuration design via constraint satisfaction is chosen for implementation in a design configuration support tool. A prototype configuration design tool is described and evaluated. A list of functional specifications for a configuration support environment is included at the end.

Document Id.	KADS-II/DM7.4/VUB/RR/004/1.0
CEC Reference	Deliverable M.7.4, Version 1
Date	May 31, 1994
Distribution	Consortium

Contents

Chapter 1

Support for KBS design

Cuno Duursma

This section presents an analysis of the KBS design task in *Common*Kads. A short overview is given of different models of design: design as a function and design as a task. Then we proceed with investigating opportunities for KBS support in KBS design. One task, high level design, is selected and elaborated.

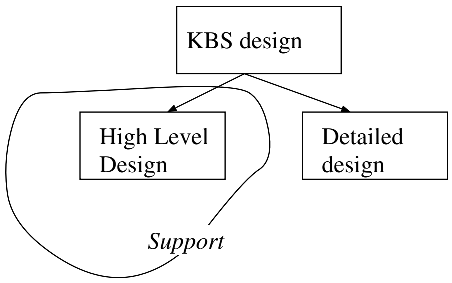

Figure 1.1: *This section discusses models of KBS design.*

1.1 Introduction

Design in general is an open ended, creative activity [Smithers, 1992]. A lot of variables have to be dealt with, many options are open and few solutions lead to satisfactory solutions. In [Bernaras, 1993] several models and views of design are discussed. Bernaras describes design as: " Design is a task in which given some needs and desires stated by a customer (or customers) a design solution is constructed such that, when it is realised, the needs and desires of the customer are satisfied". Bernaras distinguishes two approaches to modelling design; 'design as a knowledge process' and 'design as a task' . The first approach has the aim to understand the type of decisions that are being made during the design process. This approach studies design from ill-defined requirements and complex situations [Smithers et al., 1990]. It does however not offer predictive models for design decisions . The 'design as a task' approach tries to find *prescriptive* models for the the activities performed during the design process [Chandrasekaran, 1990]. This approach focuses on well defined problems only. [Bernaras, 1993] has integrated these two approaches by the use of the problem-space concept. Though this integration makes it easier to map the different views onto each other, it does not give the knowledge process oriented models more predictive strength.

In order to support the KBS designer with a KBS we have to be able to formulate a predictive model for at least parts of this task. Unfortunately, KBS design as a whole is too complex and often too ill-defined. In order to be effectively solvable a design task should belong to the 'class-3' design tasks category:

> This is relatively routine design, where effective problem decompositions are known, compiled design plans for the component problems are known, and actions to deal with failure of design solutions are also explicitly known [Chandrasekaran, 1990][page 34].

In order to find areas in design where this condition can be satisfied, we made an analysis of the design process and investigated support in different phases and sub-tasks of the design process. For instance, the designer can be helped by organising the information in such way that the 'cognitive overload' of design variables and requirement constraints is reduced. Also, the designer can be supported by using experience from previous design solutions, or checking his or her designs for intrinsic constraints, or supporting quick 'what-if' evaluations.

In *Common*Kads a KBS design is documented within the structure of the design model. This model provides among others a language to express decisions made in high level design process of KBS. Design decisions depend on the contents of *Common*Kads analysis models and intrinsic design constraints. The focus of this research reported is on a theory that predicts expert-designer's choices and its implementation in a support tool. In order to achieve this, we make a requirement analysis, a conceptual model and design of a design-supporting system for KBS.

1.2 Current design practice in *Common*Kads

The top level task in KBS design is the selection and configuration of design components that will be used to create a system with required functionality. The input of this task is distributed over five *Common*Kads models; task model, agent model, expertise model, communication model and the organisational model.

Figure 1.2 shows a diagram of the dependencies between models in *Common*Kads. The approach to the development of the models is 'risk' driven [Taylor, 1992]; this means that the methodology proposes to work first on those tasks that have the highest risk of giving development problems.Fig. 1.3

Example I *Complex interface* In the *Schedule* project [Rademakers, 1991] the acceptability of the complex interface seemed to be the major risk. The interface has the replace

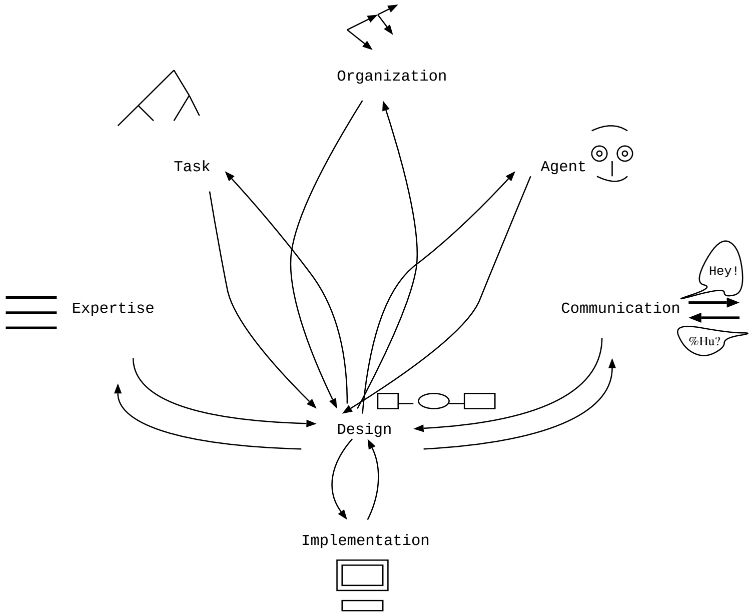

Figure 1.2: *Overview of dependencies between* Common*Kads design model and analysis models.*

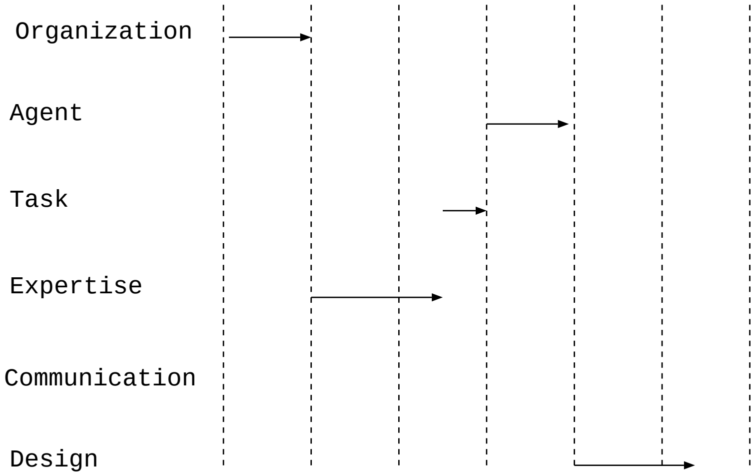

Figure 1.3: *The development of the models proceeds in a risk driven way. If a potential problem area has been identified, the model covering that area is advanced to minimize the chance of failure.*

large size paper forms, but it had to be seen whether users are able and willing to navigate by means of a visual display unit over the information. Therefore a prototype interface (maquette) of the system was built first. A second result was that the problem solving behaviour of the experts changed one they started to use the interface, because time consuming drawing operations started to be very simple, and some others became more difficult.

Example II *Complex domain knowledge* In the TROTELC project an expert system has been created that diagnoses printed circuit boards [Vanwelkenhuysen, 1991]. In this project major risk of the project was not the interface to the user, but the knowledge representation of the board to be diagnosed. This risk was however only considered when implementation had finished. This proved to be a weak point of the project later.

One of the problems the KBS designer has to deal with, is the number of requirements that must be satisfied. He or she has to integrate all these requirements which are distributed over five different models into one instance of the design model. Moreover, requirements are often contradicting and have to be reconsidered during the design process as no immediate solution can be found.

At some point in the development process, all models should have been developed to such an extend that the requirements between the models can be satisfied. Which models is elaborated first and to what extend during this process is a matter of risks and opportunities.

The following subsections present an overview of the kind of relations between *Common*Kads models and high level design decisions represented in the design model. A *detailed* design is made on the basis of the contents of the TM, EM and CM. High level design decision are made by considering the types of information within these models plus the AM and OM.

1.2.1 Requirements on KBS design from the task model

The task model [Duursma et al., 1992a, de Hoog et al., 1993b] "describes the tasks that are executed in the organisational environment where the proposed expert system will be installed in the future". The model can be represented by a (often hierarchically) structured set of frames. Among the attributes represented are input, output and goal of the task.

Several instantiations of the task model can be made during a project, for instance one describing an existing situation, a past situation, or a desired situation. Only the task model that describes the desired situation is relevant as a specification and it serves as an input to the design process.

The task model of the desired situation is a structured set of goals that should be achieved by the system and the user together. An instantiated task model of the desired situation can be part of a of an initial system model, as a description of the functionality of the system to be delivered. During the requirements analysis phase it can be elaborated to a (functional) requirements definition with a communication model. In projects following the traditional 'waterfall' cycle, the Task Model is the starting point for expertise modelling.

In projects following the risk driven life cycle this is often the case too, but problems foreseen in expertise modelling could overrule the traditional ordering of phases.

The type of the task in the required TM often guides the selection of the architecture. However, the selection of the architecture also depends on the problem solving method selected in the EM. If the task cannot be matched on any known task, it might be interesting to build a new task specific architecture.

1.2.2 Requirements on KBS design from the agent model

The agent model [de Hoog et al., 1993b, Waern et al., 1993] represents the capacities of the various people and systems that play a role in either the existing or the desired situation. The agent model is part of the problem description and can be delivered with the requirements definition.

A second use of the agent model is to check the validity of the task and the communication model. The capacities of the agent should match the capacities needed for the tasks assigned to it in the task model. The ingredients assigned to agents in the communication model should really owned by the agent. Only an agent model describing the desired situation is relevant for the design process.

Some non-functional requirements can be derived from the agent model, like the expertise of the user, the responsibility for the results.

1.2.3 Requirements on KBS design from the communication model

The communication model [de Hoog et al., 1993b, Waern and Gala, 1993] describes the communication and cooperation between users and independent system components. It describes all transactions between different agents of the system environment. The communication model thus represents requirements to communication with the future system. The communication model is currently described in a non formal way.

Functional requirements to a system can be described in terms of input output behaviour and completed with a set of non-functional requirements. The communication model is therefore an important source of functional requirements. The basic information of the communication model is what is transferred between the various agents, and who has the initiative for the transfer.

The level of detail for the requirements specifications can vary. It is possible to describe the communication until the very last mouse-click, but it is also possible to describe the communication at a higher level of abstraction, with just a reference to the specific communication metaphor to be applied, like for instance direct manipulation.

In [de Greef, 1989] a communication model has been introduced that represents some communication requirements a way structured around tasks, ingredients and initiatives. This is also how the *Common*Kads model of communication has been shaped. Though the models give some structure and focus to the investigation of the functional requirements; the formulation of the requirements if still a creative task; that can only be done successfully by a team that knows about the user needs and the technological possibilities and costs.

Several requirements to the high level design can be derived from the communication model. For instance, the initiative type and the interaction of the user, the kind of data and knowledge manipulation and the types of explanation the user requires.

1.2.4 Requirements on KBS design from the expertise model

Expertise models can be created during the analysis and design phases of a project. During analysis it is used to get an overview of the reasoning and the domain knowledge applied to the task. In the design phase it is used to make a high level and later also a detailed design of the inferencing knowledge and domain knowledge of the system.

The expertise model [de Hoog et al., 1993b, Wielinga (Ed.), 1993] refines the functional specification of the tasks assigned to the system by identifying a an approach to solve the tasks stated in the task model.

While the task model shows the input/output relations required in the system and the communication model the required communication between the system and the rest of the world; the expertise model represents a first step to how these I/O pairs can be achieved. The task and the communication model reflect behavioural views on the system; the expertise model provides an analysis of how the behaviour can be achieved in a 'rational way'.

Several requirements to the high level design can be derived from the expertise model. For instance, the type of problem solving method is related to the type of architecture.

1.2.5 Requirements on KBS design from the organisation model

The organisation model [de Hoog et al., 1993b, de Hoog et al., 1993a] represents knowledge about organisational aspects related to KBS development. Some requirements to the high level design can be derived from the role of the system in the organisation, like the environment in which the KBS will be deployed, maintenance requirements related to the stability of the organisation, organisation policies for hardware and software, etc.

1.2.6 Summary of requirements

- *Task Model*: The required functionality in terms of a natural process.

    1. The required functionality of the KBS in terms of a natural process.

- *Communication Model*

    1. Initiative type for the tasks.
    2. Initiative type for the transfer of ingredients.
    3. Explanation type

- *Expertise Model*

1. Problem Solving Method
2. Assumptions
3. Domain Ontology

- *Agent Model*

  1. Agent type

- *Organisation Model*

  1. Environment
  2. Maintenance requirements
  3. Hardware and software policies

The top level task of a 'typical' software engineer designing a KBS using *Common*Kads can be considered decomposed of several sub-tasks (see the TM worksheet in Fig. 1.4).

1.2.7 Required Task Model: design with configuration support

The task model of the current situation is the starting point of our search for opportunities to improve KBS design with *Common*Kads. We have focussed on *high level design*. *Common*Kads specific approaches to *detailed design* are discussed in [Van de Velde, 1993a] and [Duursma et al., 1992b].

We assign the Task to the system to *predict the use of design components from provided requirements in the analysis models*. Secondly, we have to investigate whether a computational realisation of this configuration design process can be made. The order of these activities is not necessarily fixed, but determined by the methodology.

In a structured software engineering process the 'components' are not only the computational programming language constructs. Higher level constructs also play an important role. A good choice of high level design concepts reduces the complexity of the design process, improves the efficiency of the software and also determines the strength of secondary requirements. The theory we are looking for should predict the choices for elements a range of different concepts. These concepts are listed in the *Common*Kads DM; they include:

1. *Application specification decomposition.* The way the tasks from the TM are distributed over separate specifications. Different application specifications can use different specification languages. This can be useful when they specify information of different nature, like interfacing and reasoning tasks.
2. *Application specification language.* The selected language to specify the application. This can either be a language with formal semantics, an operational language, or a conceptual language.
3. *Application specification.* The specification of the required behaviour in terms of the application specification language.
4. *Detailed design.* The detailed design of the application specification in terms of the architecture.

