CommonKADS: A Comprehensive Methodology for KBS Development

Guus Schreiber, Bob Wielinga, and Robert de Hoog, University of Amsterdam
Hans Akkermans, University of Twente, The Netherlands
Walter Van de Velde, Free University of Brussels

THE AIM OF COMMONKADS IS TO FILL THE NEED FOR A STRUCTURED METHODOLOGY FOR KBS PROJECTS BY CONSTRUCTING A SET OF ENGINEERING MODELS BUILT WITH THE ORGANIZATION AND THE APPLICATION IN MIND.

WHEN THE RESEARCH THAT led to CommonKADS was conceived as part of the European Esprit program in 1983, the AI community as a whole showed little interest in methodological issues. At the time, the prevailing paradigm for building knowledge-based systems was rapid prototyping using special purpose hard- and software, such as LISP machines, expert system shells, and so on. Since then, however, many developers have realized that a structured development approach is just as necessary in knowledge-based systems as it is in conventional software projects. This structured development approach is the aim of CommonKADS.

Traditionally, knowledge engineering was viewed as a process of "extracting" knowledge from a human expert and transferring it to the machine in computational form. Today, knowledge engineering is approached as a *modeling* activity. In the CommonKADS methodology, KBS development entails constructing a set of engineering models of problem solving behavior in its concrete organization and application context. This modeling concerns not only expert knowledge, but also the various characteristics of how that knowledge is embedded and used in the organizational environment. The different models are a means of capturing the different sources and types of *requirements* that play a role in realistic applications. A KBS, then, is a computational realization associated with a collection of these models.

Figure 1 summarizes the suite of models involved in a CommonKADS project. A central model in the CommonKADS methodology is the expertise model, which models the problem solving behavior of an agent in terms of the knowledge that is applied to perform a certain task. Other models capture relevant aspects of reality, such as the task supported by an application; the organizational context; the distribution of tasks over different agents; the agents' capabilities and communication; and the computational system design of the KBS. These are engineering-type models and serve engineering purposes. The models are considered not as "steps along the way," but as independent products in their own right that play an important role during the life cycle of the KBS.

Here, we give a brief overview of the CommonKADS methodology, paying special attention to the expertise modeling — an aspect of KBS development that distinguishes it from other types of software development. We illustrate the CommonKADS approach by showing how aspects of the VT system[1] for elevator design would be modeled (see sidebar, "The VT System" for background).

Project management principles

In CommonKADS, project management and development activities are separated. Project management is represented by a project management activity model that interacts with the development work through model states attached to the CommonKADS models. The development process proceeds in a cyclic, risk-driven way similar to Boehm's spiral model.[2]

0885-9000/94/$4.00 © 1994 IEEE

The VT system

The VT system was originally developed for the Westinghouse company to support the routine design of elevators. The system was developed because processing standard design took too much time, and sales people wanted to serve customers with simple design problems more quickly.

At the time of development, a number of software tools were available, such as a database of elevator components and specialized tools for calculating particular formulas. We selected the VT domain because it is well-known and is used for comparisons in the knowledge engineering community.[1]

References

1. G. Yost, "Configuring Elevator Systems," tech. report, Digital Equipment Corporation, Marlboro, Mass., 1992.

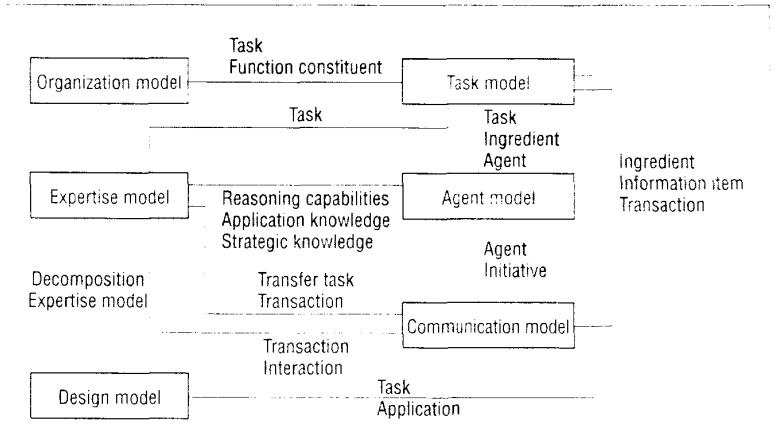

Figure 1. The CommonKADS suite of models. The lines indicate direct dependencies between elements of the models.

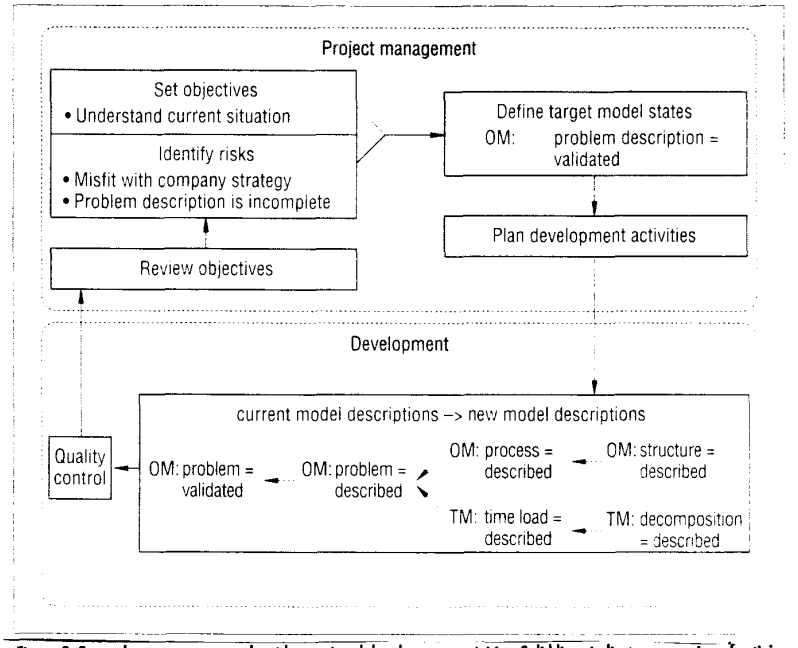

Figure 2. Example management cycle with associated development activities. Solid lines indicate sequencing of activities. Dashed lines describe information dependencies between development activities. OM = organization model; TM = task model.

