Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern

SEKI - REPORT

CAKE: Computer-aided Knowledge Engineering

Frank Maurer

SEKI Report SR-91-09 (SFB)

CAKE: Computer-aided Knowledge Engineering

Frank Maurer
University of Kaiserslautern
Dept. of Computer Science
P.O. Box 3049, D-6750 Kaiserslautern
Germany
e-Mail: maurer@informatik.uni-kl.de

Abstract

In this paper we describe the CAKE-System which uses a hypertext abstract machine (HAM) to support the development of expert systems. The CAKE-System is based on the ideas of KADS but tries to overcome some of its shortcomings. It is developed to support a scenario where multiple knowledge engineers cooperate with multiple experts to built a maintainable knowledge base.

Keywords

Knowledge Engineering, Knowledge Acquisition, Hypertext, KADS

1. The Knowledge Engineering Process

Based on our experience in expert systems for diagnosis, construction, and planning in technical domains, we work on a workbench which supports multiple knowledge engineers cooperating with many experts in building knowledge bases. We divide the process of knowledge acquisition in two steps (which may also be found in literature): First a model must be constructed which describes all relevant entities of the domain,

second these entities must be instantiated with the domain knowledge. Therefore we define:

knowledge acquisition = model construction + knowledge elicitation.

To construct the model the knowledge engineers have to collect data about the domain. Then they have to structure the gathered information to find how the problem is solved: they build a conceptual model (see [Wielinga, Schreiber, Breuker 91]). After structuring the domain a formal ("design") and operational ("implementation") model must be built. To summarize, the results of the model construction process are

- a collection of all gathered informations about the domain (today this data is stored on different media; often it is only inside the heads of the knowledge engineers),
- a description of the domain in a semi-formal language (e.g. for diagnosis this language includes concepts like symptoms, tests, fault descriptions, etc.),
- descriptions of the possible inferences and tasks,
- a specification of the integration of the resulting expert system into the application environment (e. g. the interaction with the human users, the interfaces to other software systems), and
- parts of a knowledge base and an inference engine.

In the second step the domain knowledge is elicitated and expressed in the (developed) language (e.g. the possible symptoms of a failure in a CNC machining center are represented, the symptoms which determine a special failure are asked, etc.).

The knowledge acquisition process results in an expert system which shall be used to support the every day tasks of the end users. Therefore, it normally must be integrated into a heterogeneous environment of people and computer systems which are doing their work. The resulting system is not stable over a longer period of time. People learn to do their work in a better way, the communication structures change, the access to (external) sources of information is improved. Normally, these changes result in an adaptation of the software: maintenance is needed (which rises the costs of software development). This fact forces us to define knowledge engineering as:

knowledge engineering = knowledge acquisition + integration + maintenance

This view stresses that knowledge engineering primarily is a kind of software development and should produce systems which take advantage of the inherent abilities of computers. The machines should support people in doing their jobs. For this purpose we use methods which are appropriate for computers (which means the methods are

"computer adequate"). We think that the constructions of expert systems need not produce a program which simulates the human expert (which is "cognitive adequate") because the abilities of people are different from the abilities of machines. The *cooperation* of humans and computers should produce improved problem solving behaviour. Therefore, we specify the human-computer-interaction on a very abstract layer: inside the conceptual model.

2. Requirements and Critiques

The first step in building expert systems is collecting a huge amount of information. Thereby the knowledge engineers discuss the problem with the experts, read text books, get tables and diagrams etc. A lot of different media are involved in transfering these informations. We think that these data should be stored in a single system so that it can be used for further maintenance. We call this collection "Data Level".

Based on the data level description the knowledge engineers have to structure the domain resulting in the conceptual model. In KADS this step is not supported. Also, the description of a conceptual model in KADS mainly depends on the experience of the knowledge engineers[1]. Another critique on KADS is that the inference layer is specified in a kind of flow chart depending on the functionality of the system. Research in software engineering has shown ([Meyer 88]) that the development of software should be based on the data structures because they will change less in a program´s life than the functionality. This result leads to object oriented design, which is also not supported by KADS.

A main source of costs within the life of software (and also of expert systems) arises because of the difficulties in maintenance. So, a knowledge engineering system has to support the development of easily maintainable software. This may be supported by references from the source code to its underlying assumptions and informations. Further it must be possible to express when a knowledge base is consistent, so that a system may check automatically these conditions for the actual knowledge base.

---

[1] The attempts to define a formal, which means operational, language for conceptual models is in our opinion misleading: the distance from the data level to a formalized knowledge base is often to large that it can be done in one step. We think that an advantage of KADS is that it devides this distance in more than one step.

Another requirement in the development of expert systems is the support of rapid prototyping. In the context of model-based knowledge acquisition a rapid prototype may be seen as an operational specification.