TM

Continuation of: *None*

Common Kads Task Model Worksheet

Task Model id: *Design .1*

Management

Model Type: [x] existing [ ] target: [ ] current at: [ ] other:......................... *(tick one)*

Situation: *current* *(the situation in terms of states of models or text)*

Objective *model 'today' practice in KBS design.* *(the objective of making this model, in terms of situation or states of models)*

Risk *unclear where best support to KBS design should be given.* *(the risk you want to avoid)*

Rationale: *The configuration support system should be able to help with time consuming design tasks for which we have the theories and technology available to solve it.* *(why you think this will avoid it*

Target State *(a model state of a type of task model)*

Decomposition approach: [ ] hierarchical [ ] clustering [x] model-based *(design model)* [ ] other *(tick one)*

Task Decomposition

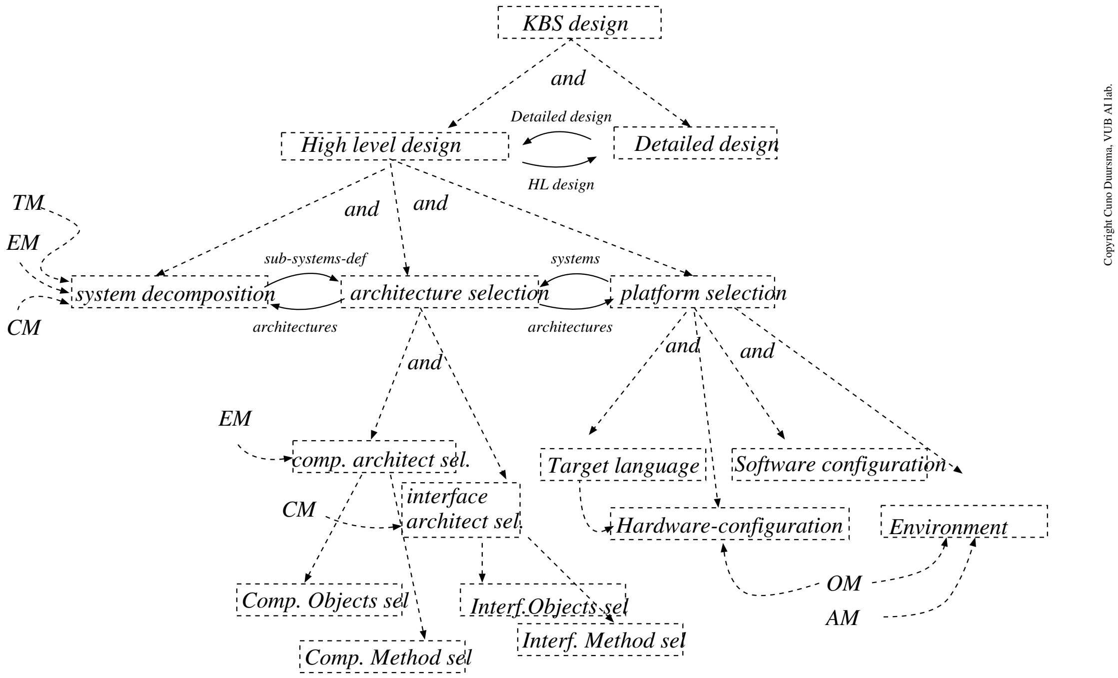

Figure 1.4: *Decomposition relation and ingredients transfer in the task model that represents of the* KBS *design task without* KBS *support.*

5. *Architecture.* The choice, or design, of the architecture used for the application.
6. *the Detailed application Design.* The design of the application specification on top of the selected architecture(s).

If a suitable architecture is not available, then it needs to be designed. Architecture design can be modelled in an additional layer using the same ontology; the task refers in this case to the design of the architecture, the application specification the specification of the functionality of the architecture. This can be repeated a number of times to create several means-ends layers. The DM comes 'standard' with terminology where the application is realised by two means-ends levels, first the application design on the architecture, and secondly architecture design on top of the target language.

A theory for design should preferably predict accurate values for all of the slots in the design model: the decomposition of the application specification, the application specification itself, the architecture, the detailed architecture, the selection of the target language, and the detailed architecture design on top of this target language. Whether such a theory can be developed depends on our insight in the design process and on the observables that can be used for the prediction.

TM

Continuation of: *None*

Common **Kads Task Model Worksheet**

Task Model id: *Design .1*

Management

Model Type:	☐ existing ☒ target: ☐ current at: ☐ other:..........................	*(tick one)*
Situation:	*required*	*(the situation in terms of states of models or text)*
Objective	*model KADS KBS design with configuration support.*	*(the objective of making this model, in terms of situation or states of models)*
Risk	*Set the scope of the configuration tool support.*	*(the risk you want to avoid)*
Rationale:	*The configuration support system should be able to help with time consuming design tasks for which we have the theories and technology available to solve it.*	*(why you think this will avoid it*
Target State		*(a model state of a type of task model)*
Decomposition approach:	☒ hierarchical ☐ clustering ☒ model-based ☐ other	*(tick one)*

Task Decomposition

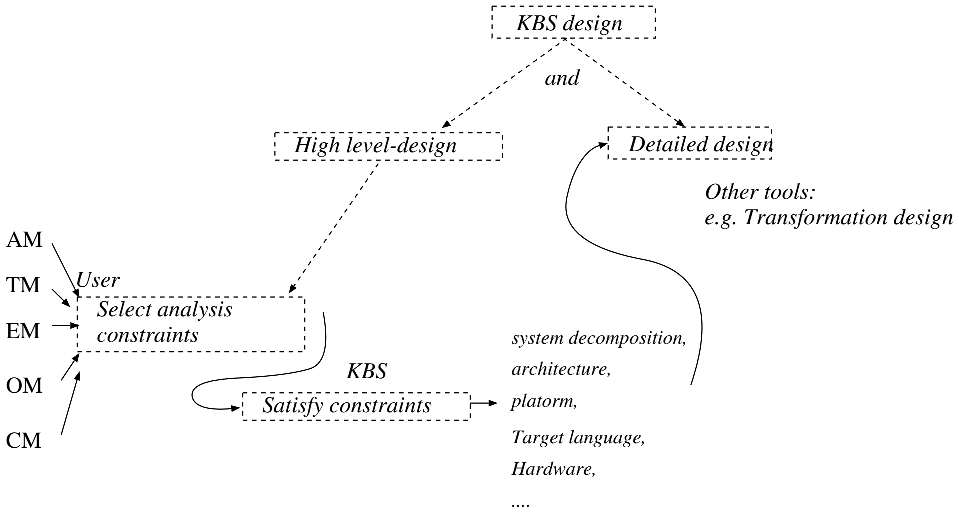

Figure 1.5: *Required task model: Design with configuration support.*

Chapter 2

Theories for high level design

Cuno Duursma

In this chapter we apply different problem solving methods on the high level design task. For each of these methods we investigate the resulting model ontology requirements and -assumptions. The results are compared and on this basis constraint satisfaction is selected and elaborated upon

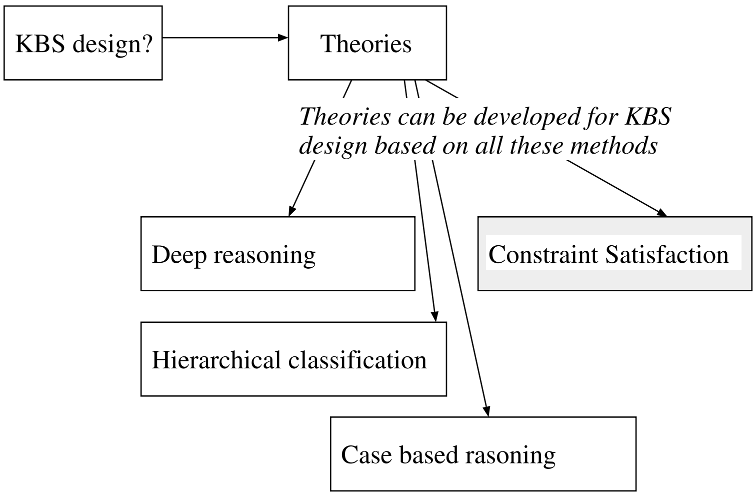

Figure 2.1: *This chapter discusses different methods that can be used to formulated a design theory.*

2.1 Introduction

A KBS can perform a task if a predictive theory is available or can be developed about the task. In this section we analyse the process of component-selection in high level design and show that a predictive theory for the selection of design components can be formulated.

- *Natural process:* the selection of design components as specified in the *Common*Kads DM. These components range from pre-existing system decompositions, architectures, fragments of architecture specification, pieces of application specification, or combinations of those.
- *Experiment:* What design model configurations are chosen by an expert KBS designer for a set of *Common*Kads analysis models.

The high level design theory should predict the use of design components from a configuration of analysis models and constraints between design elements. The theory should associate problem situation types with a selection and configuration of design elements. The associations should be bi-directional in order to make them also useful for predicting the constraints on the requirements starting from the configuration of the design.

2.2 High level design approaches

The following few sections introduce different approaches for formulating, formalising or operationalising high level design. The approaches are not completely orthogonal. They are all "generic components" in the sense of the *Common*Kads library [Breuker and Van de Velde, 1993]: components that can be used to construct a predictive model. For instance, high level design can be mapped on the generic task "configuration". Configuration can be performed by classification, by case based reasoning and also by constraint satisfaction. High level design can also be mapped directly onto these components without the mapping to a configuration task. The result is for instance high level design by case based reasoning or high level design by classification. However, if the mapping to a 'generic component' is made via a generic task available in the *Common*Kads library, then the library's model ontology translation tables can be used.

We first interpret high level design in terms of the generic configuration task. The next subsection interprets high level design as a classification task. After that we try a design approach based on first principles. Finally we present high level design as configuration via constraint satisfaction.

2.2.1 High level design as a configuration task

Configuration design is refers to designs where the problem is to find values for a set of parameters that satisfy a number of constraints. More specifically:

> Parametric design can be characterised as a kind of design where a set $E = e_1, e_2, \ldots e_n$ (elements), $R = r_1(e_1, e_2, \cdots e_n), r_2(e_1, e_2, \cdots e_j), \cdots$ (relations) and $S =$ (structure) are given (...). Constructing a solution in parametric design is the process of instantiating the values of the attributes of E in S such that R holds.

(Adapted from [Bernaras, 1993])

Examples of configuration task are the office assignment (SISYPUS '92) and the elevator design problem (SISYPUS '93 [Yost, 1992, Gaines, 1994]).

We take these constraints as a starting point and develop a theory that can predict good design configurations for KBS. There are several alternative directions in which the configuration theory can be developed from this starting point. The different alternatives have different model ontology requirements [Duursma et al., 1993], different consilience [Duursma, 1993], simplicity and make different assumptions.

The detailed design of a KBS is not a configuration task: the variables are not known in advance, all constraints cannot be enumerated nor can all possible values if the variables be enumerated. There is simply too much freedom in the number and the range of the solution variables. This makes the search space for the applicable configurations infinitely large. For the other choices of the design the dimensions of the solution are known, and less alternatives available. Design choices can also be associated with categories of requirements specifications.

2.2.2 High level design via a classification method

The simplest approach to high level design is to classify the problem case into a known and previously solved category; retrieve the solutions for that category and present these results as the prediction of the theory. In its most simple form classification is just a table lookup kind of theory. Usually these categories are organised hierarchically. In the Clancey's model [Clancey, 1985] the class hierarchy has been split into two distinct hierarchies linked with matching relations. One hierarchy contains classes representing various problem categories, the other hierarchy represents solution categories. Elaboration of the configuration theory by hierarchical classification results into the following model ontology requirements for the configuration theory:

Classification	**Configuration design**
problem class	specification type
problem class feature	feature of the specification; an aspect of the configuration of a number of *Common*Kads analysis models
matching relation	specification type – design solution type relation
problem-class – problem super-class relation	General problem-statement – specialised problem-statement relation.
solution-class – solution super-class relation	General design solution – refined design solution relation

Hierarchical classification can be performed in several distinct ways, see for instance [Breuker and Van de Velde, 1993] for examples of the application of four different role limiting methods applied to classification.

Though the structure of a theory based on classification is straightforward, the model ontology requirements listed above cannot easily be satisfied for configuration design. For configuration of *Common*Kads analysis models the relevant design model structure needs be listed. If the design model has n different parameters ranging over an average of v different values; then v^n solution classes are required. In the same way, if these solutions are required for i parameters in the *Common*Kads analysis models ranging over an average of w values, then a total of w^n problem classes are required.

These results make configuration design by classification feasible only for high level design: situations where a limited number of aspects of the five analysis models can be used to set some parameters in the design model. Knowledge acquisition for a theory of detailed configuration design by classification seems unfeasible.

2.2.3 High level design via a deep reasoning method

Another possibility is to find underlying design principles that make one configuration of the design model more successful for a set of *Common*Kads analysis models in a certain case than another configuration. This approach could be based on a theory how design choices affect the primary and secondary design requirements. It is not easy to describe how, for instance, the choice for a specific architecture affects the maintainability of a classification task starting from first design principles.

This approach requires a set of general design principles that predict the configuration of the design model from the configuration of the analysis models. This is feasible for detailed specifications. A theorem prover is actually a predictive design theory able to produce operational specifications from formal specifications.

It is difficult to formulate general principles to predict the values of parameters representing higher level design choices, like architecture and different application decompositions. However, the *structure preserving* design approach described in [Duursma et al., 1992b], is based on the design principle of structure preservation. This simple but powerful design principle predicts values for higher level design parameters like application decomposition and part of the detailed application specification design. Though quite powerful, this single principle does not make valid predictions in cases where: (1) different, contradictory requirements with different structure are used in the models. (2) An architecture is applied that is different from the language used to formulate the requirements.

Principles covering these difficult cases would have to take more detailed information of the analysis models into account. For instance, both classification task and the blackboard architecture would have to be described at much finer degrees of detail; like the ordering of data, the distribution of functionality, the match of the classification on the function decomposition on the black-board. Only then basic design principles like *maximal coherency* and *minimal dependency* could be used to configure the design model.

2.2.4 High level design via a case based approach

Extrapolation from similar cases is similar to the classification approach, but instead of abstracting in advance, the cases are kept and abstracted and matched only when they are required for solving a new case.