Whenever possible, parallel development of models is encouraged. Models must be maintained over the life cycle of the product. For each project a specialized "life cycle" can be configured depending on specific project objectives and risks. Finally, control of quality and progress is integrated through regular checking of model states that must be reached in a cycle.

Figure 2 gives a stylized representation of how project management and development work are connected through model states. At the start of a management cycle, objectives for the cycle are defined, and associated risks are identified. From these objectives and risks, a set of model states is derived that must be realized within the cycle. These target model states are projected onto development activities that should result in "filling" elements of the CommonKADS models.

In Figure 2, the target state (validation of the problem description in the organization model) leads to the exploration of a number of additional states that the target state depends on (such as a description of the structure of the organization). These information dependencies are depicted as dashed lines in the figure. CommonKADS provides extensive background information on such model state dependencies. At the end of each development cycle, a check is performed on the quality of the results. The project manager then reviews the results achieved in light of the overall objectives and risks.

A CommonKADS project usually consists of many cycles, with the actual number depending on the planning horizon and the identification of new objectives and risks. Steps within a cycle can be repeated many times. The CommonKADS model set plays a pivotal role in this process. It provides a comprehensive and organized collection of aspects that can be relevant in a KBS project. However, this does not mean that in an actual project all models have to be fully developed; only those model components and states that bear on the project objectives and risks are selected. This allows for a parsimonious approach whenever necessary.

Modeling the KBS environment

Any information system has to function in the context of the overall organization. Information and knowledge systems are minor components within an organization's business processes. A KBS is only one agent among many — human and nonhuman — and carries out only a fraction of the organization's tasks. As a result, it is essential to keep track

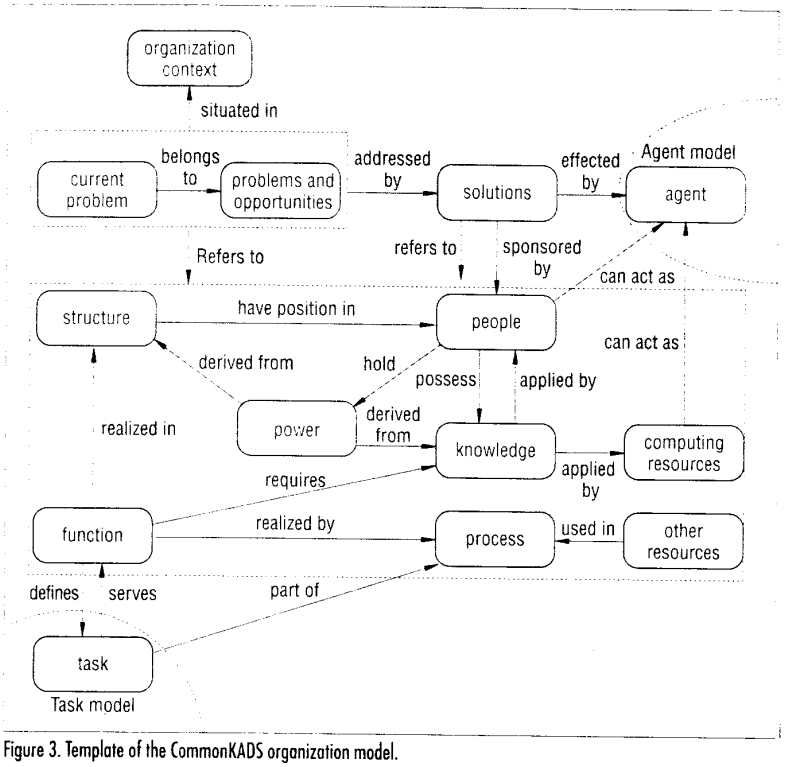

Figure 3. Template of the CommonKADS organization model.

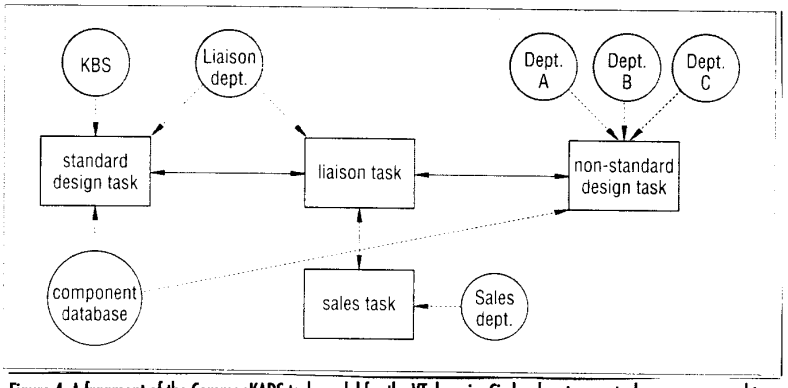

Figure 4. A fragment of the CommonKADS task model for the VT domain. Circles denote agents; boxes correspond to tasks. Solid arrows indicate exchange of information between tasks; dashed arrows assign agents to tasks. If the KBS is jointly assigned with other agents to a task (as is the case for standard design), the communication model will need to contain a specification of how the agents cooperate in performing the task.

of the overall environment in which the KBS has to operate. Many KBS failures have resulted from the lack of concern for social and organizational factors. Yet, many system development methods still focus on the technical aspects only, and provide little support for the analysis of the organizational elements that determine success or failure.

The CommonKADS model set provides four models that are specifically geared to modeling the organizational environment of a KBS: the organization, task, agent, and communication models.

The *organization model* supports the analysis of the major features of an organization to discover problems and opportunities for KBS development, as well as possible effects a KBS could have when fielded. A *template* that defines object and relation types is associated with each model in the model set (see Figure 3). The different components and relations in Figure 3 constitute topics to explore in this process and "stores" for the information obtained.

For example, an organizational analysis of the VT elevator design domain could result in the following (simplified) descriptions of organization model components.

