Between Help and Engineering: constraints on user task automation

Joost Breuker
University of Amsterdam
Social Science Informatics (SWI)
Roeterstraat 15
1018 WB Amsterdam, the Netherlands
breuker@swi.psy.uva.nl

Radboud Winkels
University of Amsterdam
Computer Science and Law
Kloveniersburgwal 72
1012 CZ Amsterdam, the Netherlands
email winkels@lri.jur.uva.nl

Abstract

In this paper we describe two projects relevant for automating user tasks. The first one, EUROHELP was aimed at tools and methods for developing Intelligent Help Systems (IHS). Experiences and accomplishments in this project shed light on understanding and monitoring task performance of users. The major bottleneck appears the lack of understanding of the content of user tasks, calling for research in ontologies rather than trying to come to grips with the common sense semantics of the workplace. A major part of the paper describes the *Common*KADS projects in which a methodology and tools were developed for building KBS. *Common*KADS has a long (> 10 years) and extensive history, and it is argued that it has survived several shifts in attention in problems about knowledge acquisition (KA), because it has focused on well defined, coherent specification languages. The need for such a core is signalled for current approaches in KA which provide operational components ((configurations of) mechanisms) to domain experts to develop small, but practical knowledge based systems.

1 Introduction

In this paper we discuss problems and limitations of automating user tasks based upon experiences in two large scale research and development projects: EUROHELP and KADS. First we will give a brief outline of the (relevant) aspects of these projects. The second part of this paper focuses on the limitations and problems we have identified in these projects with respect to (supporting) the automation of user tasks. We assume that the issue is not whether we can provide automation to users in general. Technology and tools, that range from high performance powertools to intelligent workmates have been developed (e.g. in KADS and in EUROHELP) to enable task automation. However, the question now is rather whether we are able to provide tools to users who will be able to automate their own tasks either as the side effect of their recurring activities (e.g. via user modeling, plan recognition and learning) or by simple constructive activities which do not require software or knowledge engineering skills. In short, we want a non-programmers programmer's assistant. The experiences in EUROHELP pertain in particular to understanding tasks of users while they are performing tasks with conventional applications such as word-processors or mailing systems. In discussing KADS — a knowledge engineering methodology — we will focus on a library of problem solving methods, and their indexing.

2 EUROHELP

EUROHELP was concerned with tools for the development of Intelligent Help Systems (IHS). [1] These IHSs provided both active and passive help to users of (conventional) applications such as word processors, operating systems, mailing systems etc. Passive help means that the user can ask questions to the IHS about the application; active help means that the IHS 'looks over the shoulder' of the user, interprets her actions and offers spontaneous help. Reasons for offering help are errors, inefficiencies and opportunities to expand the skills of the user. Figure 1 summarizes the architecture of a EUROHELP IHS. The monitor consists of two submodules: a planner and a plan recognizer. The plan recognizer interprets the user's performance in terms of local goals or subtasks, while the planner generates the correct/efficient ways to achieve such goals, given the current state of the user's competence (user model). The diagnoser is called in when the user's performance deviates from such plans, and traces these in terms of lacking or confused knowledge. Lacking knowledge is not restricted to the proper use of the commands of the application, or its current state, but also to its working. When a diagnosis

[1] EUROHELP was an Esprit project (P280, 1985 – 1991). It involved 120 personyear and delivered tools and a methodology for IHS construction [Breuker, 1990]