Model ontology requirements (called vocabulary) of this approach are[1]:

[1] See [Duursma, 1992b] for a discussion on the model ontology requirements of 'extrapolate from a similar case'.

- *Case:* Refers to a combination of description of an analysis models and a configuration of the design solution that worked in the past.
- *Similarity:* A measure to express the difference between the new case and the old case. This similarity measure is difficult to develop for qualitative cases like KBS design.
- *Extrapolation function:* The method used to estimate modifications to the available case solution from differences between the available case problem and the new problem.

The case based approach can be based on the higher level classifications or on basic design features: this is determined by the chosen indexing features.

2.2.5 High level design via a constraint satisfaction method

Another approach to elaborate the design theory is to focus on the constraints a choice of one element of a solution for a problem makes on the other elements that can be selected. It is a kind of jig-saw approach to problem solving; the selection of one element of the puzzle sets requirements for the neighbouring parts; and these set requirements again for their respective neighbours again, until the whole board has been covered.

Like the case-based-approach, constraint satisfaction can be used on deeper principles or on compiled knowledge. The model ontology requirements of the configuration design by constraint satisfaction are as follows:

Constraint satisfaction	**Configuration design**
variable	specification parameter, design model parameter
domain	list of possible values of each specification and design model parameter
constraint	list of tuples of requirements parameter values and design parameter values.

The constraint satisfaction approach does not work by abstracting problem cases to abstract problem descriptions and associating these with abstract and more refined solutions. Instead, constraints between the problem and the solution are represented. For the configuration approach this means that the specifications constrain the possible values for parameters in the design model.

2.2.6 Evaluation of alternative methods

It is not possible to develop a configuration theory for design with 'easy' model ontology requirements, high consilience and simplicity and limited computational complexity. The design task is simply too open ended: the complexity of the design task is too high.

The different alternatives discussed pose different opportunities and problems for the elaboration of the design theory. Elaborating the theory as a classification problem requires the identification of a lot of problem solution classes. An approach following basic designers principles requires general and widely applicable principles. These are only partly available. The case based approach and the constraint satisfaction approach seem feasible if cases or constraints are chosen at a suitable level of detail.

Chapter 3

Configuration design via constraint satisfaction

Cuno Duursma

This chapter shows the analysis and specification of the hierarchical constraint satisfaction method for Operationalisation of the generic parametric design task. As the high level design task has been described in terms of this generic task in the previous chapter, this will produce an Operationalisation of the high level design task.

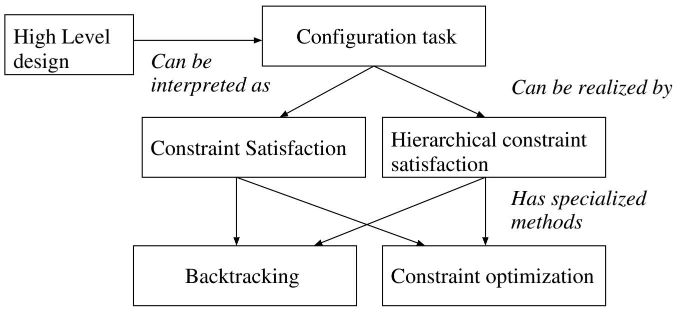

Figure 3.1: *This chapter discusses how constraint satisfaction can be realised and how can it be mapped on configuration design.*

3.1 Introduction

Constraint satisfaction refers to a range of AI techniques and systems with algorithms satisfy relations in a network of variables. Different approaches have been developed, like backtracking, constraint propagation and hierarchial constraint satisfaction. Besides these techniques, the constraint approach is also important for is model ontology requirement (see [Duursma, 1992a]): to analyse a problem in terms of variables and constraints between these variables. A range of problems can be represented naturally as constraint satisfaction problems, for instance parametric design tasks and scheduling tasks.

Two classes of constraint satisfaction problems (CSP) can be distinguished: Boolean CSP's and constrained optimisation problems.

3.1.1 Boolean constraint satisfaction problems

A Boolean CSP is characterised as follows [Mackworth, 1987][page 206]

- Given is a set V of n variables $\{v_1, \cdots, v_n\}$.
- Associated with each variable v_i there is a a domain D_i of possible values.
- On some specified subsets of those variables there are constraint relations given, that are subsets of the Cartesian product of the domains of the variables involved.
- The set of all solutions is the is the largest subset of the Cartesian product of all the given variable names such that each n-tuple in that set satisfies all the given constraint relations.

Reporting on the solutions can either consist of the complete set of solutions, of one particular solution, or of the empty set if the SCP is unsatisfiable.

The satisfiability decisions for a CSP is equivalent to determining the truth of a well-formed formula in first-order predicate logic:

$$(\exists x_1)(\exists x_1)\cdots(\exists x_n)$$
$$(x_1 \epsilon D_1)(x_2 \epsilon D_2)\cdots(x_n \epsilon D_n)$$
$$P_1(x_1) \wedge P_2(x_2)\cdots P_{12}(x_1, x_2) \wedge P_{13}(x_1 x_3) \wedge \cdots \wedge P_{n-1}, P_n(x_{n-1} x_n)$$

P_{ij} is only included in the formula above if $i < j$ since it is assumed that the constraints are bi-directional and P is therefore commutative, i.e. $P_{ij}(x_i x_j) = P_{ji}(x_i x_j)$.

Algorithms for Constraint Satisfaction Problems Two types of algorithms have been developed for solving CSP's: backtracking algorithms and consistency algorithms. The first class of algorithms explore the set D by:

1. sequentially instantiating the variables $\{v_1 \cdots v_n\}$,
2. checking the validity of the constraining relations,
3. re-instantiating variables with alternative values if relations cannot be fulfilled.

Consistency algorithms are complementary to the backtracking algorithms. Consistency algorithms have been developed after a careful analysis of thrashing behaviour in backtracking algorithms [Mackworth, 1977]. They can be most easily be explained by the use of the network model:

- For binary constraints each edge in the graph between vertices i and j is replaced by arc (i, j) and an arc (j, i)

- A Node, consisting of a vertex i and the associated domain of variable v_i is *node consistent* if: $\forall(x)[x \epsilon D_i] \supset P_i(x)$

  This means that for each of the nodes, the variable assignment satisfies the local constraints. This does imply not that all constraints have been satisfied.

- The graph is *arc-consistent* if:

  $(\forall x)[x \epsilon D_i] \subset (\exists y)(y \epsilon D_j) \wedge P_{ij}(x, y)$

3.1.2 Constraint optimisation problems

Constraint optimisation extends constraint satisfaction with optimisation support. In problems we like to solve different constraints often have different priorities in being realised in the solution. Weights or preferences can be specified in different ways, and can be combined using several schemata. The definition of constraint optimisation is not as clear cut as in boolean constraint satisfaction; and several algorithms give their own interpretation to the problem.

3.1.3 Hierarchical Constraint satisfaction

Hierarchical constraint satisfaction (HCS) can be used to improve the representational strength of boolean constraint satisfaction and constraint optimisation by taking advantage of the common structure: in many real-world problems, objects can be grouped into hierarchical classes according to some of their properties. This feature makes hierarchical constraint satisfaction a useful representational device. Isa relations are transitive relations and can therefore be represented by a directed acyclic graph (DAG).

CONTAX [Meyer, 1992] is an example hierarchical constraint satisfaction system. CONTAX offers boolean constraint satisfaction and also, though limited, constraint optimisation. In CONTAX, domains are arranged in isa hierarchies. Constraints can be defined between constraint variables, and sub-variables inherit the constraints of their supers. Inheritance of constraints is formally defined as follows in CONTAX:

Definition **Inheritance of constraints in CONTAX**: *Let* $R_j \subseteq D_1 \times \cdots \times D_i \times D_k$ *be a constraint,* $(d_1, \cdots, d_i, \cdots, d_k) \epsilon Rj$ *be a tuple in* R_j. *Then for all* $\tilde{d}_i \epsilon D_i$ *the tuple* $(d_1, \cdots, \tilde{d}_i, \cdots, d_k) \epsilon R_j$ *if* ISA$(\tilde{d}_i, d_i)$.

3.2 High level Design as a HCSP

The final step is to formulate the high level design problem in terms of hierarchical constraint satisfaction. Once this has been done, the resulting task-, domain- and method-specific ontology can be 'stuffed' with design knowledge.

HCSP required the identification of:

1. The domain hierarchy,

2. The variables,
3. Constraints,
4. and Constraint strenghts

The *domain hierarchy* is split into two parts; analysis situations on one side and design solutions on the other. On each of these sides several hierarchies can be build representing either analysis or design decisions in abstraction hierarchies linked by ISA links (see figure Fig. 3.2)

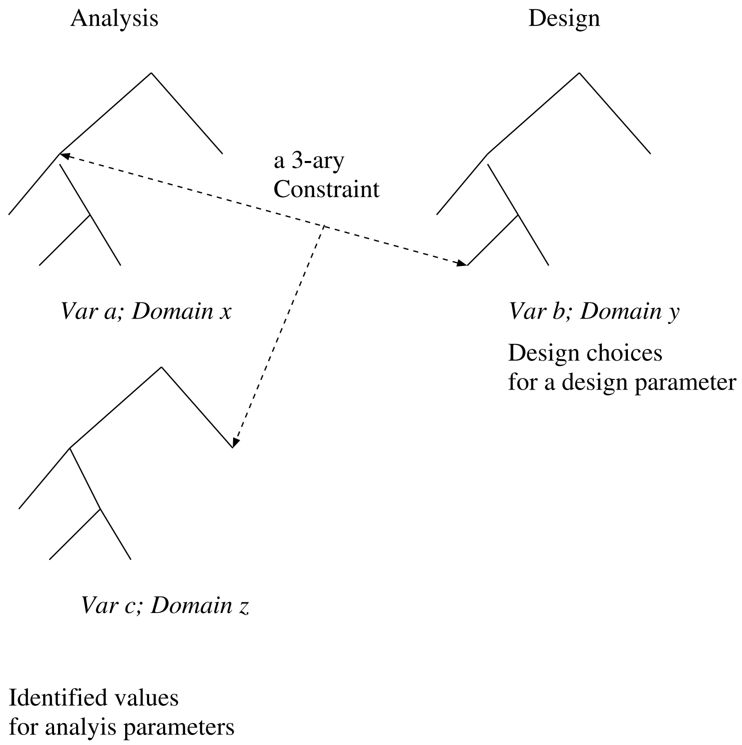

Figure 3.2: *Examples of domain hierarchies in* HCSP *applied to high level design.*

The *variables* represent analysis situations and design decisions. Each of the variables ranges over a set or a hierarchy of possible values. The next chapter will show how design principles are used to choose these variables.

The *constraints* represent constraints between analysis and design situations. They can be of any arity higher then zero. Constraints can be represented between different abstraction levels in the domain hierarchies.

The *constraint values* represent the importance of satisfying a constraint. *Hard* constraints must be satisfied for a succesfull design. *Weak*constraints should preferably be satisfied but are not essential.

Chapter 4

Design Guidelines

Ronald Schrooten

This chapter introduces design guidelines as knowledge that is used in the configuration design process. Examples are provided and the relation with constraint satisfaction is identified. At the end a brief evaluation of the approach is included.

4.1 Introduction

Design guidelines form a bridge between the *Common*KADS analysis models and the design model.
A design guideline is a relation between a design component and a recognisable feature in the problem domain. We stress the fact that a design guideline should always be related to a design component. In this approach of configuration design the design components are the reusable building blocks. Recognisable features of the problem domain can be needs, requirements, facts or other specifications in the analysis models. The fact that they are recognisable depends on whether they can be discovered during analysis or not.
Because of this strong connection with design components we believe that useful design guidelines can only be introduced via investigation of reusable design building blocks. A lot of work in this area is done in *"Designing Expert Systems - A Guide to Selecting Implementation Techniques"* [Kline and Dolins, 1989]. Most of the guidelines that we use are adopted from this book.

4.2 Guidelines as Constraints

In our configuration design approach we implemented the design guidelines as bi-directional constraint relations. In constraint satisfaction terminology this means that we use the guidelines for identifying the constraints between several *domain values* which serve as instances for constraint *variables*. In this way we introduced meaningful knowl-

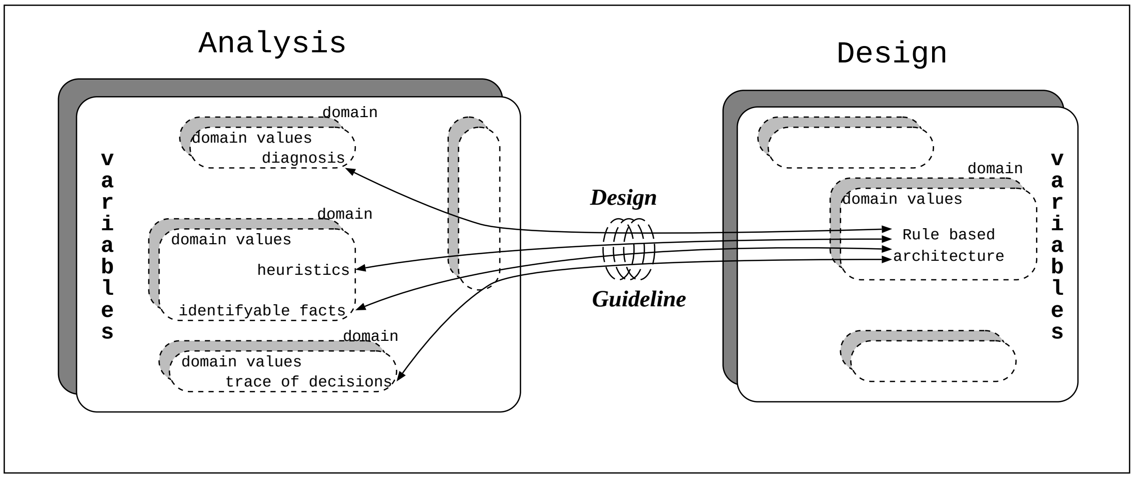

Figure 4.1: *A design guideline is knowledge that is used to bridge the gap between the analysis models and the design model.*

edge that can be handled by a constraint satisfaction inference mechanism.