- *Function*: The central organizational function under consideration is design.
- *Structure*: Currently, three departments are carrying out design activities.
- *Computing resources*: A database of elevator components and some specialized computational tools are available.
- *Current problems*: First, the design lead time (currently three weeks) is too long. Second, communication between the three involved departments is cumbersome and time consuming.
- *Solution*: First, a separate group for solving standard design problems will be formed, recruiting members from the existing departments. Second, the three (reduced) departments will act as expert groups for special, nonstandard designs. Third, the new group will act as the liaison with the sales department, and will be supported by a new computational tool: a KBS.

Clearly, this solution affects the organization. Effectively, the design function is divided into two new subfunctions — standard and nonstandard design. The organization structure is adapted accordingly. The KBS forms an addition to the computing resources of the organization and should fit into the current infrastructure. The intended organizational changes will also lead to changes in other aspects, including the distribution of knowledge. Separate variants of the organization model could model both the old and new situations.

The *task model* describes, at a general level, the tasks that are performed or will be performed in the organization where the expert system will be installed. The tasks it covers are those that help realize an organizational function. The task model is represented as a hierarchy of tasks. In addition, aspects like inputs and outputs of tasks, task features, and task requirements can be modeled. The task model also specifies the distribution of tasks over agents.

An agent is an executor of a task. It can be human, computer software, or any other "entity" capable of executing a task. In the *agent*

model, the capabilities of each agent are described. The model can also be used to represent constraints on an agent, such as norms, preferences, and permissions that apply to the agent. For example, a constraint might be an organizational rule: A specific decision-making task should not be performed by a computer. Often, more agents are involved in a task than just a user and a KBS. In the VT case, for example, there is a database of elevator components.

Because several agents are usually involved in a task, it is important to model the communication between agents. This is the purpose of the CommonKADS *communication model.* The transactions here are modeled at a level that is still independent of a computational realization.

Figure 4 illustrates part of the task model for the VT application. The figure shows several agents (five departments, the KBS, and a database) in relation to several tasks. In the new situation, the liaison department handles standard designs with support from the KBS. Liaisons give nonstandard designs to specialized departments, and the design output is routed back to the sales group.

Using these models, a developer can build a project-specific picture of the social context in which a KBS must operate. The organization model supplies the main high-level aspects of the organizational environment, while the task model focuses on a subset of tasks directly related to the problem to be solved. These tasks are allocated to agents characterized through the agent model. Information-exchange acts between agents are detailed in the communication model. Reasoning capabilities required for tasks can be analyzed with the aid of the expertise model explained below. Together, the expertise and communication specifications form the conceptual basis for technical system design.

Modeling expert knowledge

Expertise modeling is a focus point of CommonKADS, and is a specific activity in the type of systems we have targeted. The first-generation knowledge-based systems used one relatively simple *inference engine* working on a knowledge base in a particular representational format, usually production rules. But such a knowledge base hides important properties of the reasoning process and knowledge structure in the application domain.[3] Certain rules, or parts of rules, fulfill particular roles in the reasoning process that remain implicit in such a KBS organization. This implicitness of underlying structures impairs the acquisition and refinement of knowledge for the KBS, as well as hampering the reuse of the system, its explanatory power, and the assessment of its relation with other systems.

During the eighties, the idea of introducing a knowledge-level description was taken on in knowledge-engineering research to solve these problems.[4] A knowledge-level model of a KBS makes the organization of knowledge in the system explicit through elaborate *knowledge typing.* This knowledge typing should provide an implementation-independent description of the role that various knowledge elements play during the system's problem solving process. A knowledge-level model should explain how and why the system carries out a task in a vocabulary understandable to users. The model is thus an important vehicle for communicating about the system, both during development and during system execution.

With respect to knowledge categories, a distinction is often made between *domain knowledge* and *control knowledge.* Domain knowledge is static, and consists of the concepts, relations, and facts that are needed to reason about a certain application domain. We divide control knowledge into two categories: *inference knowledge,* which describes how to use domain knowledge in elementary reasoning steps (*inferences*); and *task knowledge,* which describes how to decompose the top-level reasoning task, and how to impose control on this decomposition.

```
C1: leftplatformedge_lefthoistwaywall =
    openinghoistwayleftspec - carreturnleft

C2: leftplatformedge_lefthoistwaywall >= 8

C3: counterwtovertravel =
    toplandtobeam - (deflsheavep + counterwtfooth + counterwtbuffblockh
    + counterwtbuffheight + counterwtrunby + (counterwtframeh - pitdepth))

C4: counterwtovertravel >= carrunby + (1.5 * (carbufferstroke +6))

C5: IF machine-model is-one-of {"machine28","machine38"}
    THEN 3 >= noofhoistcables >= 6

C6: cwt_to_hoistway_rear >= cwt_ubracket_protrusion + 0.75

C7: IF machine-model = "machine18" AND elevatorspeed = 200
    THEN  machine_efficiency = 0.78
```

Figure 5. Domain knowledge fragments in the VT domain.

Domain knowledge. A CommonKADS description of domain knowledge defines both the *content* and the *structure* of the domain-specific knowledge base in a declarative form. Figure 5 shows some typical fragments of the knowledge base used in the VT domain. The formulas specify dependencies between elevator system parameters. (The examples were derived from the Ontolingua version of the VT knowledge base.[5])

When the formulas are studied in more detail, it becomes clear that there are in fact two types:

(1) Calculation formulas, which can be used to compute the value of a parameter (C1, C3, C7); and
(2) Constraint formulas, which define parameter-value restrictions that should not be violated (C2, C4-6).

Such an underlying structure of domain knowledge elements is represented in CommonKADS through an *ontology.* Figure 6 shows the ontology for the formulas in Figure 5. The notation is part of the CommonKADS Conceptual Modeling Language (CML) (see also the sidebar, "Specification formalisms"). In the figure, the diamonds represent relation types. The calculation is represented as a ternary relation between a formula, a set of parameters playing the role of inputs (represented by the ▷◁ symbol), and a single parameter serving as output. The constraint relation is modeled as a binary relation between a formula and the parameter involved. Modeling complex expressions such as formula types is a typical feature of KBS construction.