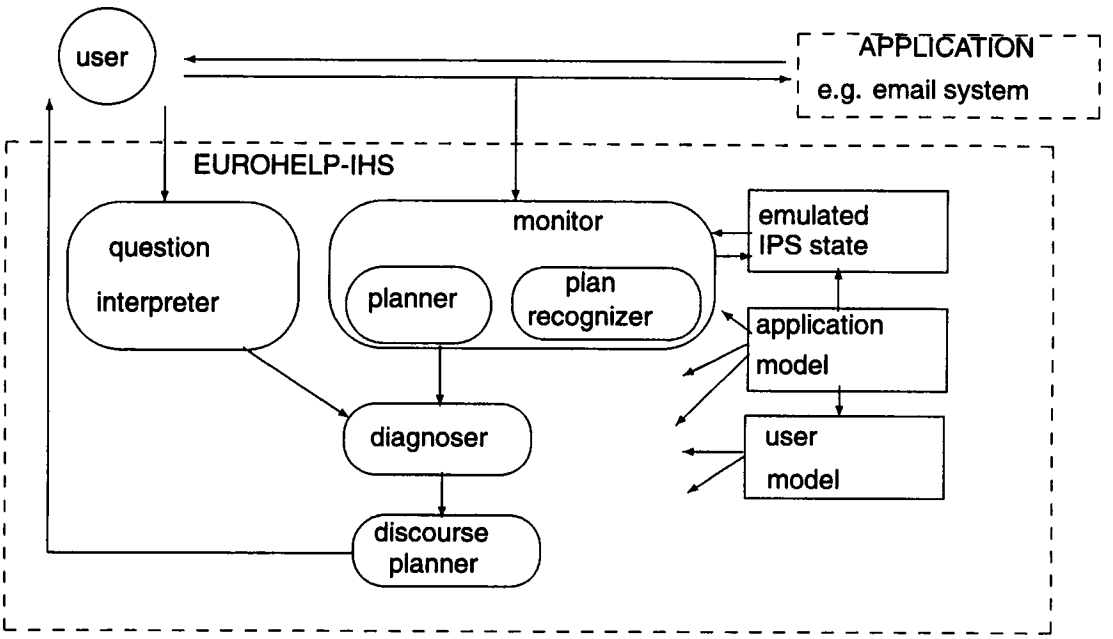

Figure 1: Architecture of an EUROHELP IHS

is found, the discourse planner provides an explanation [Winkels, 1992]. The question interpreter allows the user to specify his problem or question via the use of an application dependent menu structure [Pilkington, 1992]. Because the question interpreter is coupled to the monitor, questions are understood in terms of the current state of performance and of the application. This is in a nutshell what a EUROHELP IHS consists of (see also [Breuker and Wielinga, 1987; Winkels, 1992]). A complete description of EUROHELP is available in [Breuker, 1990].

EUROHELP is based upon detailed empirical studies of novice users of conventional applications [Sandberg *et al.*, 1988]. These studies showed that, when monitored by human experts, users hardly take the initiative to ask questions. Most interactions are the result of expert interventions. The major reason is that the user is often not aware of problems or alternative routes, or is not capable of specifying his problem. Note, that in these experiments full communication is possible between user and expert. A second important issue is that the focus is on task performance. However, the plans (methods) for executing these tasks easily breakdown, because they often "encapsulate interactions" [Simmons, 1992]. Making or correcting plans often involves *understanding how the application works.* The distinction (and mapping) between 'task knowledge' and 'insight in the application' is explicit in the domain representation (application model). Therefore, we have taken the fact that a user should know how an application works, or at least what the world it operates on consists of, is as important as to know how to handle it. In any educational philosophy, skill and insight are mutually supportive (e.g. [Anderson *et al.*, 1990]).

A third, related issue is the fact that task decompositions are highly dependent on the world in which the task is to be executed. This seems also a trivial observation, but it leads to most of the trivial problems of novice users. If one does not have a good insight of the (side) effects of automated actions, planning can only be performed 'by analogy' to known worlds. However, this leads to problematic differences or mismatches as Fig 2 shows, because the digital (mail) world is different from the physical one.

3 KADS

*Common*KADS [2] is a comprehensive methodology for developing KBS. A *Common*KADS project consists of the development of a suite of dependent models (see Fig. 3). Two models, the Organization model and the Task model describe the operational context of the KBS(s) (the "workplace" [Yost *et al.*, 1994]). The combination of Task model and Agent model contains the specification of the task distribution between the agents involved (machine, human). The Communication model specifies the required communications to support this cooperation (see [Breuker and de Greef, 1993]).