4.3 Examples

Examples of design guidelines represented as constraint relations between reusable design components and recognisable analysis features.

Guidelines related to a blackboard architecture. First we give a brief description of a blackboard in terms of the Kads design model [Golfinopoulos, 1993, Van de Velde, 1993a].

Name: Blackboard

Content Description: A blackboard architecture consists of three major components: *knowledge sources* which contain the knowledge needed to solve the problem and can work independent. A *blackboard data structure* that keeps track of the global problem solving state. A *controller* that let the knowledge sources respond to changes in the blackboard. [Velthuijsen, 1992].

Architecture design .

```
Computational:
  Paradigm: Blackboard;
  Command: read; (from the blackboard)
           write; (write on the blackboard)
           match; (monitor the blackboard)
           act; (perform some activities)
           start;(control)
           end; (control)
```

```
Uses: Objects used by the knowledge sources (unit, hypotheses,
                                  elements, attributes, etc.);
Supports: knowledge sources; (a KS is a an activity, implemented
          as a piece of program, called the KS action part,
          together with a data structure that describes the action part.
```

Design Guidelines:

- There is a constraint relation between the *use of a blackboard data structure to record hypothesis* and the fact that *solving the problem involves transformation between several different levels of description* or *the availability of several diverse knowledge sources*.
  These features can be found through analysis done in the expertise, organisation and agent model.
- There is a constraint relation between the *explanatory capabilities of a blackboard system* and the fact that *explanation should concentrate on accounting for details of the situation under analysis and not on explaining the reasoning*.
  These features can be found through analysis done in the expertise and communication model.
- There is a constraint relation between the *control mechanism of a blackboard architecture* and the fact that *an opportunistic problem-solving strategy is required (i.e. decide dynamically on a problem-solving strategy* or *a planning task type is identified.*
  These features can be found through analysis done in the expertise and task model.

Guidelines related to a rule-interpreter architecture. First we give a brief description of a general rule-interpreter in terms of the Kads design model.

Name: Rule-interpreter

Content Description: A Rule-interpreter architecture uses rules to store knowledge (in most of the cases heuristics derived from an expert). Rules are condition(antecedent)/action(consequent) frames that can be handled by the inference mechanism. Selection of the rules is managed by a control strategy and conflicts are resolved via a conflict resolution strategy. [Bahrami, 1988]

Architecture design .

```
Computational:
  Paradigm: Rule-based;
  Command: rule of inference;
           control strategy;
           Conflict-resolution strategy;
 Uses: Objects used by condition and action parts of the
       rules or the rules themselves;
 Supports: rule of inference: modus ponens, first order logic,
                horn clauses.
           control strategies: forward chaning, backward
                chaning.
```

```
conflict-resolution strategy: rule ordering, size
    ordering, least recently used, most recently used,
    data ordering.
```

Design Guidelines:

- There is a constraint relation between the *shallow reasoning capabilities of a rule-based architecture* and the fact that *the problem type is in the area of diagnoses and can be solved through the use of heuristic that shortcut the causal path of symptoms to conclusions*
  These features can be found through analysis done in the expertise and task model.

- There is a constraint relation between the *ability of representing and reasoning with incomplete knowledge in a rule-based architecture* and the fact that *incomplete knowledge of facts of the problem, limited access to information in this environment so that these uncertainties about facts of the case cannot be resolved simply by asking for the information, important to achieve as complete as possible an understanding of the fact of the case*
  These features can be found through analysis done in the expertise, communication and organisation model.

- There is a constraint relation between the *explanation in a rule-based architecture* and the fact that *it is sufficient to get explanations that tell how the expert system arrived at its answer*
  These features can be found through analysis done in the expertise and communication model.

- There is a constraint relation between the *gathering information in stages through the use of "screening clauses" in rules* and the fact that *certain facts should be established before the expert system request a particular item of information*
  These features can be found through analysis done in the expertise, task and communication model.

4.4 Evaluation

4.4.1 Advantage

Besides the benefits of guiding a designer towards a reusable design this approach has some extra benefits.

Maintainability : Having the link between needs taken from the analysis models and the actual design of the software can be an improvement over current debugging and maintenance methods [Green et al., 1986]. Design guidelines provide this link. A maintainer can use these guidelines to see if original stated needs and requirements are not violated by maintenance changes in the design of the KBS.

Rational : The design is configured via the use of the design guidelines. These guidelines represent the taken design decisions. A report of all the guidelines that were used to construct the design model can serve as a rational for the design at hand.

Compromising : Design is often stated as a problem of compromising on requirements that are formulated by interest groups (i.e. users, management etc.) [Vanwelkenhuysen, 1993]. A designer can now *plug and play* with the severable compromisable requirements and see to what kind of designs the different combinations result. The inner relations of the different requirements and their impact on the design will become more clear and the several possible designs can be generated and compared very fast. This can be seen as a small tornado in the huge spiral model of the KBS development life cycle Fig. 4.2.

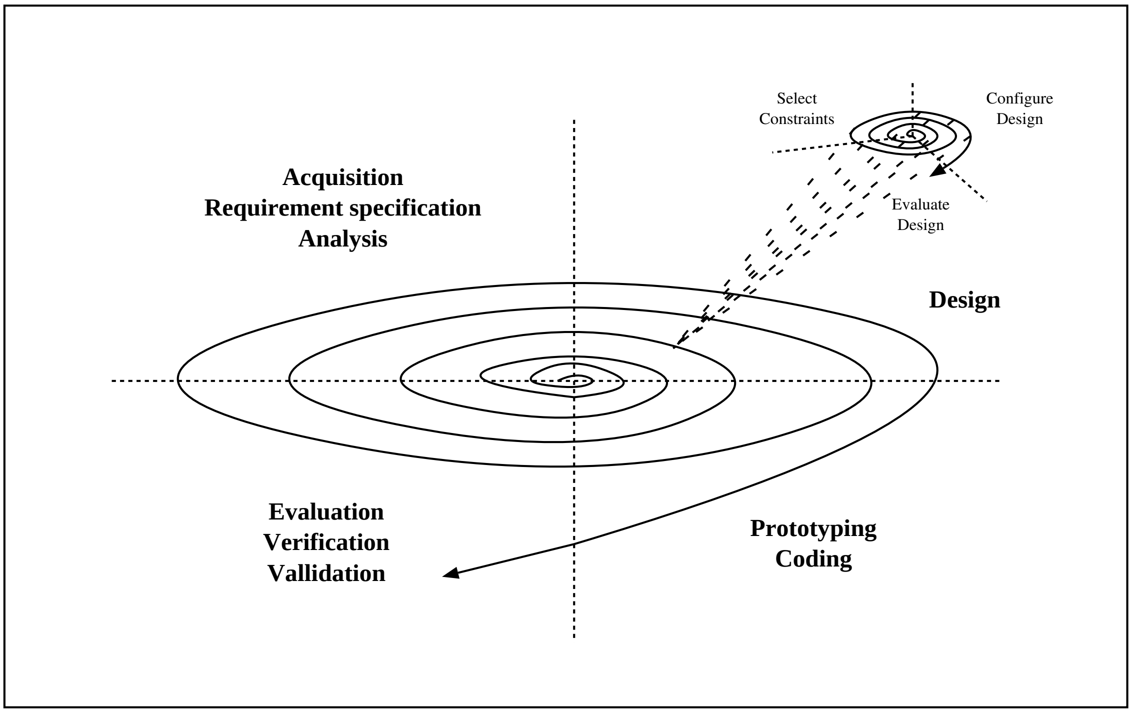

Figure 4.2: *The design process is in itself a spiral development process within the overall spiral model of* KBS *development.*

Validation : Validation is testing the issue of *building the right system.* This reflects on the relation between the *system itself* and with what was *expected* from the system. Design Guidelines provide support for this relation because they make this link very explicit. Furthermore it becomes possible to inspect, motivate or criticise the resulting system on the based of the knowledge that the guidelines represent.

4.4.2 Disadvantage

Knowledge Acquisition : Acquiring the correct guidelines for each design component is difficult.

Terminology & Context : Working with reusable component requires a general terminology to identify them. In this guidelines approach we use analysis features. Because there is no fixed terminology we create the problem of mismatching features that have the same meaning but have a different name because they where formulated in a different context.

Theory : A good metric or reliable theory to reduce the set of possible solutions and order them according to quality criteria is not included.

Chapter 5

Design Process

Ronald Schrooten

This chapter focusses on the transition from knowledge acquisition to implementation. This transition process is identified as the design process in the knowledge engineering life cycle model. The identified design process will be used in the configuration design environment.

5.1 Introduction

Traditional software engineering [Boehm, 1988] showed that there is a large distance between an informal description of knowledge and the implementation of this informal description. Intermediate stadia were worked out in order to bridge this gap. The *Common*Kads methodology adopted this engeneering via phases. The basics behind *Common*Kads are that they take a modelling approach and that they focus on knowledge based systems. In order to bridge the previous mentioned gap *Common*Kads provides six models. Each model focusing on a specific type of information. One can take the perspective in which five models (Organisation, Task, Agent, Communication, and Expertise model) serve as input [1] for the sixed model (Design model) Fig. 5.1.

We want to support this process of filling the design model. The goal is that the outcome of this process is a model of design suitable for the implementation phase of a KBS.
In this deliverable we have taken a configuration design approach based on constraint satisfaction and design guidelines. We will elaborate this process in the forthcoming sections.

[1] We do not exclude additional information that is not modelled in one of these five models. For instance, design specific knowledge from an expert in KBS design or case specific knowledge can also be used as additional input for the to be constructed design. A similar view on the design process of a KBS is described in [Schreiber, 1992]. The model of expertise and the model of cooperation are identified as the major input for the design process.

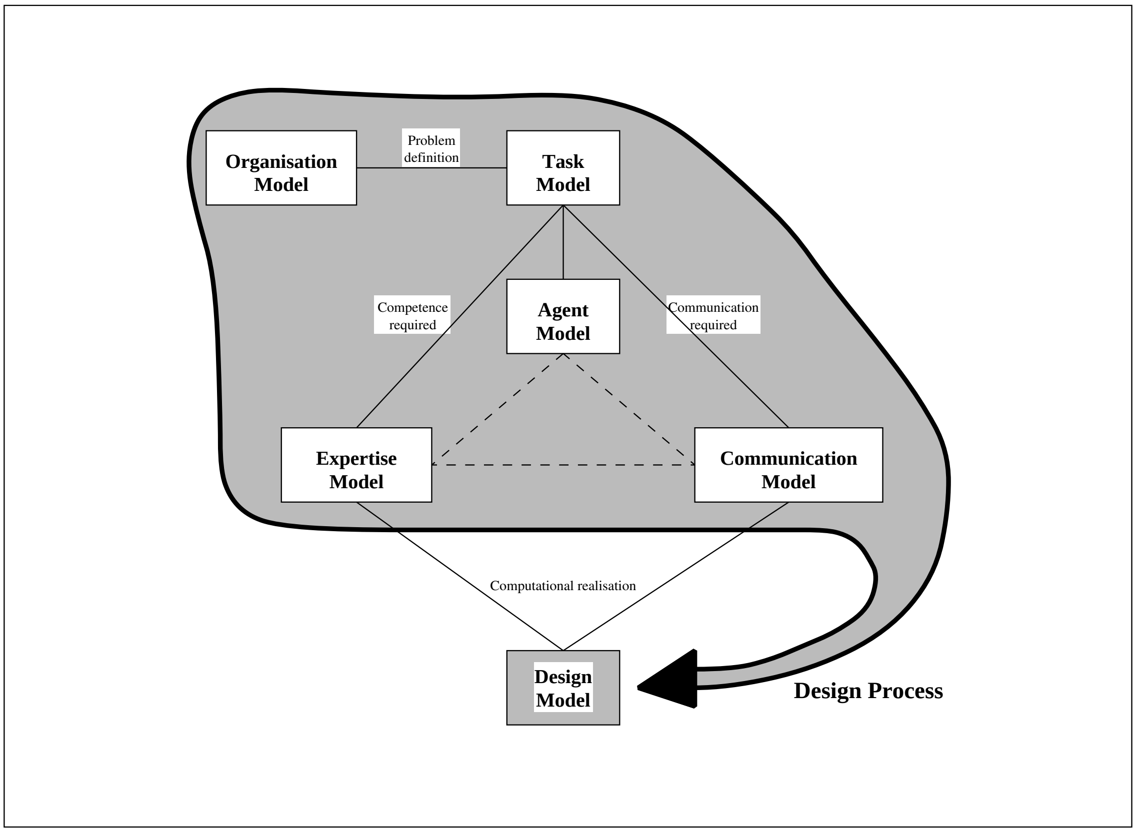

Figure 5.1: *The design process takes the organisation model, task model, agent model, expertise model and communication model as input and produces and output according to the design model framework. The lines indicate direct dependencies between elements of the models.*

5.2 Description of the input

Chapter 1 describes the requirements analysis on each of the *Common*Kads input models. Using these requirements as an input for the design process requires to represent them in a suitable format for the constraint satisfaction system. As an example we identified the following hierarchical structure of variables and domains Fig. 5.2.

This structure is implemented in the constraint satisfaction system and gets extended and modified depending upon requirements analysis done in the *Common*Kads models and/or via adding formalised design guidelines to the constraint satisfaction system.