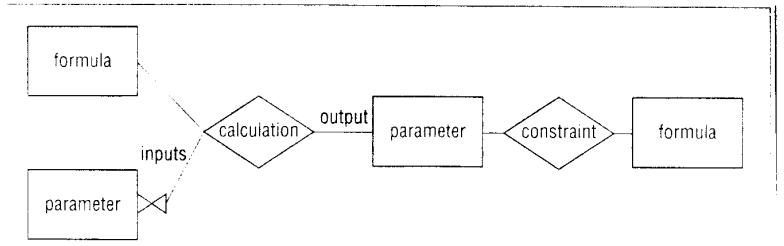

Figure 6. Graphical representation of the structure ("ontology") underlying the knowledge fragments in Figure 5.

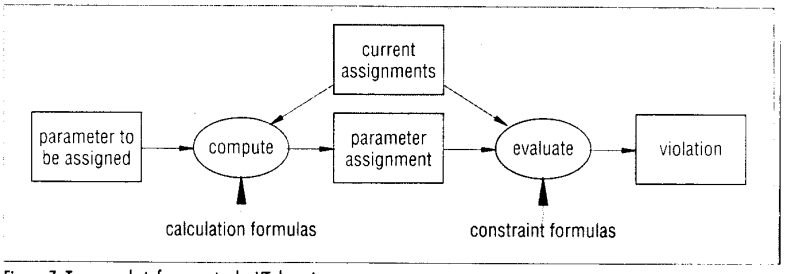

Figure 7. Two sample inferences in the VT domain.

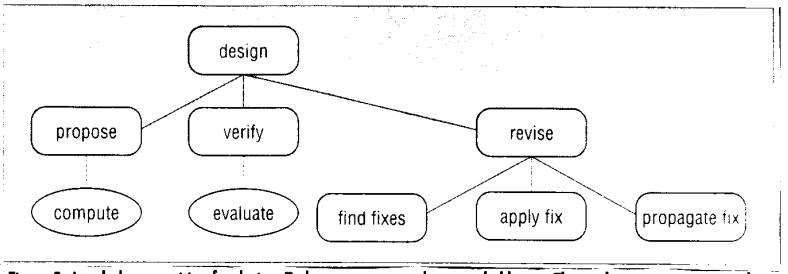

Figure 8. A task decomposition for design. Tasks are represented as rounded boxes. The ovals represent two sample inferences (part of the inference knowledge) that are invoked by leaf tasks.

Inference knowledge. Inference knowledge is modeled in CommonKADS in terms of the operations on domain knowledge (*inferences*) and in terms of *roles*. A role is a label for some class of domain knowledge elements that are used in a particular inference operation. The label indicates the role these elements play in the reasoning process (such as "hypothesis").

Figure 7 shows two inferences present in the VT application. Ovals represent inferences; rectangles denote data elements manipulated by the inference (*dynamic* roles). The double arrow indicates the underlying domain knowledge that is used by the inference (*static* roles). In this case, a *compute* inference computes a value for a parameter, using the calculation formulas in the knowledge base. This new parameter assignment can be used as input for an *evaluate* inference that can produce constraint violations.

Structures like the one in Figure 7 are called *inference structures*. They show the data dependencies between inferences and constrain (but do not define) the flow of control. Also, the inference knowledge is phrased in a domain-independent vocabulary: No VT-specific terms, such as "elevator," are used.

The role of inference knowledge is similar to that of inference rules in classical logic. In logic, an inference rule describes how axioms (domain knowledge) can be combined to derive new information. The sequence or purpose of the inferences is not described in the inference rule, but may be part of a mechanism embodied in a theorem prover. Inferences in CommonKADS can be viewed as *generalizations* of inference rules in logic. The main differences lie in the following features of CommonKADS inferences:

- they operate on restricted parts of domain knowledge;
- they are not necessarily truth preserving; and
- they refer to a computational method that has a specific purpose in problem solving.

An inference specified in the inference knowledge is assumed to be *basic* in the sense that it is fully defined through its name, an input/output specification, and a reference to the domain knowledge that it uses. The computational way in which the inference is carried out is assumed to be irrelevant for the purposes of modeling expertise. From the viewpoint of the expertise model, no control can be exercised on the internal behavior of the inference. The inference is only assumed to be basic with respect to the expertise model. It is very possible that such a basic inference is realized in the actual system through a complex computational technique.

Task knowledge. Task knowledge in CommonKADS is modeled as a hierarchy of tasks. Figure 8 shows a task decomposition for the standard design task based on the propose-and-revise method.[6] This method requires that a design be represented as a set of parameter assignments (parametric design). The leaf nodes in this task hierarchy (such as *propose* and *verify*) invoke particular inferences (such as *compute* and *evaluate*).

A specification of a CommonKADS task is divided into two parts. The *task definition* is a declarative specification of the goal of the task, describing *what* must be achieved. The *task body* specifies a procedure, and prescribes the activities to accomplish the task. The task body describes *how* the goal can be achieved.

In Figure 9, a specification of the top-level task for the VT application is shown. The task definition defines the overall goal of this design task and its I/O. This particular task definition requires that the domain knowledge can be viewed in terms of a set of parameters representing the skeletal design, and a set of constraints that involve these parameters. Design starts off with proposing a design extension (a new parameter value). This value is checked to see if it introduces a constraint violation. If it does, the revise task is invoked with the violated constraint as input. This process is repeated until all parameters in the skeletal design have been assigned a value. If for some reason the propose task or the revise task fails, the overall design task fails.

Expertise modeling: support and reuse

It is important to consider how to support the process of defining the expertise model for a particular application. Like most other methods, CommonKADS provides this support by enabling reusability of previously defined model components. The main difference is that in CommonKADS, the components are *metamodels* of the domain and control knowledge descriptions.