[2] KADS, initiated by SWI, University of Amsterdam, was developed in various Esprit projects, between 1984 and 1994. It took more than 200 personyears. Some of the major partners were: Cap Gemini, Siemens, IBM, STC, Touche Ross, Lloyds Register, ENTEL, SICS, Free University of Brussels. KADS, or some adapted form is used by many industries in Europe and the USA.

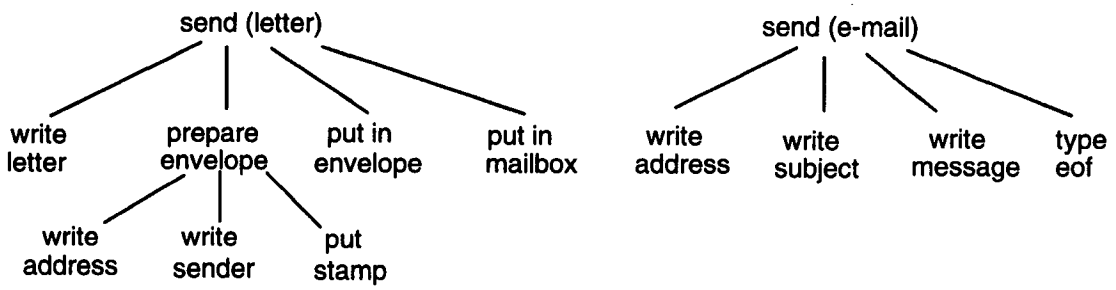

Figure 2: Sending mail: plans (task decompositions, methods) for e-mail (Unix-mail) and snail mail

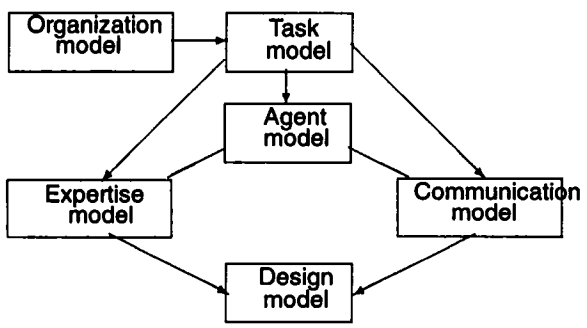

Figure 3: Suite of models in a *Common*KADS project

Two other models provide specifications of the KBS itself: the Expertise model and the Design model. The Expertise model contains a 'knowledge level' specification of the required knowledge to build a knowledge based system, analogous to a data model in software engineering. The Expertise model is the input for the construction of the Design model, i.e. the specification of the system architecture that implements the knowledge specified in the Expertise model. This two step system specification philosophy separates considerations of understanding and modeling the required problem solving competence ('expertise') from *efficient* and well structured implementations.

The Expertise model can be viewed as a conceptual interface between the "workplace" and the machine. This means that it has an inherent ambiguity. On the one hand, the terms to be used should be easily identifiable and understandable; on the other hand these terms should not only be well defined, but also easily map onto machine mechanisms. Most research efforts in KADS have therefore been devoted to the design of a "Conceptual Modeling Language" (CML) to express Expertise models [Breuker and Wielinga, 1989; Wielinga *et al.*, 1992; Schreiber *et al.*, 1993].