5.3 Required Output

In principle the design of a KBS should provide a programmer enough information to implement this KBS. We know that for further development, maintenance and validation it is valuable to know why a design looks like it is and how it was constructed. Therefore

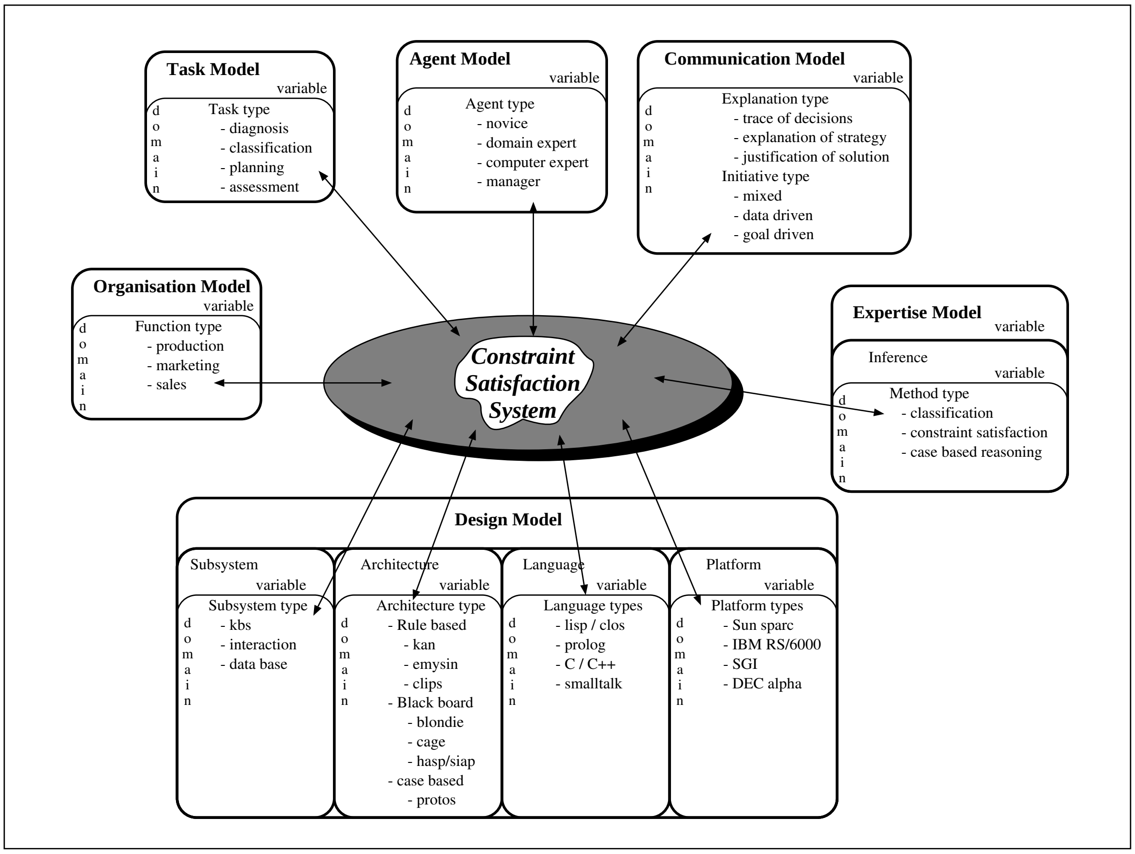

Figure 5.2: *The arrows indicate that analysis requirements are formulated as constraints to the constraint configuration system and visa versa.*

the output of the design process should be more that just a KBS design according to the format of the *Common*Kads design model Fig. 5.3.

We will enrich this design with its rational, a record of taken design decisions and explanation of why certain decisions where taken.
To realise this enrichment we need extra information about:

- *why* a component is included in the final design.
- *why* the design model is structured as it is in *Common*Kads.

The first item is completely covered and explained in the chapter about design guidelines. The second item is related to design principles.

5.3.1 Design Principles

Design Principles contain established knowledge about the design of an expert system. They emerged from people who have worked in the field of KBS design over the last ten to fifteen years [Barr et al., 1989]. A number of these design principles will be formulated

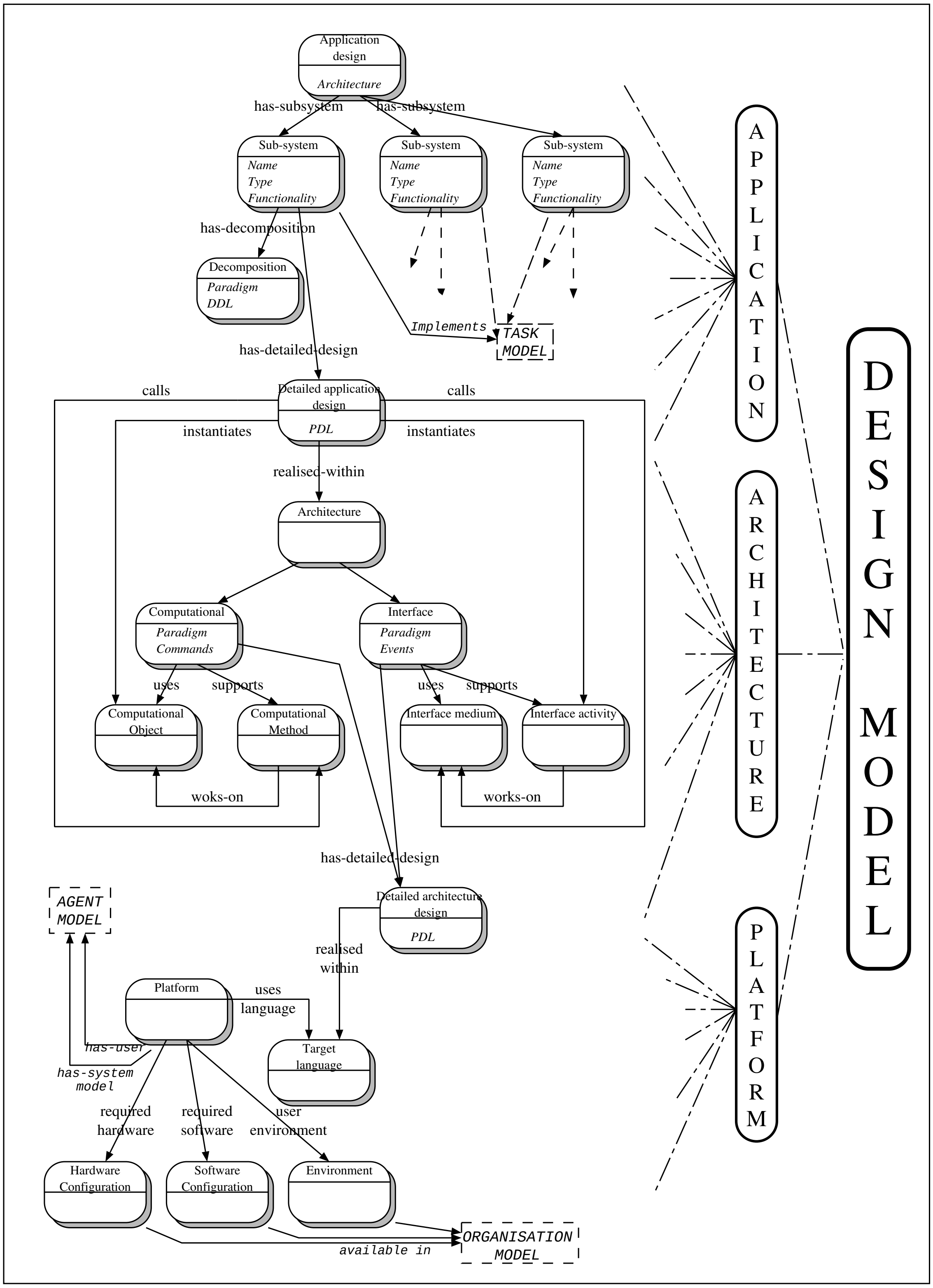

Figure 5.3: *The format of the* Common*Kads design model.*

and we will show how we embedded them into our configuration design approach.

In [Barr et al., 1989, Davis, 1982, McDermott, 1983, Hayes-Roth et al., 1983] a number of design principles are stated:

- *"Represent all knowledge explicitly."*
- *"Keep elements of the knowledge base as independent and modular as possible."*
- *"Separate the knowledge base from the programs that interpret it."*
- *"Use as uniform a representation as possible."*
- *"An object-oriented paradigm offers the most flexibility."*
- *"Exploit redundancy."*

Design principles form the rational behind many knowledge model frameworks, for example the *Common*Kads models or the Expertise model in the COMMET methodology [Steels, 1993]. We used these principles in our configuration approach to identify the necessary variables for the constraint satisfaction system. This results into a structure over the variables which is almost identical to the structure of the analysis models. The advantage is that we now have a transparent relation between constraint relations over variables on one side and relations between analysis and design on the other side.

As an example we show that some of the above principles are realised via identification of *sub-systems* in the design model with respect to *tasks* in the task model. And on a finer scale of granularity the identification of *computational objects* and *computational methods* in the architecture of the design model as *objects that describe* the structure of the knowledge and *methods that use* this structure to apply the inferences. For the constraint system this means that we have to identify variables that correspond with *tasks, inferences, agents, etc.* on the analysis side as well as with *sub-systems, architecture, interface, etc...* on the design model side.

We will now introduce some more principles and identify to which variables they correspond.
An other fundamental design principle is *"Integration of a component that is concerned specifically about the interaction"* [Barr et al., 1989] This can be represented in the interface component of the design model and the requirements on this interface will mainly be derived from the communication model.

"Identify the intended users of the final system", and *"Make system input output appear natural to the user"* [Hayes-Roth et al., 1983]. These are principles related to the agent and communication model and which are preserved in the platform and Interface components of the design model.

"Avoiding assumptions about context and use" [Barr et al., 1989] is an other fundamental principle. We take care of this through explicitly representing all the inter- dependencies

between the design components in the design model (eg. a certain architecture is *realised within* a certain target language which *is used by* a platform that *requires* a certain software and hardware configuration). Together with the idea of using design guidelines to index instances of these components this reduces the risk of making wrong assumptions about *context* and *use* and therefore increases the reusability of the design components.

In general design principles are more related to the overall design methodology and less to the actual design process of one case. The principles are the foundation of the structure of the design model in the design methodology. They can be used as background knowledge for the production of a design rational. Making these design guidelines explicit provides the opportunity to use them as documentation in the explanation report of why a certain design component is represented in a certain placeholder in the *Common*Kads design model template.

5.4 Configuration design process

We distinguish three phases in the proposed configuration design process. Constraint identification, design configuration, design evaluation. These phases are traversed in a cyclic fashion.

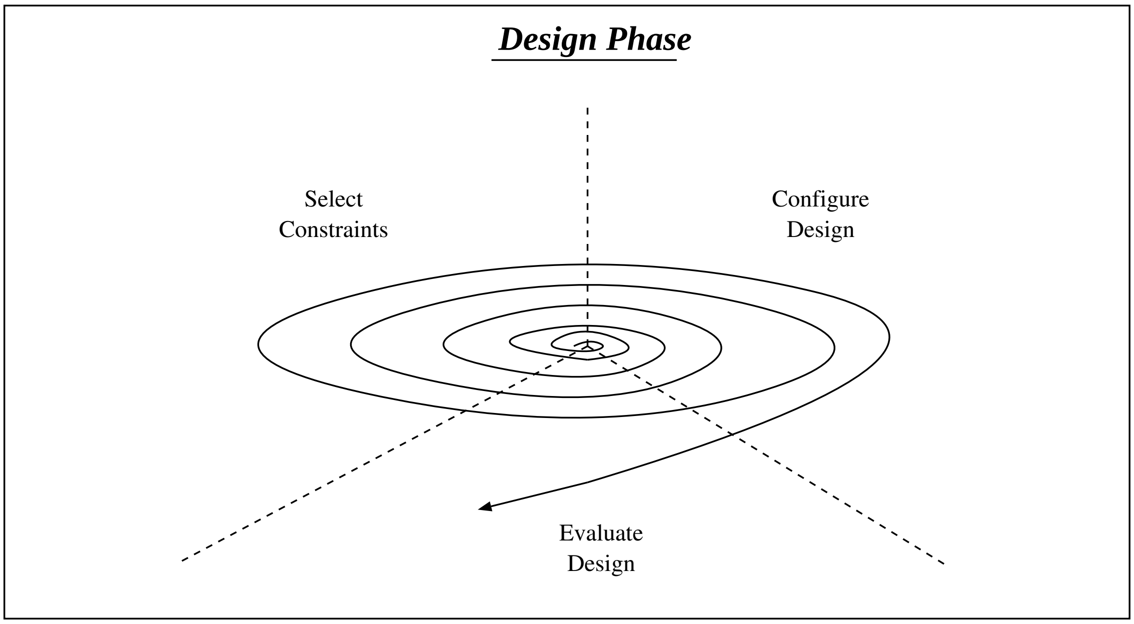

Figure 5.4: *The design phase is a spiral development process consisting of constraint identification, design configuration and design evaluation.*

A similar three phase cyclic approach is also done in the design phase of the MIKE knowledge engineering process [Angele et al., 1993, Landes and Stuber, 1994]. If we look in the *Common*Kads library we will find that this design process matches on the innovative model of design (MR2) [Bernaras, 1993]Fig. 5.5.

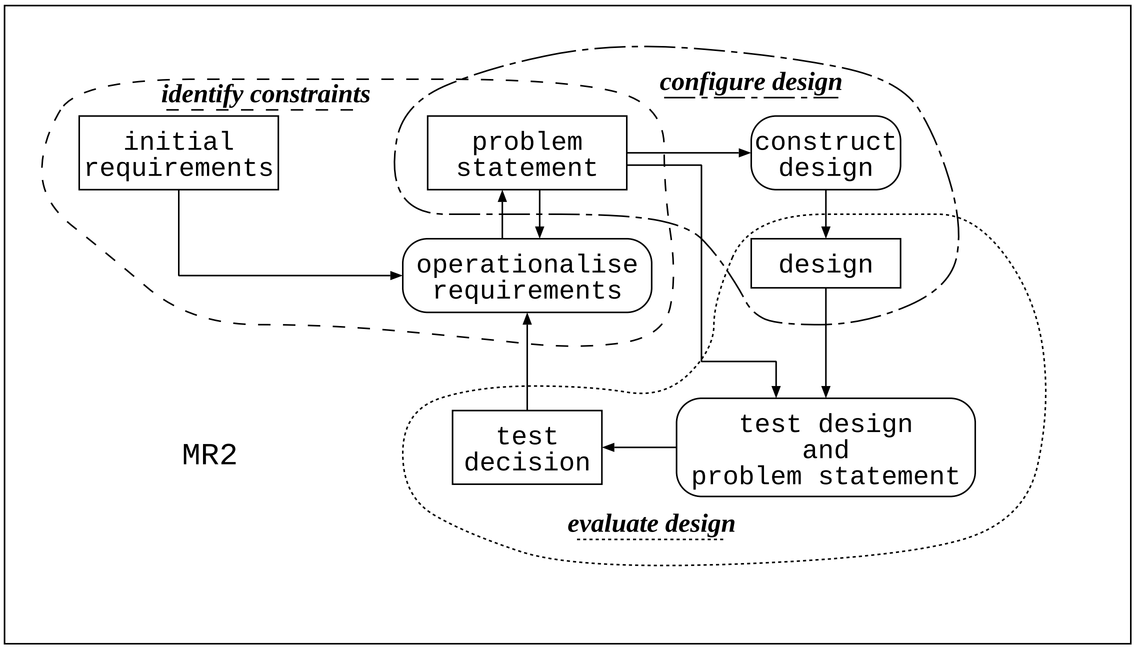

Figure 5.5: *Mapping the three design phases on the* Common*Kads MR2 innovative design inference structure.*

Identification of constraints : Identification of constraints is operationalising the initial set of requirement into a problem statement in terms of constraints. The operationalisation itself is not directly supported. Only a set of already formulated constraints can be provided. The designer can select from this set of constraints. Guidance for the selection process can be provided via explanation about the use of a constraint and descriptions that put relations between the constraints and the initial needs and requirements.