The separation of the domain knowledge and control-related knowledge gives rise to an important question: What are the dependencies between the two parts of the model? While the so-called *interaction problem* states that control knowledge and domain knowledge are highly dependent[7] — one cannot define the domain knowledge without knowing what the task is going to be, and vice versa — early work on KADS was done under the assumption that domain knowledge can be formulated independently from the task.[8]

Domain metamodels: ontology. There is a growing consensus that some interaction between the domain knowledge and the task must exist, but that different types of interaction can be distinguished. In CommonKADS, this is called the *relative interaction hypothesis* — different types of knowledge differ in the degree to which they are dependent on the nature of the task. In CommonKADS, these different knowledge types are explicitly described in a number of *ontologies*.

These ontologies are metamodels describing the model structure of (part of) the domain knowledge. The ontologies can be organized in a multilevel structure, where each level corresponds to a particular type of interaction. Mappings between the layers represent "viewpoints" on the domain knowledge. Multiple mappings of a certain knowledge type can exist, representing multiple viewpoints. Figure 10 shows a graphical representation of ontologies involved in the VT application. The bottom shows two fragments in the knowledge base (taken from Figure 5).

Several ontologies serve as metamodels of the VT knowledge base. The *parametric design ontology* introduces the general notion of *constraint expression* to describe parameter dependencies (among other definitions not shown). This ontology should contain ontological commitments that are required by the parametric design task in general, but are not necessarily sufficient for the method used

```
task parametric-design;
    task-definition
        goal: "find a design that satisfies a set of constraints";
        input:
            skeletal design: "the set of system parameters to which values need to be assigned";
            requirements: "the set of initial parameter/value pairs";
        output:
            design: "final set of assigned parameters";
    task-body
        type: composite;
        sub-tasks: init, propose, verify, revise;
        additional-roles:
            extended-design: "current set of assigned parameters";
            design-extension: "proposed new element of the extended model";
            violation: "violated constraint";
        control-structure:
            parametric-design(skeletal-design + requirements → design) =
                init(requirements → extended-design)
                REPEAT
                    propose(skeletal-design + extended-design → design-extension)
                    extended-design := design-extension ∪ extended-design
                    verify(design-extension + extended-design → violation)
                    IF "some violation"
                    THEN revise(extended-design + violation → extended-design)
                UNTIL "a value has been assigned to all parameters in the skeletal-design"
                    design := extended-design;
end
```

Figure 9. Sample task specification of the top level task for the VT application. In the control structure, arrows are used to distinguish input and output. The statements in *italics* describe actions whose representational details have to be decided during KBS design.

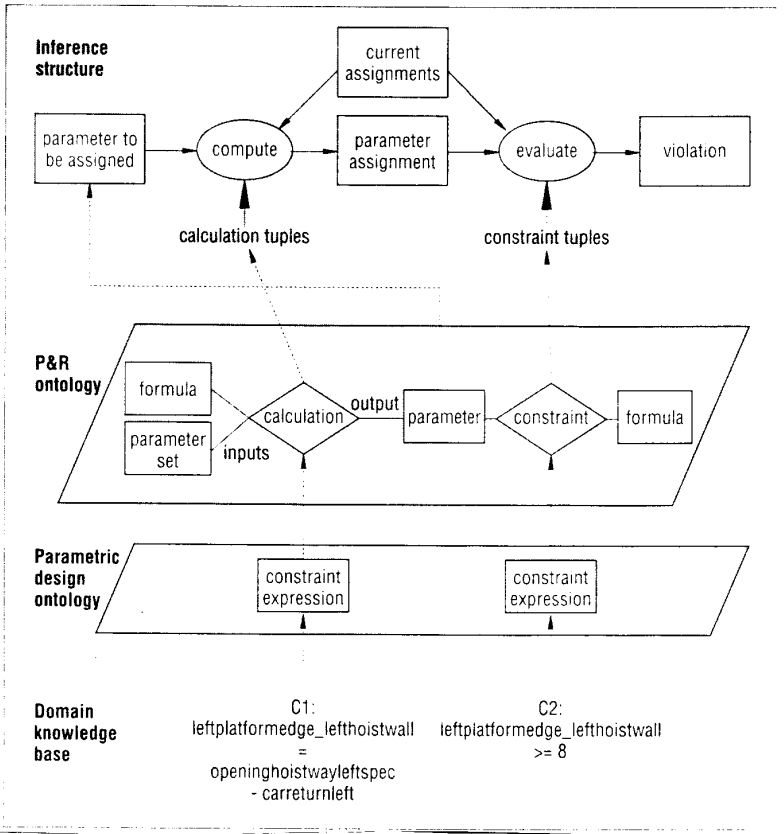

Figure 10. Linking domain and inference knowledge through ontologies, thus making their interaction explicit.

Table 1. Characterization of the propose-and-revise conglomerate of methods.

Task	PSM	Resulting decomposition
design	Propose-critique-modify	propose, verify, revise
propose	Decomposition into design plan	*no subtasks*
verify	Domain-specific calculations	*no subtasks*
revise	Dependency-directed backtracking	find-fixes, apply-fix, propagate-fix

Specification formalisms

CommonKADS provides two formalisms for the specification of an expertise model. The CML formalism used here is a highly structured but still informal notation. It is used for the initial specification, and designed to be easily usable by knowledge engineers with a CommonKADS background. If the domain is well understood, this CML specification can be judged to be sufficient input for system design. Alternatively, the ML^2 formalism can be used to construct a formal specification. For ML^2, a theorem prover is available that allows model validation through (partial) simulations of the reasoning behavior. Tools that support transformations from a CML specification to skeleton ML^2 specification are available.

to carry out the task. Therefore, we can characterize it as a task-type-oriented ontology. The *propose-and-revise ontology* is shown in Figure 6. It describes the structure of the domain knowledge in the format required by the inferences of the method selected (propose-and-revise).

In Figure 6, *constraint* and *calculation* are defined as *viewpoints* on constraint expression in the parametric-design ontology. This method-oriented viewpoint enables us to partition the set of constraint expressions into two subsets that are each used in a different way: The "calculations" are the constraint expressions used by the *compute* inference; the "constraints" are used by the *evaluate* inference. In this way, each ontology can define its own interpretation of terms. For example, the term "constraint" in the propose-and-revise ontology has a much more restricted meaning than "constraint expression" in the parametric design ontology. The ontological levels attribute *context-specific semantics* to domain knowledge elements. This is in contrast with the traditional logicist's view of model-theoretic semantics, which implies a description of semantics at one level.