In CML, like in most other approaches in knowledge acquisition (KA), domain knowledge is distinguished from task knowledge. In task knowledge there are two views. (1) A knowledge flow view, in which the dependencies ((dynamic) roles) between basic reasoning steps (inferences) are specified. Such a network is called an inference structure. (2) Control can be added to an inference structure, resulting in a task structure. A task structure is a(n instantiated) problem solving method (PSM), i.e. its major structure is a task decomposition, bottoming out in inferences. We will not describe here CML in detail (see e.g.[Schreiber *et al.*, 1994]). Different from most other approaches in KA (e.g. SBF [Yost *et al.*, 1994], EXPECT [Swartout and Gil, 1995; Gil and Paris, 1994], KREST [Geldof and Slodzian, 1994]) CML specifications are not directly operational for several reasons such as efficiency (cf. the Design model) and independence of specific implementation platforms. However, CML can be translated into its formal language $(ML)^2$, i.e. FOPL augmented with reflection and various other logics, which is executable by very inefficient proving [van Harmelen and Balder, 1992]. The role of $(ML)^2$ is rather in validation and consistency checking than for operationality.

Expertise models are not necessarily to be built from scratch, in a bottom up way. The CK-Library [3] provides reusable components for constructing expertise models [Breuker and de Velde, 1994]. This does not mean that the CK-Library is to be used exclusively in projects that follow a *Common*KADS methodology. Most of the components to be found in the CK-Library are in fact inspired by or copied from other approaches, and in particular from fundamental research in Artificial Intelligence (e.g. on diagnosis or planning methods). Moreover, it can be shown that the CK-Library also supports other approaches [Valente *et al.*, 1993].

The role of such a library is not so much the reuse of these components, but rather the *application* of these modeling fragments. In reuse, the primary effect is efficiency by copying or reinstantiation. However, the problem solving components are relatively simple objects: only in their combinations and their domain specific instantiations may turn out to be very complex. The major role of the CK-Library is the guidance of the modeling process, i.e. the

[3] The full name is *Common*KADS Expertise modeling Library, or CKEL. Here I will simply use the term CK-Library

knowledge acquisition. The problem in knowledge acquisition is that it is very difficult for a (novice) knowledge engineer to identify the types of tasks and knowledge in the data — documentation, interviews, etc. — of some domain. A knowledge engineer may need to know whether it is a diagnostic or an assessment task which has to be automised, what problem solving methods may be applicable to this task, what assumptions on the nature and structure of the domain knowledge are implied by a particular method, etc. Therefore, the CK-Library is not a simple store, but rather a *framework for reuse*, that should provide guidance to its user. This guidance is not so much in separate, linked documentation, but it is more or less wired in by the structure and the access of the CK-Library. This structure and access guides the process by constraining the possible routes through the library. Only those components are accessible from another, selected component which are compatible. In traversing the CK-Library, local selections are made, and the selected components preserve the relations which are implied by the structure of the CK-Library. The selections are made on the basis of data on knowledge in the application domain that do or don't match the assumptions — features — that are associated with a component. [4]

The major access to the CK-Library is by a typology of problem types, such as diagnosis, planning, assessment [Breuker, 1994]. The typology is not a taxonomy, but a "suite", which expresses the dependencies between these types as they occur in (sub)tasks (see Fig. 4). These types are characterized by their solutions (see Table 1). It is beyond the scope of this paper to explain in detail the nature of these problem types. This suite exhibits a number of properties which make it attractive for KA. The types are mutually exclusive and appear to cover the full range of problems/tasks as identified in the literature and in a decade 'KADS experiences'. It is based upon the notion put forward by [Clancey, 1985] that problem solving consists of the construction of "a model of a system-in-the-world". Although artificial problem solvers hardly ever construct a case model *in an explicit way*, [5] this is indeed what is implied by understanding a problem. When the model is already given as generic knowledge in the knowledge base, this simplifies the construction to the selection and instantiation of the applicable knowledge. In that case, the task contains problems at the analytic spectrum of the suite. Otherwise, the case model has to be made, constrained by requirements, and we will characterize the task as a synthetic one (design, planning). Therefore, an important property of the suite is that it guides the decomposition of tasks. For instance, any PSM has two functional components: a part which generates solutions and a part which evaluates or tests these solutions. The test part invariably follows the suite at the analytic side. The suite not only enables the recognition of this paradigm in real life tasks, but also enables the construction of new task decompositions (PSM).