From our point of view, a knowledge engineering system has to support a kind of "model-based rapid prototyping". The building of a rapid prototype is often needed to convince and satisfy the client who normally will not be able to handle an abstract description of the expert system to be built. A prototype may also help in the knowledge acquisition: It may serve as a starting point in a discussion of what is needed by the customer. Often he will only be able to specify further requirements if he gets a demonstration of a prototypical system. The prototyping should be "model-based" to overcome some shortcomings of usual rapid-prototyping approaches: they often result in a huge, unstructured rule-base. The KADS approach does not support the integration of model-based approaches with rapid-prototyping techniques.

Our experience showed that the development of expert systems is a cyclic process where specification and implementation phases are repeated several times. This is not a new result: in software engineering the early waterfall models where replaced by a spiral model. The basic KADS approach is mainly based on the overcome waterfall model[2].

Large expert systems will be developed by multiple knowledge engineers cooperating with multiple experts to collect multiple kinds of knowledge. Therefore, a knowledge engineering environment has to support this groupwork.

3. CAKE: A Hypertext-Based Knowledge Engineering System

Based on hypermedia techniques we develop a software system which supports the knowledge acquisition process based on the KADS methodology. Thereby we try to overcome the shortcomings mentioned above. Our approach emphasizes a design for maintainability of knowledge bases.

The CAKE system uses the hypertext abstract machine (following the ideas of [Campbell, Goodman 88]) for storing and retrieving the informations. Its basic entities are:

- Net or graph

---

2 The work of [Taylor et al. 89] tries to overcome this situation, too.

A net is a collection of nodes, links, and contexts.

- Node
  A node is an information unit. Nodes may contain texts, pictures, audio signals, tables, formulas etc. Nodes are typed to provide special semantics (e. g. also rules are expressed as nodes).
- Link
  A link connects nodes to express a kind of dependency (e. g. special links express that a node is a comment of another node or one node is a specialization of another). The user may follow a link from one node to the connected informations and so explore the hyperspace.
- Context
  Contexts divide a net into sets of objects belonging together (e. g. a context may contain all informations belonging to the conceptual model whereas another context stores the design model descriptions). A context may include other contexts (e. g. for describing the conceptual model on different levels of abstraction).

To support the knowledge engineering process we enlarged the HAM by possibilities to express consistency conditions. We implemented a production rule language which is able to check these conditions referring to objects in a hypertext network[3]. The consistency rules are bound to the contexts of the net.

In the rest of this chapter we describe the different models which are created by the knowledge engineering process. Each model is represented as a context. Each context is a collection of (sub)contexts which express different levels of abstraction inside a model. Contexts on the lowest level only contain nodes and links. This results in the multi-layer abstraction hierarchy shown in figure 1. Links may connect objects inside a or between different contexts. If a context is the target of a link then all the objects contained may be accessed.

Cooperative work is supported by

- The possibility to define different visibility conditions on the net by access control rights for each entity.

---

[3] Based on this rule language we integrate an inference engine into the HAM which will be used for the building of an expert system about environmental planning.

- The possibility to define different views on the net by a filter mechanism[4]. Thereby we take advantage of the typing of nodes, contexts and links.
- The possibility of an adaptation of the user interface depending on the needs of different users.
- The browsing facilities of the hypertext machine.
- The possibility to comment any information stored.