The elements of the propose and refine ontology are linked to inferences. This creates yet another metamodel defining how domain knowledge elements are manipulated dynamically during reasoning. In addition, methods often require specific knowledge that is not part of a more general ontology such as the parametric design ontology. For example, the propose-and-revise method requires knowledge of *fixes*: knowledge that describes how to change parameters when a constraint is violated. This additional method-specific knowledge is represented as a separate knowledge base, and has its own ontology.

By using different ontologies with different generality, and by partitioning the knowledge base accordingly, we can identify different classes of knowledge bases with different scope, generality, and reusability. For example, when the distinction between task-type-oriented and method-oriented ontologies is identified, it is easier to identify parts of the knowledge base for reuse in a similar task when applying a different method.

Control knowledge: problem-solving methods. We have mentioned the propose-and-revise method to solve the VT task. However, in the task specification in Figure 9 this method is never explicitly mentioned. The reason for this is that a method is in fact a metalevel notion that prescribes how a task definition (a goal) can be mapped onto a task body (a goal satisfaction procedure). Such a method is called a *problem solving method* (PSM).

A specification of a PSM is similar to a task specification. The main difference lies in the additional information about *competence* and *acceptance criteria* of the PSM. The PSM can be selected when a task definition specifies a goal that matches the competence of the method, provided that the acceptance criteria are met. A PSM decomposes a task into subtasks (such as propose, verify, and revise) or, alternatively, provides a direct way to achieve a task. A PSM can introduce additional roles that serve as place holders for intermediate results, and can provide a template for a control regime over the subtasks. This information is essentially sufficient to create a task body.

The propose-and-revise method used in solving the VT task is in fact a conglomerate of methods for solving a design problem. Table 1 characterizes propose-and-revise in terms of the design-task analysis framework.[9] The method underlying top-level decomposition in propose-and-revise is an instance of the class of propose-critique-modify methods (although in propose-and-revise there is no explicit critique task).

A KADS library of PSMs developed in 1987 has proven to be of help to many knowledge engineers in application development. At the minimum it provides useful initial ideas for expertise models, and ideally it changes the nature of the modeling process from a design-from-scratch task into a configuration-like activity. In the present CommonKADS library,[10] the support has been improved by reducing the grain size of library elements from wholesale models to a broad range of configurable components, and by giving better guidance to the actual *construction* of an appropriate model. The CommonKADS expertise modeling library covers nine problem types: diagnosis, prediction, assessment, design, planning, assignment, scheduling, configuration, and modeling.

The PSM specification in the library does not provide automatic mechanisms to apply a method to a task definition; the PSM specification should be viewed as a structured way to write down knowledge about problem solving. In principle, it is possible to integrate the PSMs as an explicit part of the expertise model, and allow the system to decompose the task dynamically. This approach can greatly enhance the flexibility of the KBS, allowing it to cope with a wider range of problems. It requires, however, additional knowledge about how to achieve goals. We call this additional metaknowledge *strategic knowledge*.[11] It is in fact similar to the *strategic layer* in earlier versions of the KADS expertise model.[8]

System design

The models discussed so far capture the various types of requirements for the target system, in particular the expertise model and the communication model. Based on these requirements, the CommonKADS design model describes the structure of the system that needs to be constructed in terms of the computational mechanisms, representational constructs, and software modules that are required to implement the expertise and communication models. The design model has three constituents:

- *Architecture design* defines an abstract computational machine that provides the basic primitives for realizing the application.
- *Application design* describes how the various elements of the expertise model and communication model are mapped onto the primitive elements of the architecture.
- *Platform design* defines the hardware and software infrastructure in which the system will be implemented.

CommonKADS does not prescribe a particular design approach, such as object-oriented or rule-based design. As a general rule, realizing a system will be simple and transparent if the gap between application and architecture specification is small — that is, that the expertise and communication modeling constructs map easily onto computational primitives in the architecture. For example, although it is possible in principle to map the expertise model onto a first-generation rule-based architecture, such a design would lose the distinctions between the various types of knowledge. All knowledge types would be mapped onto the flat rule base, reducing maintainability and reusability.

The approach that is favored in CommonKADS is the *structure-preserving design* approach. The basic principle here is that distinctions made in the expertise model are maintained in the design and the implemented artifact, while design decisions that add information to the expertise model are explicitly documented. (Design decisions specify computational aspects that are left open in the expertise and communication models, such as the representational formats, computational methods used to compute inferences, dynamic data storage, and the communication media.) The advantage of a structure-preserving design is that the expertise and communication models act as a high-level documentation of the implementation, and thus provide pointers to elements of the code that must be changed if the model specifications change.

The structure-preserving design approach predefines a skeletal architecture that provides the basic computational mechanisms needed to implement the expertise model: task execution mechanism, dynamic data storage, inference method execution, and access mechanisms for the domain knowledge.[12] To construct the part of the design model related to the expertise model, we first map the elements of the expertise model onto elements of the skeletal architecture. Additional design decisions are then made to specify the precise representation of the domain knowledge, the locus of inference control, and so on. Finally, an implementation platform is chosen that further constrains the design decisions.