Not only the access, but also the bottom of the CK-Library is defined by a typology. The inferences, which are the terminal functions of PSMs, are typed according to the operations they perform with respect to a KL-ONE like ontology (see Table 2). A formalization of this typology can be found in [Aben, 1994]. It should be noted that other ontological commitments result in other typologies. However, this typology has the advance that the commitments are relatively small and widely accepted as a basis for representing knowledge, so that the scope for application is large. The types are well differentiated, and based on a systematic covering of "what can happen to a concept", as compared to typing like deduction, abduction and induction ("what can happen to a set of propositions"). Moreover, the terms refer to reasoning operations which are not context dependent, as e.g. the teleological terms that are used to characterize, for instance, task or problems. This is a nice property for reusable building blocks. (See also [Wielinga *et al.*, 1992] and [Aben, 1994] for alternatives.) These inferences have a fixed meaning but not a fixed grain size level. They can be further decomposed, if necessary. They are standard or canonical terms, rather than primitives or atoms.

This concern in KADS with language, typology and formalization shows that apparent approach to knowledge acquisition is in providing conceptual, knowledge level tools, which on one hand easily map onto real life data, and which on the other hand are well defined to allow computational rendering. These analytic tools are supplemented by reusable models and modeling components for specifying the problem solving knowledge for an application.

4 Coming to terms

The problem typology of the CK-Library claims that the dependencies between kinds of (well defined) problems are more or less universal. An analysis of PSMs for diagnosis reveals that subtasks solve a particular kind of problem, whose result is used to solve problems in another (next) subtask [Benjamins, 1995]. For instance, in the test parts of a diagnosis-PSM contain prediction and monitoring problems. The predictions and observations are a necessary input for a monitoring problem/subtask, which identifies conflicts, etc. These dependencies are most explicit in all model based diagnosis PSM. In association based (heuristic) methods they are less clear, because of the compiled out nature of associations between symptoms and (classes

[4] An implemented version of the CK-Library is commercially available as part of the KADS-Tool workbench by ILOG (France); [Breuker and de Velde, 1994] provides a description of the philosophy, structure and ingredients of the CK-Lib. The CK-Library is not 'complete'. Not only new (arrangements of) PSM may be developed, but there is hardly any support for domain knowledge modeling. The emerging interest in reusable ontologies may fill this gap in the near future.

[5] A notable exception is ABEL's 'patient specific model' [Patil, 1988]

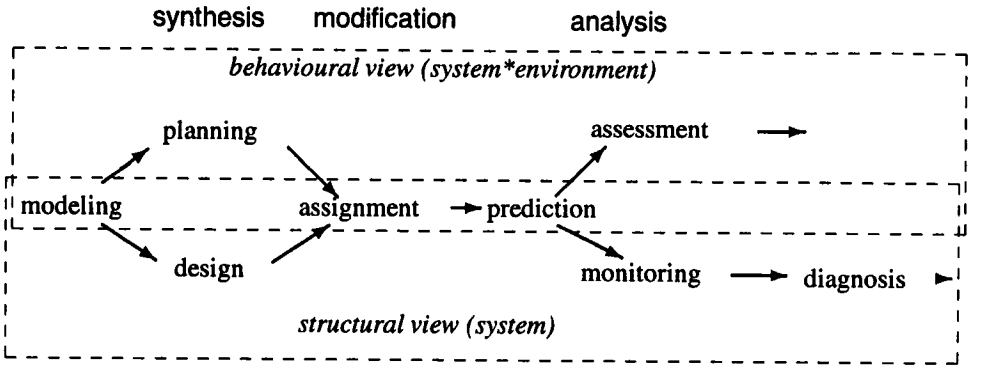

Figure 4: A suite of problem types, dependencies and views

major type	type of problem	solution
synthesis	modeling	behavioural model
	design	structure of elements
	planning/reconstruction	sequence of actions
modification	assignment (scheduling, configuration)	distribution/assignments
analysis	prediction	state of system
	monitoring	discrepant states
	diagnosis	faulty element
	assessment	class/grade attribution