Configuration of the design : The configuration problem is situated in an area where the solution consists of a construction, using elementary reusable design components, which satisfy as good as possible a number of pre-specified constraints. This mechanism of constraint satisfaction is explained in chapter 3.

The evaluation phase : The constructed design has to be evaluated. In the simplest case this can be done by just looking at the configured design. However we want to give some more support to the designer and what we actually want is to give feedback to the selected constraints. This makes the circle round. The process can then start again and becomes a spiral development process towards the aimed goal state of the design model.
Support for the feedback to the original set of selected constraints is completely embedded in the design configuration process itself. It is done via explicit links (provided via design guidelines and principles) between design components and constraints. Extra constraints that are introduced via the configuration of the design are identified during the evaluation process. The system takes care of not violating

the original set of constraints but the designer has to check if the new constraints do not violate the requirements that the final design should satisfy. If the evaluation points out failures then the taken design decisions and the configured design should be revised. If the evaluation was satisfactory then the original set of constraints can be extended with the new generated constraints and be used as the basis for a new phase of constraint identification.

5.5 Embedding

In this section we show how the proposed configuration design process can be embedded in the overall ideas behind the *Common*Kads methodology.

5.5.1 Modelling states

Developing a model is seen as pushing it through a sequence of modelling states [Van de Velde et al., 1993]. A modelling state should contain all the desired behaviour that you expect that its instance will produce. It is a specification of the things you hope to be true for its instance. This specification can be in terms of qualities, requirements, assumptions, task features, characteristics, criteria, ...
The general idea is that modelling states will be used as a guidance throughout the configuration design process. They act as intermediate goals that we want to achieve during the design process. This makes it possible to evaluate the configured design during the design process. It provides the benefit of incremental validation. Validation becomes now really a part of the design process and it will not be suspended until the end of the design phase.

5.5.2 Risk driven

Knowledge based system development in *Common*Kads is controlled by an management cycle that shows the types and sequencing of activities [Taylor, 1992]. One of the basic stages is assessing risks. Risks are areas of uncertainty that could jeopardise the project. This configuration design process fits completely into this risk driven approach. Risks can be taken and also immediately be identified during the design. Each configured partial design can be inspected with respect to its own risk. When a designer finds out that a partial configured design includes a high risk, according to the risk driven approach he can elaborate this design component first. Instead of resulting in an overall design like in a waterfall approach, this design process has the benefit of partially doing some reconfiguration and reducing the risks before the complete end product is achieved. The risks that where taken during the configuration design process can be documented in the design rational or design history. See the Design model document for a list of risks and associated guidelines [Van de Velde, 1993a].

Chapter 6

A Configuration Design Tool

Ronald Schrooten

A configuration design tool is implemented to test our theoretical ideas. Specifications for configuration design are made explicit. The design and use of this tool is described. At the end an evaluation of the prototype is given.

6.1 Introduction

Before describing the functionality of the tool we first will situate it according to other existing tools. This identification and classification of the tool will tell us something about the range of functionalities it should provide.
According to Forte and McCulley [Forte and McCulley, 1991] tools can be classified into two main classes: horizontal and vertical tools. Horizontal tools are used throughout the entire software process (like the Kads Workbench). Vertical tools are used in a specific phase of the life-cycle or activity. The configuration design tool can be classified as a vertical tool which is restricted to translating analysis into the design of a KBS implementation. It is useful to know this since it narrows the application domain of the tool. In this case a small restricted part in the overall development process of a KBS˙
Within a recently developed classification scheme for CASE technology [Fuggetta, 1993] we can situate our tool in the category of *Configuration management tools*. This tells us for instance that the tool should be able to do *item identification*, *adaptation of items to construct a configuration* and *control on the impact of changes on items*. Items are in this case the reusable design components. The identification is done through the use of guidelines, adaptation is provided via the *Common*KADS design model framework and the control on impact is realised via the design guidelines as well as by the internal relationes in the design model framework.

6.2 High Level Design

Application Configuration Design

Name: Configuration Design Tool

Type: Knowledge based

Functionality: The configuration design tool is a design aid tool that provides computer support for finding and structuring the software design of a to be build KBS. To do this, it interlinks reusable design building blocks which come from a library and provides a convenient graphical interface. The reusable designs are indexed through the use of design guidelines. A reusable design component is selected by means of a constraint network that is based on these design guidelines. The design guidelines form the back-bone knowledge of the complete configuration design process. It is the constraint satisfaction algorithm that will use them as heuristics in order to solve the problem.

The tool is not an automatic design configurer but a configuration *support* tool. This means that interaction with the KBS designer is necessary. Therefore we have chosen for a mixed initiative approach. With this we allow the user to interrupt the flow of control whenever he does not agree with the computed deductions. It gives him the opportunity to change design decisions whenever he thinks this is necessary. It is the tool who has to be adaptive to these changes and reconfigure its own reasoning process. The tool also has to be able to do something at any time the user requests this. In other words a user must be able to mediate between two extreme cases.

1. The user configures his design without making any use of the reasoning capabilities of the tool (i.e. only using the graphical interface as a support for visualisation).
2. The user relies completely on the reasoning aspect of the tool and only has to express his approval for suggested solutions.

Functional requirements.

- Support a knowledge engineer in the design of a KBS.
  Make use of reusable design components
- Visualise and construct a design in terms of the *Common*Kads design model.
- Make explicit use of stated design knowledge (in terms of guidelines)
- Be able to explain the solution in terms of these guidelines together with the design decisions the user has made.

Decomposition: A functional task decomposition of the tool.

The top task "configuration design" is the task that the tool should support. It is decomposed into the following sub-tasks.

Collect analysis features. This task supports the acquisition of possible recognisable analysis features. They are generated out of the available guidelines and are stored in a library from which the user can select them.

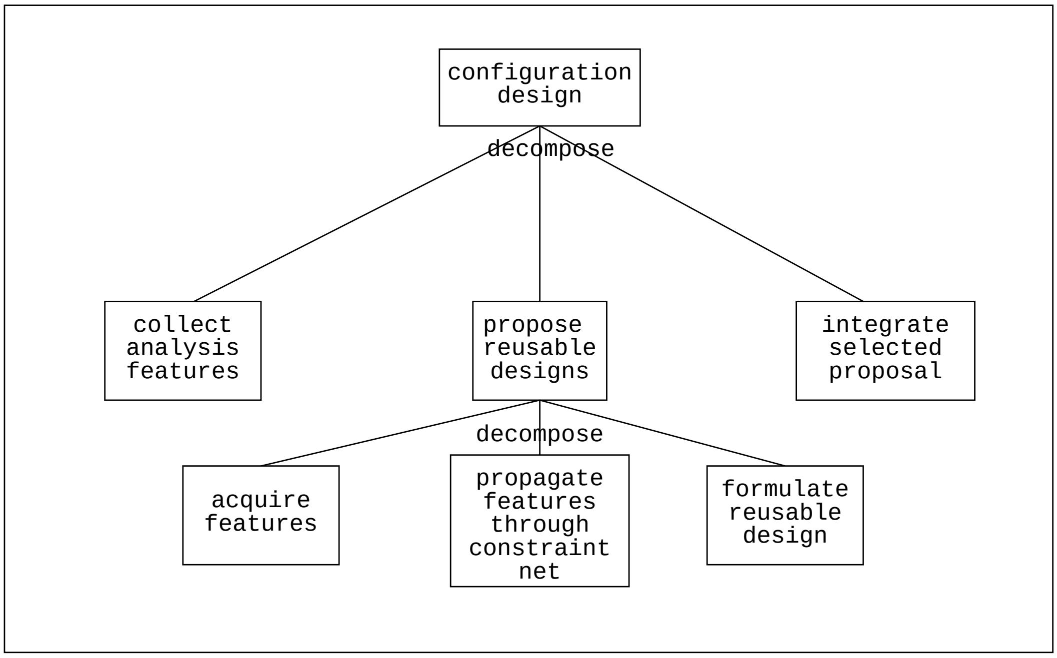

Figure 6.1: *Task decomposition of the configuration design tool.*

Propose reusable designs. This task has the goal to generate all possible reusable design component according to the given analysis features. It is further decomposed into the following three sub-tasks.

Acquire features. Select the given analysis features, put them in the right format and feed them to the constraint system.

Propagate features through constraint net. Activate the constraint satisfaction system.

Formulate reusable design. Retrieve a proposed design component from the constraint satisfaction system. Generate all the features that are attached to this component and put them in a format according to the *Common*Kads models.

Integrate selected proposal . Let the user select one of the proposed designs and integrate it in the existing design situation.

Architecture: Contax shell [Meyer, 1992] extended with X-windows interface calls.

Computational architecture: simple constraint satisfaction architecture.

Computational objects: guidelines formulated as constraints.

Computational method: forward constraint propagation.

Interface: Human-Computer

Interface paradigm: window based,

Interface medium: X-windows, motif style menus, buttons and popup windows.

Interface activities: mixed initiative, keyboard and mouse interactions.

Platform: sun spark workstation.

Target language: Harlequin lisp with integration of CLX (Common Lisp X interface) and CLUE (Common Lisp User Interface Environment).

6.3 System Interaction & Use

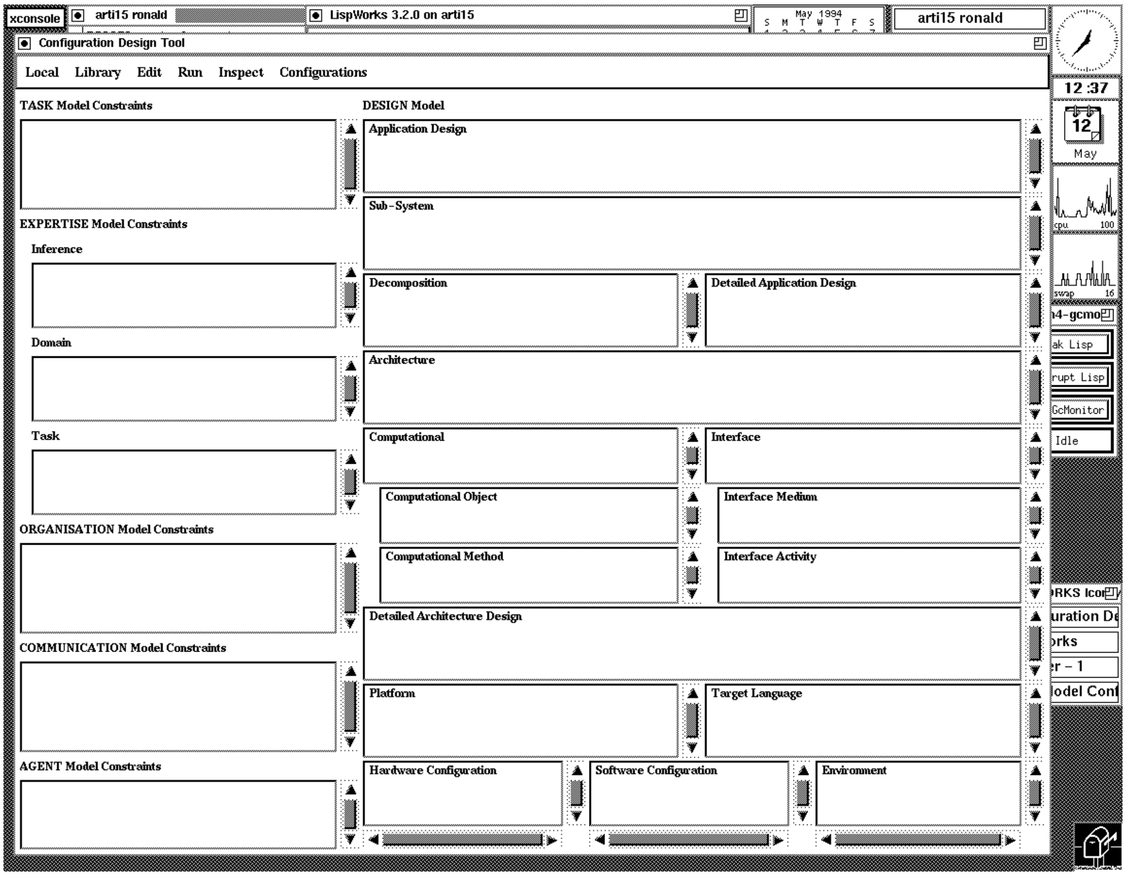

Figure 6.2: *Start up screen of the tool. On the left side are the KADS analysis models. The right side contains a detailed representation of the design model.*

Summary of the menu entries.

Library: The library menu gives the user the possibility to select recognisable analyses features which are related with a reusable design component via at least one design guideline. The features are extracted form the constraint net and provided to the user in a separate window for each type of analysis models Fig. 6.3.

Edit: Provides the necessary edit functionalities like inserting, deleting or changing instances of the tool.