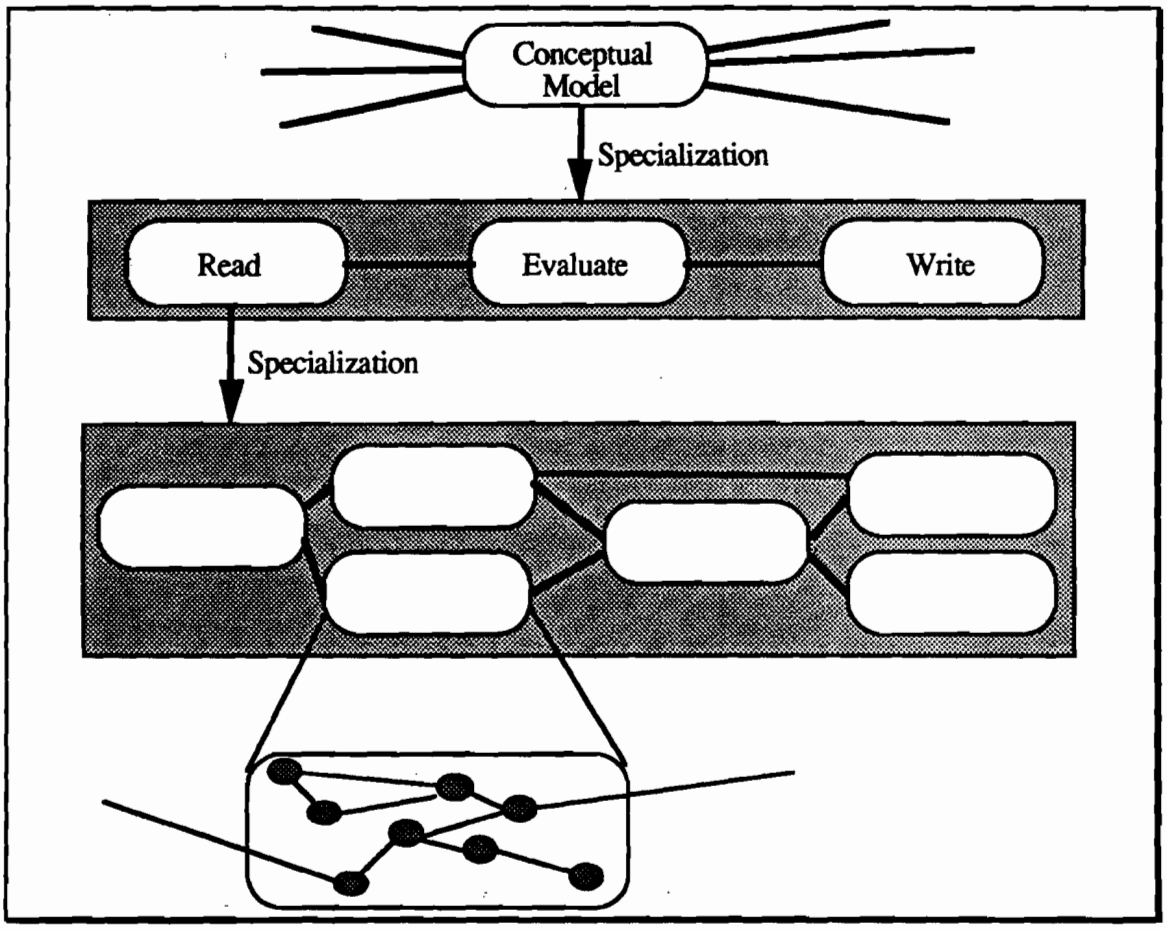

Figure 1: A multi-layer abstraction hierarchy

3.1. The Data Level

All gathered information about the domain is represented as (typed) nodes inside the hypermedia network in a natural way. So we are able to store and easily retrieve texts, pictures, tables, mathematical formulas, and audio signals. This informal, unstructured conglomerate of knowledge is structured by the knowledge engineers. They are able to

---

[4] The filtering mechanism is a property of the underlying HAM.

insert new links which represent connections between the nodes (e.g., a node may be the reason for an inference step, which is contained in another node). The CAKE-System supports the model construction process by handling multiple versions of nodes and by storing refinement and reason links[5]. In fact, the hypermedia network may be used for building an explanation component for the expert system. The resulting data level model consists of

- a specification of the input/output functionality of the KBS to be built,
- a specification of the concepts of the domain (the *language*),
- a specification of the possible inferences, and
- some (prototypical) instances of the domain.

All specifications are stored informally using natural language or other representation formalisms like sound and pictures with all its ambiguities. The advantage is that a data level model uses the language of the highest expression power for human communication.

3.2. The Conceptual Model (The Knowledge Level)

The next step is based on the work on CASE (see for example [Gane 90]). We will implement a software tool for a graphical description of a conceptual model. Thereby we focus on an object-oriented design perspective and not on the functionality of the expert system to develop.

The tool is going to be integrated in the hypermedia environment. Therefore, we are able to document the transition from the informal data level model to the semi-formal conceptual model by inserting special links between nodes on different layers. In this way the development process becomes more transparent.

3.3. The Design Model

The design model refines the conceptual model by formalizing the data structures and algorithms. Here the needed inference engines and data structures have to be specified:

- What representation formalisms are needed (Frames, Rules, Constraints etc.)?
- Do we need a kind of TMS for non-monotonic reasoning?
- How can we solve the control problem?

---

5 Refinement links connect abstract and concrete information on multiple layers, reason links connect inferences and their reasons.

Further the access of needed data is described. This includes:

- A description of the interaction of the user with the system (the "model of cooperation").
- A specification of the access path to information stored in databases and spreadsheets.
- A description of the access of sensoric informations.

The design model is tailored to object-oriented specifications. This means that we have to specify the slots (instance variables), an inheritance hierarchy and the protocol of the objects. This results in multi-dimensional object definitions. CAKE will be able to support an efficient reorganization of the definitions: moving slots up and down in the inheritance hierarchy, reorganizing the hierarchy itself based on the protocol of objects.