Table 1: Type of problem and corresponding solutions

operation type	inference	input → output arguments
generate concept	instantiate	concept → instance
	classify	instance → concept
	generalize	set of instances → concept
	abstract	concept → concept
	specify	concept → concept
	select	set → concept/instance
change concept	assign-value	attribute → attribute-value
	compute	structure → attribute-value
differentiate	compare	value, value → difference value
	match	structure, structure → difference
manipulate structure	assemble	set → part-of structure
	decompose	part-of structure → set
	transform	structure → structure

Table 2: Typology of inferences [Wielinga *et. al.*, 1992]

of) malfunctions. The identification of (sub)tasks characterized by a type of problem from the suite can be obscured by these shortcuts, but another important reason is that the *terms* by which these problems/subtasks are described are highly context dependent. Diagnosis may mean such diverse things as a malfunctioning component, a class of malfunctions or a sequence of events that caused a problem (accident), which refer to different problem types: diagnosis, assessment, respectively reconstruction (planning [Simmons, 1992]).

If in the literature there is a terminological confusion on what the terms mean and refer to (see [Breuker, 1994]), in the world of users the terminology has become a major bottleneck in the access and deployment of 'knowledge level' tools for automating problem solving tasks. Initially, i.e. at the end of the 80-ies, most of these tools incorporated some particular PSM for a particular kind of problem and with a particular set of implied requirements on the nature of the domain knowledge e.g. Generic Tasks [Chandrasekaran, 1988], Role Limiting Methods [Marcus, 1988]. Although domain expert (users) were quite capable to fill these shells, it appeared that the scope was too limited, i.e. only for a few problem-type/domain combinations the implied PSM appeared to be suitable. A more general solution was found in assembling systems rather than in filling readily available structures. PSM can be built from combinations of basic, reusable subtasks: inferences (KADS), or mechanisms (SBF). Tools for building artificial problem solvers, like KREST, EXPECT and SBF contain some *library of mechanisms* which can be accessed and configured by some task description. As it appears that the terms used in these task descriptions are different across domains, basic mechanisms are added, and the access interface grows to complex libraries by itself (e.g. [Yost *et al.*, 1994]). This pragmatic approach facilitates easy reuse, but lacks a structuring framework for systematic indexing. There is no way to assess whether, or to what extent, the mechanisms are functionally equivalent or the terms are distinctive. In KADS exactly the opposite happened. Much effort has been put in the grounding of terms and structuring of the CK-Lib, but no reusable operationalizations are available. KADS fits within the software engineering tradition that puts the emphasis on accurate specification (languages) rather than easy programming of problem solving tasks. However, the *Common*KADS CML and related typologies could well be used as unifying descriptors of tasks and mechanisms (cf. [Linster and Musen, 1992]). The problem typology provides a reference framework for task descriptions; the typology of inferences gives a terminology for the functional (and formal) description of mechanisms [Aben, 1993].

The EUROHELP experiences provide insights at the other end of the user task spectrum. The tasks of users of conventional applications are not typical problem solving tasks, but rather information management tasks. They are not knowledge intensive and the intelligence required for performing these tasks is the *planning* of automated actions (leaf sub-tasks). The planning problems involved are not too complex, so that EUROHELP could construct executable scripts instead of instructing the user what to do if in trouble. However, such a "butlering" mode was not recommended for the simple reason that it would only hide the application, and learning to talk to the butler and to understand his capabilities would be as difficult as learning to use the application itself. We may hypothesize that the scope and size of effective automation of a user task is proportional to its intensity of knowledge use. In knowledge extensive tasks, which do not have a strongly recurring routines, a set of simple procedures may be more effective than encapsulating complex routines (cf. the flexibility of knowledge acquisition toolboxes with simple mechanisms).