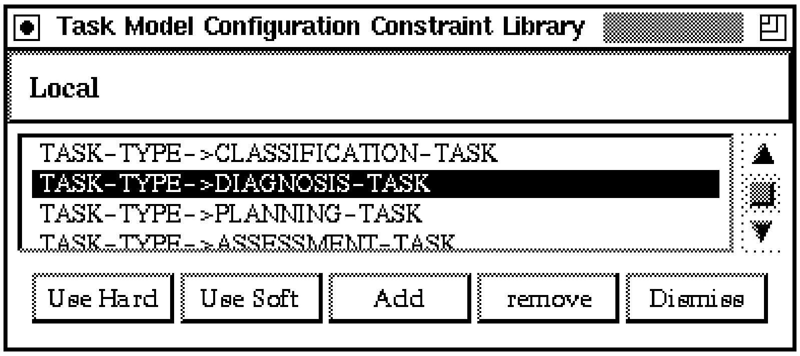

Figure 6.3: *Selection of Task analysis features. Select a feature and use it as a hard or soft constraint for the configuration of the design.*

Run: has two sub-menus.

- **Make Configuration:** Feeds all the filled in constraints to the constraint propagation net and returns all the possible design configurations. Fig. 6.4.
- **Reset Configuration:** Resets the constraint configuration system i.e. the constraint propagation system will start from scratch the next time you call it.

Inspect: Includes four menu entries. The first three (**Domain concepts, constraints, configuration**) activate tools that make it possible for the user to inspect the state the underlying constraint satisfaction network. In the domain concepts window all the implemented domains are displayed Fig. 6.5. The other two tools are harlequin lispworks inspectors with which you can browse through all the interesting configuration and constraint variables.

The fourth menu entry activates a monitoring window. Fig. 6.6. The configuration design tool monitors itself and reports all the meaningful actions that were taken by the user or the tool itself. This monitoring report can be made visible via the monitoring window. This monitor behaves like a text editor and extra information can be added by the user and printed out on paper or saved to a disk. Afterwards this report can serve as a rational for the constructed design model.

Configurations: Contains a entry **Work Sheet** which gives you the ability to go back to the situation you had before you generated configurations (i.e. some sort of *undo*). If there are possible configurations available then a second entry becomes visible. This entry is called **Show Configurations** and provides a window from which the user can see all possible configurations and from which he can include one of these configurations into the work sheet at hand.

6.4 Evaluation of the tool

For the visualisation of the configuration design process we have chosen to represent all the *Common*Kads models in one framework. On first sight this may look confusing but we found out that once you get used to this interface it resulted in the best way to overlook everything that happens during the configuration design process. In a next version also

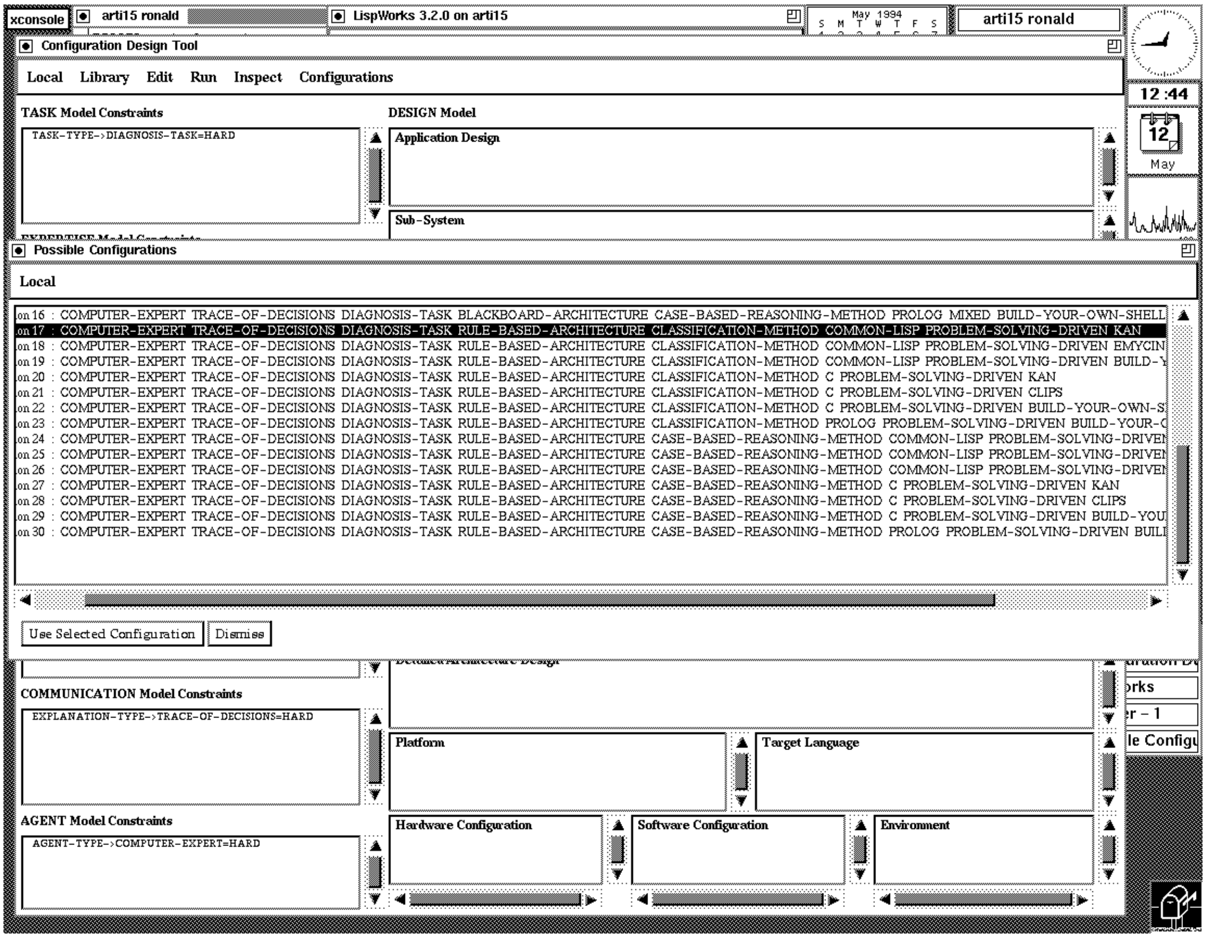

Figure 6.4: *The configuration tool after running a proposal for possible configurations.*

the ability to represent and inspect each model separately should be implemented. Also an extra functionality to aid the process of entering or selecting these requirements would be nice.

While testing we noticed that this environment should also be used to add new design configuration knowledge and new design components. At the moment this is only possible by writing the configurations yourself in the language that the Contax shell provides. In the successor of this tool it should be possible to add this knowledge and extend the tool with new components via the simple graphical interface of the tool itself.

Explanation of why a certain component is selected by the configuration system is very poor at the moment. Instead of browsing through the internal of the configuration system or looking up some information in the monitoring window we propose that the user should be able to retrieve this information in a much more easy way. For example, there should be an explicit explain button that pops up a window with a textual explanation of why a certain design component was chosen and what the influence was of this component is on the rest of the *Common*Kads model. All this information can be retrieved from the system via the implemented design guidelines.

Selection of proposed configurations is not supported by the system. In order to give this

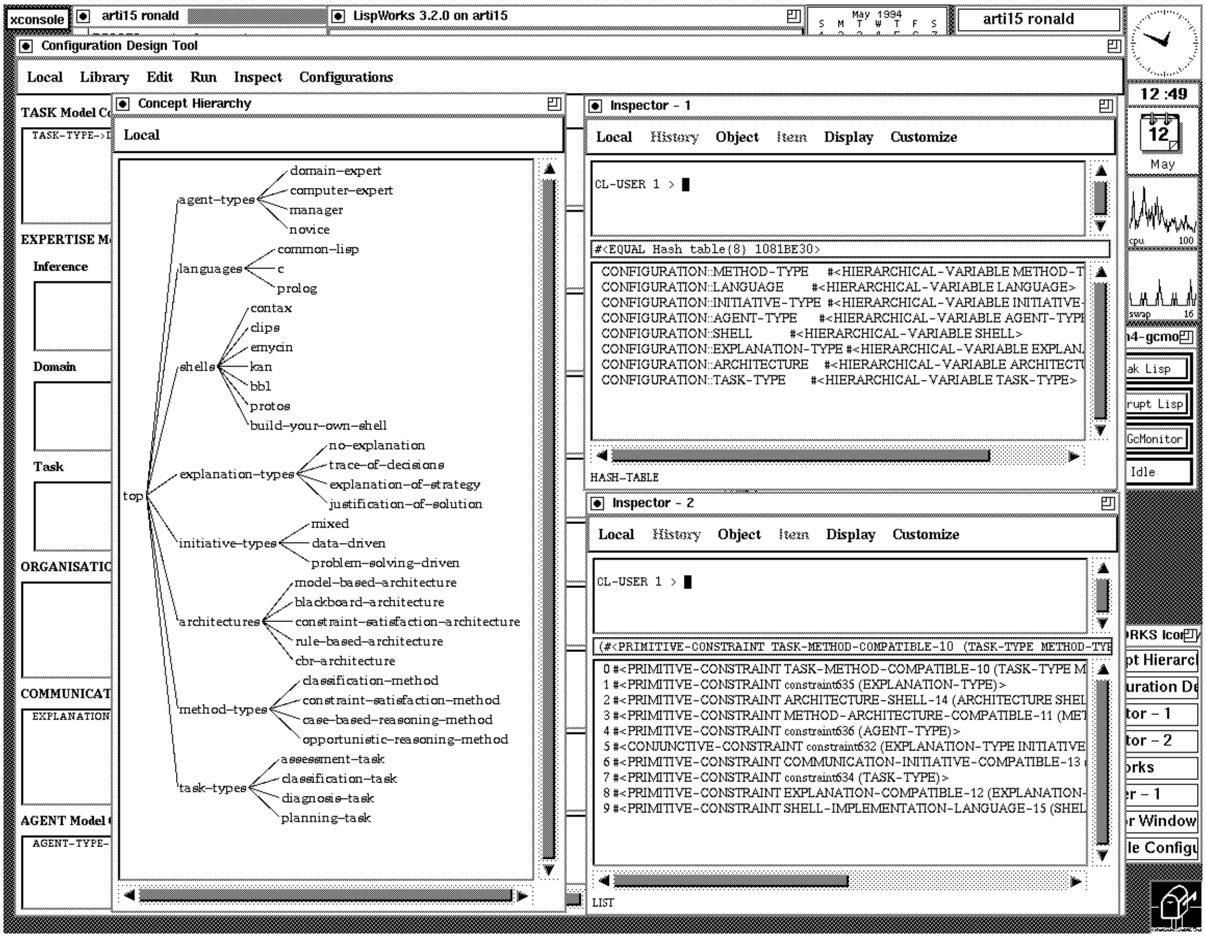

Figure 6.5: *Inspector tools.*

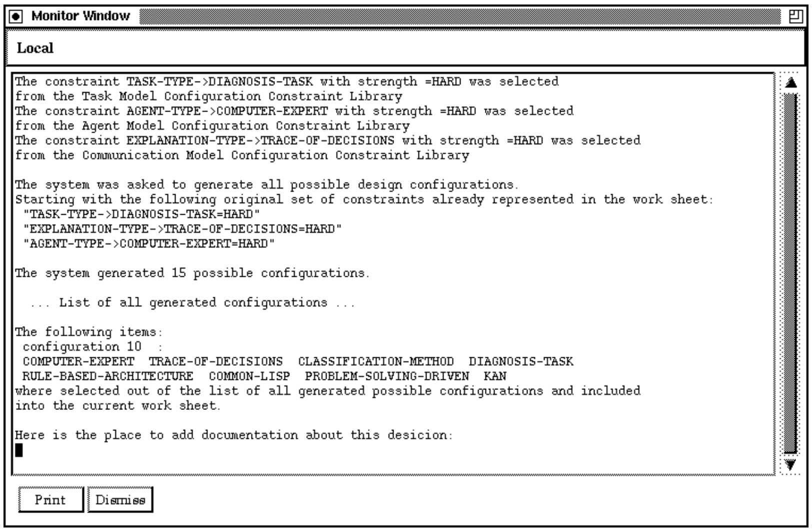

Figure 6.6: *The Monitoring window.*

support to a user there should be worked out some kind of quality criteria for the selection and configuration of the design components as well as for the integration of a configured design into the existing analysis situation. This quality metrics would make it possible for the system to advise a certain solution above an other one.

6.5 Functional Specifications

Building and evaluating the configuration design tool showed us some extra specifications. The following overview list summarises the functional specifications for the to be build configuration support environment.

List of Functional Specifications:

High level requirements: These are general functionalities required from the system and the main reason why such a system should be build.

- Support the process of building a KBS design conform with the *Common*Kads methodology.
- Use information that is retrieved via the analysis done in the *Common*Kads models. This should not exclude that extra design specific knowledge which is not be represented in one of the *Common*Kads models may not be used.
- Configure a design by using predefined reusable design components.
- Represent the solution in terms of the *Common*Kads design model.
- The system should not result into an automatic analysis to design compiler. It should be a support and aid tool for a human designer. Interaction with this user is important and the interface must be user friendly.
- The granularity of the design is related to the modelling state of the design model. It is not required to identify for each component the modelling state to which it belongs. The modelling state is determined by the overall completeness of the design. (i.e. Whether an architecture is *identified* or *implemented* can say something about the model state of the design, but it is not the task of the tool to determine this state or to take care of the state while it is trying to selecting an architectural component.)

Input and background assumptions: Starting assumptions that are required but for which the system is not responsible.

- Constraints can be identified and formulated through analyses of the *Common*Kads models.
- Configurable and reusable design components are available. They can be retrieved from a library [Van de Velde, 1993b], or via analysis of literature [Bahrami, 1988, Barr et al., 1989, Hayes-Roth et al., 1983, Breuker and Van de Velde, 1993, Terpstra and Schrooten, 1993] and practise. A direct link to the *Common*Kads library is not required.

- Design knowledge is available. The *Common*Kads design model framework [Van de Velde et al., 1993] and design guidelines [Bahrami, 1988, Barr et al., 1989, Hayes-Roth et al., 1983, Kline and Dolins, 1989]) are representative examples of such design knowledge.
- It is assumed that the analyses as well as the to be build design fit within the *Common*Kads methodology.

Assumption on use: Global assumptions required from the user of the configuration design support environment.

- The user is familiar with the *Common*Kads methodology.
- The user is aware of the notion of constraints.
- The user is familiar with the aspects of designing KBS specific software.