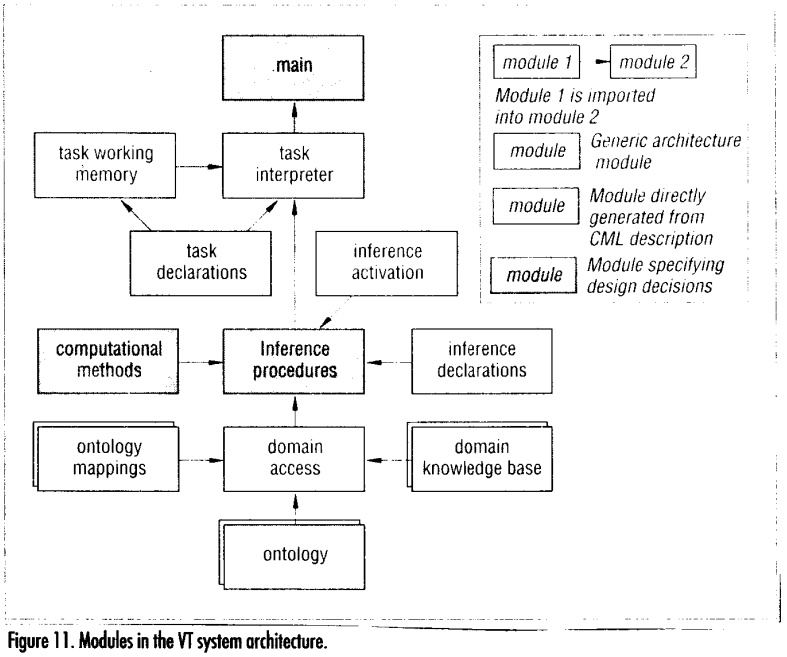

Figure 11. Modules in the VT system architecture.

Figure 11 shows the software modules resulting from the high-level architecture design for the VT application, based on the structure-preserving principle. White boxes represent the modules that are part of the skeletal architecture. Boxes with lightly shaded edges represent the modules that can be generated automatically from the expertise model specification. Together, these modules contain all the information present in the expertise model: task and inference declarations, ontologies, definitions of mappings between ontologies, and domain models (the domain knowledge bases). The dark boxes are modules that contain specifications of additional design decisions, such as those with respect to the computational methods used. The result is a highly modular design, parts of which can be easily reused in other applications.

CommonKADS in perspective

One of the salient features of CommonKADS is the modeling approach. The suite of models not only acts as a means to carve up the world in manageable pieces, but it also supports the selection of those aspects that are relevant and potentially amenable to risks in the development process. The templates for each of the models form the core of the methodology. The notion of model states, along with the decoupling of project management and planning from the development process, also provide mechanisms for flexible project configuration and control.

The knowledge modeling approach in earlier versions of KADS has matured in CommonKADS, and now includes extensive facilities for modeling of domain knowledge. This approach has also been brought into line with others, such as components of expertise, generic tasks, Protégé-II, and SBF. The CommonKADS expertise model supports the introduction of ontologies as a mechanism for generating abstracted descriptions of the structure and vocabulary of the domain knowledge. The ontologies also link the domain knowledge to the inference knowledge, thus explicating and minimizing the interaction between declarative knowledge and the task.

The decomposition of a knowledge system as used in the CommonKADS expertise model has similarities with the three viewpoints identified by current software-engineering approaches: data, functional, and control view. Separation of these views supports structured analysis and design, as well as modularization of the models and systems. However, the methods employed in Com-

monKADS differ in several ways from conventional modeling techniques. First, the expressiveness of the knowledge modeling formalisms is larger than that of entity-relation models or object-oriented analysis techniques. Second, the functional view (inference structure) in CommonKADS is much more constrained than in typical data-flow modeling techniques: the primitives that result from a full, functional decomposition are limited to certain sets of standardized operations. Finally, CommonKADS differs from the more conventional views in the nature of the link between domain knowledge ("data") and the inferences ("functions"). Through introduction of abstraction mechanisms in the form of knowledge roles and ontologies, the coupling between the data and functional views of the model is indirect, and hence there exists more potential for reuse of individual components.

ALTHOUGH THE COMMONKADS design model does not prescribe a particular approach, the structure-preserving design was found to be particularly fruitful. Preserving the structure of the expertise model in the system design allows linking between the knowledge model and the actual code. Such links can be instrumental in explanation, testing, and maintenance. In addition, structure-preserving design provides handles for an advanced design-support environment.

CommonKADS is a good candidate for becoming the *de facto* European standard and point of reference for knowledge engineering methodologies covering a wide range of KBS development aspects. Many successful projects have been performed with CommonKADS, albeit not always with the methodology in its present form. Commercial as well as academic (research-oriented) support tools are available for versions of CommonKADS. As a framework, CommonKADS has given rise to numerous research projects, and keeps on doing so. Ideas like knowledge-level reflection, formal specification, automated code generation, and knowledge sharing are being researched within the common framework provided by CommonKADS. The effects of these explorations will become more and more visible in the future.

Acknowledgments

Almost all participants in the KADS-II project have contributed to this work with comments, validation feedback, reviews, tool development, and so on. In particular, we acknowledge the contributions of Manfred Aben, Anjo Anjewierden, John Balder, Christian Bauer, Richard Benjamins, Bart Benus, Amaia Bernaras, Bert Bredeweg, Joost Breuker, Clive Bright, Cuno Duursma, Frank van Harmelen, Christiane Löckenhoff, Rob Martil, Winifred Menezes, Olle-Olsson, Klas Orsvärn, Philip Rademakers, Luc Steels, Peter Terpstra, Jan Top, André Valente, Johan Vanwelkenhuysen, and Steve Wells. We are grateful to Manfred Aben, Ameen Abu-Hanna, Anjo Anjewierden, Bart Benus, Frank van Harmelen, Gertjan van Heijst, and Peter Terpstra for their comments on earlier drafts.

Our research was carried out in the course of the KADS-II project. This project was partially funded by the Esprit Programme of the Commission of the European Communities as project number 5248. The partners in the project were Cap Gemini Innovation (France), Cap Programmator (Sweden), Netherlands Energy Research Foundation ECN (The Netherlands), Eritel SA (Spain), IBM France (France), Lloyd's Register (United Kingdom), Swedish Institute of Computer Science (Sweden), Siemens AG (Germany), Touche Ross MC (United Kingdom), University of Amsterdam (The Netherlands), and Free University of Brussels (Belgium). This paper reflects the opinions of the authors and not necessarily those of the consortium.

References

1. S. Marcus, J. Stout, and J. McDermott, "VT: An Expert Elevator Designer That Uses Knowledge-Based Backtracking," *AI Magazine*, Vol. 9, Spring, 1988, pp. 95–111.