3.4. Remarks on the Implementation Level

Based on the design model, an expert system shell and the runtime environment are implemented (resulting in an implementation model). This shell mainly contains a formal language which represents the acquired domain concepts (e.g., symptoms, failure descriptions (see above)). Furthermore, an acquisition interface is implemented which allows the expert to enter his knowledge. We think that this direct transfer is only possible if the knowledge representation language reflects the expert's terminology. To accelerate the filling of the shell, advanced expert system techniques (case-based reasoning, inductive learning, qualitative modelling) may be used (see [Althoff, Maurer, Rehbold 90]).

The implementation is supported by a knowledge dictionary which enables the knowledge engineers to search for already implemented object classes and generic interpretation models. Although software reuse is a main goal of the CAKE approach the indexing of the available software is, until now, not solved in a really satisfying way. Today we are able to find existing modules by keywords or synonyms of keywords. This is only a syntactical match. A further possibility is to browse existing nets for informal specifications of old solutions which solve the new problem. Here the work is mainly done by the knowledge engineers by hand. This possibility heavily depends on a well organized group of networks.

The implementation should use a interpretative programming language to shorten the edit-compile-execute-cycle. This is needed to support a fast development and allows rapid-prototyping. We decided to use Smalltalk-80 as the programming environment.

3.5. Filling the Model

To fill the constructed model (= to edit the knowledge base) our system is able to use different approaches: first we support the building of an interactive editing environment, secondly we are able to integrate case-based reasoning and machine learning approaches.

3.5.1. Consistency Checking and Maintenance

A problem with expert systems is that knowledge is often not stable over time and only the expert(s) know when and how the knowledge base must be updated. So our goal is that the expert himself maintains the knowledge base. For this reason, our first step in the knowledge acquisition process is to provide a knowledge representation language which uses the expert's terminology. Further, we provide a generic maintenance component which allows to specify consistency conditions about the objects of the shell. These are automatically preserved by the system. Last, we are able to built easily an environment for the editing of the knowledge base which is constructed depending on the wishes of the experts. The knowledge engineers are only needed for maintenance if the constructed model must be updated. They are not needed for changes in the elicitated knowledge.

3.5.2. Automated Knowledge Acquisition

The result of the model construction process is, among other things, a definition of the needed data structures. Often we will get some kinds of formulas and rules as the appropriate description of entities. For example, an error classification is naturally described by a rule with a specification of a situation as condition and the related diagnosis as right hand side. These data structures may be generated by case-based reasoning an machine learning approaches. The work of [Weß 91], [Althoff, Maurer, Rehbold 90] and [Althoff, Weß 91] deal with these aspects.

4. State of Realization and Future Work

The hypertext abstract machine which allows to store and retrieve text and audio information is (prototypically) implemented. A forward-chaining rule interpreter and a

filtering mechanism are also realized. Now we start the implementation of two applications in environmental planning domains. These application will show shortcomings of our current prototyp and lead to its improvement.

Our prototype is realized in Smalltalk -80. Because of the image concept of Smalltalk-80 we are not able to share parts of a hypertext network. Also the access control lists are not implemented. We will improve our CAKE-System by the use of an object-oriented database (Gemstone) for storing and retrieving of informations. Then access rights will also be supported resulting in better possibilities for cooperative work.

Acknowledgments

I like to thank Scarlett Nökel, Mike Stadler and Stefan Weß for reading and discussing preliminary versions of this paper.

References

[Althoff, Maurer, Rehbold 90]

K. D. Althoff, F. Maurer, R. Rehbold: Multiple Knowledge Acquisition Strategies in MOLTKE, in: Proc. European Knowledge Acquisition Workshop 90

[Althoff, Weß 91]

K.-D. Althoff, S. Weß: Case-based Knowledge Acquisition, Learning and Problem Solving for Diagnostic Real World Tasks, Proc. EKAW 91, 1991

[Campbell, Goodmann 88]

Brad Campbell, Joseph M. Goodman: HAM: A General Purpose Hypertext Abstract Machine, Communications of the ACM, July 1988, Vol. 31, No. 7

[Wielinga, Schreiber, Breuker91]

B.Wielinga, A Th. Schreiber, J.Breuker: KADS: A Modelling Approach to Knowledge Engineering, KADS-II/T1.1/PP/UvA/008/1.0, Esprit Project P 5248 KADS-II, 1991

[Gane 90]

Chris Gane: Computer-Aided Software Engineering, Prentice-Hall, 1990

[Meyer 88]

Bertrand Meyer: Object-oriented Software Construction, Prentice-Hall International, 1988

[Taylor et al. 89]

R. Taylor, D. Porter, F. Hickman, K.-H. Streng, S. Tansley, G. Dorbes: System Evolution - Principles and Methods (the Life-cycle Model), ESPRIT project P 1098, Deliverable task G9, Touche Ross, 1989

[Weß 91]

S. Weß: PATDEX/2: Ein System zum adaptiven, fallfokussierenden Lernen in technischen Diagnosesituationen, SEKI Working Paper SWP-91-01, University of Kaiserslautern, 1991 (in German)