A second suggestion from EUROHELP concerns the feasibility of automated user-task automation. Specifications were written to combine plan recognition, student modeling and machine learning techniques. Recognized plans were stored in the student model and routine plans would be induced from repeated patterns. These routine, stereotypical plans could be used both for user-tailored recognition and help, and for limited butlering. Implementation, however, was abandoned, because experiments showed that 1) plan recognition had a very limited scope, because the intentions of users were not available, nor could they be assumed and (2) sequences of actions and their variations were largely 'content' dependent. For instance, understanding what the user of a text editor is up to may require understanding writing competence (e.g. spelling, style, grammar [6]) and (some of) the topics involved. This domain dependence for understanding the tasks that users perform with machines (and of course, also without machines) suggests that at least the terminology (ontology) of the domain should be available and explicitly represented before automated user task induction becomes feasible.

References

[Aben, 1993] M. Aben. Formally specifying re-usable knowledge model components. *KAJ*, 5:119–141, 1993.

[Aben, 1994] M. Aben. Canonical functions: CommonKADS inferences. In Joost Breuker and Walter Van de Velde, editors, *CommonKADS Library for Expertise Modelling*, pages 89–120. IOS-Press/Ohmsha, Amsterdam/Tokyo, 1994.

[Anderson *et al.*, 1990] J.R. Anderson, C.F. Boyle, A. Corbett, and M. Lewis. Cognitive modeling and intelligent tutoring. *Artificial Intelligence*, 12:7–49, 1990.

[6] One may object then that text editing is a knowledge intensive task. Indeed it is, but in automation only the information management parts have been taken out. Fuller automation would indeed require text generation competence, but the core of this competence is already automated in humans: spelling and style checking are now often integrated functions.

[Benjamins, 1995] Richard Benjamins. A suite in Dm. In *The Knowledge Engineering Forum Conference*, pages –, 1995.

[Breuker and de Greef, 1993] Joost Breuker and Paul de Greef. Modelling system-user cooperation. In Guus Schreiber, Bob Wielinga, and Joost Breuker, editors, *KADS: a Principled Approach to Knowledge Engineering*, pages 47 – 70. Academic Press, London, 1993.

[Breuker and de Velde, 1994] Joost Breuker and Walter Van de Velde, editors. *The Common KADS Library: reusable components for artificial problem solving*. IOS-Press, Amsterdam,Tokyo, 1994. to appear june 1994.

[Breuker and Wielinga, 1987] J.A. Breuker and B.J. Wielinga. Use of models in the interpretation of verbal data. In A. Kidd, editor, *Knowledge acquisition for Expert Systems; a practical handbook*, pages 12 – 30. Plenum Press, New York, 1987.

[Breuker and Wielinga, 1989] J. A. Breuker and B. J. Wielinga. Model driven knowledge acquisition. In P. Guida and G. Tasso, editors, *Topics in the Design of Expert Systems*, pages 265–296, Amsterdam, 1989. North Holland.

[Breuker, 1990] J.A. Breuker, editor. *EUROHELP: developing Intelligent Help Systems*. EC, Copenhagen, DM, 1990. 401 pp.

[Breuker, 1994] Joost Breuker. Components of problem solving. In Luc Steels, Guus Schreiber, and Walter van de Velde, editors, *A Future for Knowledge Acquisition: proceedings of the EKAW-94, European Knowledge Acquisition Workshop*, pages 118 – 136, Berlin, 1994. Springer Verlag.

[Chandrasekaran, 1988] B. Chandrasekaran. Generic Tasks as building blocks for knowledge based systems: the diagnosis and routine design examples. *The Knowledge Engineering Review*, 3(3):183 – 210, 1988.