2. B.W. Boehm, "A Spiral Model of Software Development and Enhancement," *Computer*, Vol. 21, No. 5, May 1988, pp. 61–72.

3. W.J. Clancey, "The Epistemology of a Rule-Based System — A Framework for Explanation," *Artificial Intelligence*, Vol. 20, No. 3, 1983, pp. 215–251.

4. A. Newell, "The Knowledge Level," *Artificial Intelligence*, Vol. 18, No. 1, 1982, pp. 87–127.

5. G. Yost, "Configuring Elevator Systems," tech. report, Digital Equipment Corporation, Marlboro, Mass., 1992.

6. S. Marcus and J. McDermott, "SALT: A Knowledge Acquisition Language for Propose-and-Revise Systems," *Artificial Intelligence*, Vol. 39, No. 1, 1989, pp. 1–38.

7. T. Bylander and B. Chandrasekaran, "Generic Tasks in Knowledge-Based Reasoning: The Right Level of Abstraction for Knowledge Acquisition," in *Knowledge Acquisition for Knowledge Based Systems*, B. Gaines and J. Boose, eds., Vol. 1, Academic Press, London, 1988, pp. 65–77.

8. B.J. Wielinga and J.A. Breuker, "Models of Expertise," in *Advances in Artificial Intelligence II*, B. du Boulay, D. Hogg, and L. Steels, eds., Elsevier Science, Amsterdam, 1987, pp. 497-509.

9. B. Chandrasekaran, "Design Problem Solving: A Task Analysis," *AI Magazine*, Vol. 11, Winter, 1990, pp. 59–71.

10. J.A. Breuker and W. Van de Velde, eds., *The CommonKADS Library for Expertise Modeling*, IOS Press, Amsterdam, 1994.

11. F. van Harmelen et al., "Knowledge-Level Reflection," in *Enhancing the Knowledge Engineering Process – Contributions from Esprit*, B.L. Pape and L. Steels, eds., Elsevier Science, Amsterdam, 1992, pp. 175–204.

12. A.T. Schreiber, B.J. Wielinga, and J.A. Breuker, eds., *KADS: A Principled Approach to Knowledge-Based System Development*, Vol. 11 of *Knowledge-Based Systems Book Series*, Academic Press, London, 1993.

Guus Schreiber is a postdoctoral fellow at the University of Amsterdam, where he is currently technical coordinator of an Esprit Project on reusable ontologies. He studied medicine at the University of Utrecht, The Netherlands, and worked for two years in medical informatics. Since 1986 he has been involved in a number of Esprit projects on methodologies for knowledge-based system development and reflective reasoning. In 1992, he was awarded a PhD on a thesis entitled "Pragmatics of the Knowledge Level." He has also co-organized a number of knowledge acquisition workshops. Guus Schreiber can be reached at the University of Amsterdam, Social Science Informatics, Roetersstraat 15, NL-1018 WB Amsterdam, The Netherlands; Internet: schreiber@swi.psy.uva.nl.

Bob Wielinga is a professor of social science informatics in the Faculty of Psychology, The University of Amsterdam, where he is team leader of several research projects. Since 1983, Wielinga has performed research on the methodology of knowledge-based system design and knowledge acquisition, and was one of the main developers of the KADS methodology for knowledge-based system development. He received his PhD degree cum laude from the University of Amsterdam in 1972 for a thesis in nuclear physics. From 1974 to 1977 Wielinga performed research on knowledge representation for computer vision in the Department of Computer Science of the University of Essex, Great Britain, and in 1977 was appointed senior lecturer at the Department of Psychology of the University of Amsterdam. Bob Wielinga can be reached at the University of Amsterdam, Social Science Informatics, Roetersstraat 15, NL-1018 WB Amsterdam, The Netherlands; Internet: wielinga@swi.psy.uva.nl.

Hans Akkermans is a professor of information systems at the University of Twente and principal consultant at ECN, the Netherlands Energy Research Foundation. Prior to that he was manager of ECN's software engineering department. He recently founded a consulting firm AKMC Knowledge Management. His research interests include the use of knowledge engineering methods in technological applications, in particular modeling, simulation, and engineering design, and he regularly publishes on topics in theoretical and computational physics. He received his MS and PhD degrees cum laude in physics from the State University at Groningen. He can be reached at the University of Twente, INF/IS Department, P.O. Box 217, NL-7500 AE Enschede, and at AKMC Knowledge Management, Klareweid 19, NL-1831 BV Koedijk, The Netherlands; Internet: J.M.Akkermans@cs.utwente.nl.

Walter Van de Velde is a senior researcher for the Belgian National Science Foundation, and co-director of the Artificial Intelligence Laboratory of the Vrije Universiteit Brussel. His research topics include second generation expert systems, machine learning, general architectures of intelligence, and autonomous systems. Within knowledge engineering, he has been particularly interested in knowledge-level modeling, reusability, and understanding the modeling process. His present research centers on coordination of behavior and its relation to cognition, specifically in a multiagent context. His advanced education is in mathematics, and in 1988 he earned his PhD from the Vrije Universiteit Brussel with a thesis on learning from experience. Walter Van de Velde is a member of the board of the European Coordinating Committee on AI, and president of the Belgian AI Association. He can be reached at the Free University of Brussels, AI Lab, Pleinlaan 2, B-1050, Brussels, Belgium; Internet: walter@arti.vub.ac.be.

Robert De Hoog is an associate professor of social science informatics at the University of Amsterdam, Faculty of Psychology. His main research interests are in knowledge-based systems and decision support systems. He is involved in several international research projects sponsored by the European Community. He has also carried out several studies in the field of knowledge-based systems for government agencies and the European Space Agency. He has published more than 50 papers in these fields and also coauthored a book on project management. Robert De Hoog can be reached at the University of Amsterdam, Social Science Informatics, Roetersstraat 15, NL-1018 WB Amsterdam, The Netherlands; Internet: dehoog@swi.psy.uva.nl.

Additional information about CommonKADS can be obtained through anonymous FTP at swi.psy.uva.nl/pub/CommonKADS, or through the World Wide Web at http://www.swi.psy.uva.nl/projects/CommonKADS/home.html.