Computational specifications: Reasoning capabilities and performance of the tool.

- Exhaustive and explicit use of formalised design guidelines.
- Formalisation of the design guidelines in terms of constraints.
- Use design and configuration constraints for indexing and identification of reusable design components.
- Constraint satisfaction via propagation of constraints in an hierarchical structure together with backtracking provides enough computational time performance. Additional optimisation techniques are always encouraged.

Implementation specifications: Requirements and needs related to the implementation of the support environment.

- Functional requirements as specified in this document should be satisfied.
- The purpose is to build a tool for demonstration of the theory. The impact of this on the implementation of the tool is that it should be crash secure under normal use. The correctness of the solutions should be of the same order as the correctness of the implemented heuristic knowledge (i.e. the design guidelines).
- The constraint satisfaction system should be bug free and completely reliable.
- Explicit interface calls and actions may cause no error.
- Validation of the solutions is not within the scope of the implementation.
- Execution is not time critical. Optimisation of the code is not required. A prototyping language is preferred over a fast execution language as long as the response time to the user is acceptable. It might be possible that speed optimisation is necessary for the constraint satisfaction algorithm.
- Applying the convention of a good programming style (be specific, use abstractions, be concise, don't be obscure, be consistent, document your code, don't be lazy, ...) are requested for the sake of maintainability. No other special functions are required for maintenance of the code.
- Portability of the code is not required since the system is for testing theoretical ideas and demonstration purpose only.

Global system interaction: The overall interaction, use, and communication between the system and the user.

- A human-computer interface with a window based paradigm.
- Mixed initiative.
  Mediate between two extreme cases.
  1. The user configures his design without making any use of the reasoning capabilities of the tool (i.e. only using the graphical interface as a support for visualisation).
  2. The user relies completely on the reasoning aspect of the tool and only has to express his approval over suggested solutions.
- The user should be able to interrupt the flow of control whenever he does not agree with the computed deductions.
- Interaction with the system should be easy. (Mouse, button and menu support instead of complicated key combinations.)
- A common look-and-feel should be maintained for every instance of the system.

Explanation: Specification of the explanation facilities that the system should provide.

- Be able to explain the complete solution in terms of design guidelines together with the design decisions the user has made.
- An easy explanation facility that tells the user why an certain design component was chosen by the configuration algorithm.
- Determining what the impact is of selecting a certain design component should be aided by the system. This could be done via clicking on a design component or even an analysis constraint. The system should highlight all the involved constraints in the corresponding *Common*Kads models. A textual explanation is not explicit required because highlighting items in the interface results in a faster information sharing with the designer. On the other hand a textual explanation can be more clear and exhaustive but the risk is that the designer will not take the time to read all the information. This is because in most cases he just wanted to have a quick view on the impact of selecting a component and not a complete study of configuring this component with the rest of the existing situation.
- The explanation of the end solution in terms of the decisions that where taken during the design configuration process should also be able to serve as a rational for the constructed design.
- It should be possible for the designer to add explanation related information to the system. For example information about why a certain constraint or design component was entered. This information will then be included in the explanation reports that are generated by the system. Editing the generated explanation reports should also be possible during the design process.
- It is required that the explanation reports can be saved on to a disk or directly printed out on paper.

6.6 Conclusions

Conclusions from this study with respect to the specification of a configuration design environment.

- First the results of modelling the task, expertise, agent, communication and organisation should be acquired and represented in terms of requirements that can be seen as constraints on the to be build design. For each *Common*Kads model we identified some of these requirements.
- Configuration via constraint satisfaction can be used as an underlying theory for the reasoning mechanism that a system would need to be able to do configuration design. This theory is worked out in detail and compared with others.
- The knowledge that the system needs can be introduced via the use of design guidelines. We formulated examples of these guidelines and implemented them in a constraint satisfaction system. We also identified some advantages and disadvantages of this approach.
- Reusable design components should be described and identified via the use of design guidelines. We did this as an example for a blackboard and a rule based architecture.
- The process of design via configuration and constraint satisfaction is specified. We described where and how it should be used in the *Common*Kads methodology.
- While analysing and building a prototype configuration design tool we identified a number of specifications. Most of them can be found back in the *functionality* slot of the High Level Design of the tool.
- During evaluation of the tool we found out some extra useful functionalities. Applying these functionalities in a next generation of the tool will result in a more complete design configuration support environment. We also proposed possible ways to realise these extra specifications.

Bibliography

[Angele et al., 1993] Angele, J., Fensel, D., Landes, D., Neurbert, S., and Studer, R. (1993). Model-based and incremental knowledge engineering: The mike approach. In Cuena, J. (Ed.). , *Knowledge Oriented Software Design*, 139–168. IFIP Transactions A-27, Elsevier, Amsterdam.

[Bahrami, 1988] Bahrami, A. (1988). *Designing Artificial Intelligence Based Software*. Sigma Press, South Oak Lane, Wilslow, SK9 6AR, England.

[Barr et al., 1989] Barr, A., Cohen, P., and Feigenbaum, E. (1989). *The handbook of artificial intelligence*, volume IV. Addison-Wesley Publishing Company, inc.

[Bernaras, 1993] Bernaras, A. (1993). Models of Design for the CommonKADS Library. KADS-II/M2.3/VUB/DD//1.0, VUB.

[Boehm, 1988] Boehm, B. W. (1988). A spiral model of software development and enhancement. *IEEE Computer*, 61–72.

[Breuker and Van de Velde, 1993] Breuker, J. A. and Van de Velde, W. (1993). The commonkads library. Technical Report ESPRIT Project P5248 KADS-II/M2/VUB/TR/054/1.1, The KADS-II Consortium.

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

[Clancey, 1985] Clancey, W. J. (1985). Heuristic classification. *Artificial Intelligence*, 27, 289–350.

[Davis, 1982] Davis, R. (1982). Expert systems: where are we? and where do we go from here? AI Memo 665, MIT AI-Lab.

[de Greef, 1989] de Greef, P. (1989). Control of cooperation. the design of a knowledge based system that can operate in different modalities. Technical report, University of Amsterdam.

[de Hoog et al., 1993a] de Hoog, R., Benus, B., Metselaar, C., and Vogler, M. (1993a). The common KADS organisation model. ESPRIT Project P5248 KADS II KADS-II/TM6/DM6.2/UvA/041/1.0, Social Science Informatics, University of Amsterdam.

[de Hoog et al., 1993b] de Hoog, R., Martil, R., Wielinga, B., Taylor, R., Bright, C., and Van de Velde, W. (1993b). The CommonKADS model set. ESPRIT Project P5248 KADS-II/M1/DM..1b/UvA/018/5.0, University of Amsterdam, Lloyd's Register, Touche Ross Management Consultants & Free University of Brussels.

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

[Duursma, 1992b] Duursma, C. (1992b). Role limiting methods for the KADS-II conceptual model library. ESPRIT Project P5248 AI-Memo 92/4, AI-Lab, Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium.

[Duursma, 1993] Duursma, C. (1993). Expertise assessment through theory evaluation. Technical Report AI-Memo 93/7, AI-Lab, Vrije Universiteit Brussel, Brussels, Belgium.

[Duursma et al., 1992a] Duursma, C., Aamodt, A., and Van de Velde, W. (1992a). Requirements definition, task analysis and specifications of the CM-editor. Esprit project p5248, AI-Lab, Vrije Universiteit Brussel, Pleinlaan 2 Brussels.

[Duursma et al., 1993] Duursma, C., Cañamero, D., and Schrooten, R. (1993). Model ontology requirements for domain model validation. In Meseguer, P. and Cardenosa, J. (Eds.). , *Proceedings of EUROVAV 93*.

[Duursma et al., 1992b] Duursma, C., Van de Velde, W., Schreiber, G., and Terpstra, P. (1992b). Operationalising knowledge models. Esprit project p5248 kads ii, AI-Lab, Vrije Universiteit Brussel, Pleinlaan 2 Brussels, Belgium.

[Forte and McCulley, 1991] Forte, G. and McCulley, K. (Eds.). (1991). *CASE Outlook: Guide to Products and Services*. CASE Consulting Group, Lake Oswego, Ore.

[Fuggetta, 1993] Fuggetta, A. (1993). A classification of case technology. *Computer*, 26(12), 25–38.

[Gaines, 1994] Gaines, B. (Ed.). (1994). *International Journal of Human - Computer Studies*, volume 40, Number 2. Academic Press.

[Golfinopoulos, 1993] Golfinopoulos, V. (1993). Quality assurance for blackboard systems. Master thesis. Vrije Universiteit Brussel, Faculty of Applied Sciences.

[Green et al., 1986] Green, C., Luckham, D., Balzer, R., Cheatham, T., and Rich, C. (1986). Report on a knowledge-based software assistant. In Rich, C. and Waters, R. C. (Eds.). , *Readings in Artificial Intelligence and Software Engineering*, 377–428. Morgan Kaufmann, San Mateo, CA.

[Hayes-Roth et al., 1983] Hayes-Roth, F., Waterman, D. A., and Lenat, D. B. (1983). *Building Expert Systems*. Addison-Wesley, New York.

[Kline and Dolins, 1989] Kline, P. J. and Dolins, S. B. (1989). *Designing expert systems. A guide to selecting implementation techniques*. Wiley, New York.

[Landes and Stuber, 1994] Landes, D. and Stuber, R. (1994). The design process in mike. In *Proceedings of the 8th Knowledge Acquisition for Knowledge-Based Systems Workshop KAW'94*.

[Mackworth, 1977] Mackworth, A. (1977). On reading sketch maps. In *Proceedings of IJCAI–77*, 598 – 606. International Joint Conference on Artificial Intelligence.

[Mackworth, 1987] Mackworth, A. (1987). Question answering. In Shapiro, S. C. (Ed.). , *Encyclopedia of Artificial Intelligence*, 205–211. John Wiley and Sons, New York.

[McDermott, 1983] McDermott, J. (1983). Extracting knowledge from expert systems. In *IJCAI–83*, 100–107.

[Meyer, 1992] Meyer, M. (1992). Using hierarchical constraint satisfaction for lathe-tool selection in a cim environment. Technical Report RR 92-35, DFKI Gmbh, Postfach 20 80 D-6750 Kaiserslautern, Germany.

[Rademakers, 1991] Rademakers, P. (1991). Task analysis of an equipment assignment problem. AI Memo 91-1, AI-Lab, Vrije Universiteit Brussel, Brussels, Belgium.

[Schreiber, 1992] Schreiber, A. T. (1992). *Pragmatics of the Knowledge Level*. PhD thesis, University of Amsterdam.

[Smithers, 1992] Smithers, T. (1992). Taking eliminative materialism seriously - a methodology for autonomous systems research. AI Memo 92-10, AI-Lab, Vrije Universiteit Brussel, Brussels, Belgium.

[Smithers et al., 1990] Smithers, T., Conkie, A., Doheny, J., Logan, B., Millington, K., and Tang, M. (1990). Design as intelligent behaviour: An AI in design research programme. *Artificial Intelligence in Engineering Journal*, 5(2), 78–109.

[Steels, 1993] Steels, L. (1993). The componential framework and its role in reusability. In Jean-Marc David, J.-P. K. and Simmons, R. (Eds.). , *Second Generation Expert Systems*. Springer Verlag, Berlin.

[Taylor, 1992] Taylor, R. (1992). Risk management in knowledge-based systems development. KADS-II Report KADS-II/T2.1.2/WP/TRMC/003/1.0, Touche Ross.

[Terpstra and Schrooten, 1993] Terpstra, P. and Schrooten, R. (1993). Kads specific design decisions and their notation. D.M.7.2a ESPRIT Project P5248 KADS-II/M7/UvA/043/1.1, The KADS-II Consortium.

[Van de Velde, 1993a] Van de Velde, W. (Ed.). (1993a). *Design Model and Process*. Number KADS-II/M7/VUB/RR/064/1.1 in ESPRIT P5248. VUB AI-Lab, Brussels, Belgium. with contributions from Cuno Duursma and Guus Schreiber.

[Van de Velde, 1993b] Van de Velde, W. (1993b). Generic library facility: Specification. KADS-II report KADS-II/WPM/TM.8/DD/001/0.2, Artificial Intelligence Laboratory, Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussel.

[Van de Velde et al., 1993] Van de Velde, W., Duursma, C., and Schreiber, G. (1993). Design model and process. Deliverable D.M.7, Version 1.1 ESPRIT Project P5248 KADS-II/M7/VUB/RR/064/1.1, The KADS-II Consortium.

[Vanwelkenhuysen, 1991] Vanwelkenhuysen, J. (1991). Troubleshooting digital printed circuit boards by means of an expert system. In *Proceedings of the 2nd European Test Conference*, Munich, Germany. vde-verlag.

[Vanwelkenhuysen, 1993] Vanwelkenhuysen, J. (1993). *Participative design of industrial knowledge based systems*. PhD thesis, Vrije Universiteit Brussel, Brussels, Belgium. Also as VUB AI-Lab TR-93-1.

[Velthuijsen, 1992] Velthuijsen, H. (1992). *The Nature and Applicability of the Blackboard Architecture*. PhD thesis, Rijksuniversiteit Limburg te Maastricht.

[Waern and Gala, 1993] Waern, A. and Gala, S. (1993). The commonkads agent model. ESPRIT Project P5248 KADS II KADS-II/M4/TR/SICS/004/V.1.0, Swedish Institute of Computer Science.

[Waern et al., 1993] Waern, A., Hook, K., and Gustavsson, R. (1993). The communication model. ESPRIT Project P5248 KADS II KADS-II/M3/SICS/TR/002/V1.0, Swedisch Institute of Computer Science, Stockhom, Sweden.

[Wielinga (Ed.), 1993] Wielinga (Ed.), B. (1993). Expertise model: Model definition document. KADS-II report KADS-II/M2/UvA/026/2.0, Universtiy of Amsterdam, Social Science Informatics, University of Amsterdam. with contributions from Manfred Aben, Hans Akkermans, Cuno Duursma, Olle Olsson, Klas Orsvarn, Guus Schreiber, Walter Van de Velde, Steve Wells and others.

[Yost, 1992] Yost, G. R. (1992). Configuring elevator systems. Technical report, Digital Equipment Corporation, 111 Locke Drive, Marlboro, USA.