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

[Geldof and Slodzian, 1994] Sabine Geldof and Aurelien Slodzian. From verification to modelling guidelines. In Luc Steels, Guus Schreiber, and Walter Van de Velde, editors, *A Future for Knowledge Acquisition: Proceedings of the 8th European Workshop on Knowledge Acquisition*, pages 226–243, Berlin, 1994. Springer.

[Gil and Paris, 1994] Y. Gil and C. Paris. Towards method independent knowledge acquisition. *Knowledge Acquisition*, 6:163–178, 1994.

[Linster and Musen, 1992] M. Linster and M. A. Musen. Use of KADS to create a conceptual model of the ONCOCIN task. *Knowledge Acquisition*, 4(1), 1992. Special issue: 'The KADS approach to knowledge engineering'.

[Marcus, 1988] Sandra Marcus, editor. *Automating Knowledge Acquisition for Expert Systems*. Kluwer, Reading MA, 1988.

[Patil, 1988] R.S. Patil. Artificial intelligence techniques for diagnostic reasoning in medicine. In H.E. Shobe and AAAI, editors, *Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence*, pages 347–379. Morgan Kaufmann, San Mateo, California, 1988.

[Pilkington, 1992] R. Pilkington. *Intelligent Help: communicating with knowledge based systems*. Oxford University Press, 1992.

[Sandberg *et al.*, 1988] J.A.C. Sandberg, J.A. Breuker, and R. Winkels. Research on help-systems: Empirical study and model construction. In Y. Kodratoff, editor, *Proceedings of the European Conference on Artificial Intelligence, ECAI-88*, pages 106 – 111, London, 1988. Pitman.

[Schreiber *et al.*, 1993] Guus Schreiber, Bob Wielinga, and Joost Breuker. *KADS: a Principled Approach to Knowledge Engineering*. Academic Press, London, 1993.

[Schreiber *et al.*, 1994] Guus Schreiber, Bob Wielinga, Hans Akkermans, Walter Van de Velde, and Anjo Anjewierden. Cml: the CommonKADS conceptual modelling language. In Luc Steels, Guus Schreiber, and Walter Van de Velde, editors, *A Future for Knowledge Acquisition: Proceedings of the 8th European Workshop on Knowledge Acquisition*, pages 1–25, Berlin, 1994. Springer.

[Simmons, 1992] R.G. Simmons. The role of associational and causal reasoning in problem solving. *Artificial Intelligence*, 53(2–3):159–207, 1992.

[Swartout and Gil, 1995] Bill Swartout and Yolanda Gil. EXPECT: explicit representations for flexible acquisition. In *Proceedings of the Ninth Knowledge Acquisition for Knowledge Based Systems Workshop, KAW95*, 1995.

[Valente *et al.*, 1993] Andre Valente, Joost Breuker, and Bert Bredeweg. Integrating modelling approaches in the Common KADS library. In A. Sloman, D. Hogg, G. Humphreys, Allan Ramsay, and D. Partridge, editors, *Prospects for Artificial Intelligence, Proceedings of the AISB-93*, pages 121 – 130. IOS Press, Amsterdam, 1993.

[van Harmelen and Balder, 1992] Frank van Harmelen and John Balder. ML^2: a formal language for KADS models. *Knowledge Acquisition*, 4:127–161, 1992.

[Wielinga *et al.*, 1992] B. J. Wielinga, A. Th. Schreiber, and J. A. Breuker. KADS: A modelling approach to knowledge engineering. *Knowledge Acquisition*, 4(1):5–54, 1992. Special issue "The KADS approach to knowledge engineering".

[Winkels, 1992] R. Winkels. *Explorations in Intelligent Tutoring and Help*. IOS-Press, Amsterdam, Tokyo, 1992.

[Yost *et al.*, 1994] Gregg Yost, Georg Klinker, Marc Linster, David Marques, and John McDermott. The SBF framework, 1991 – 1994: from applications to workplaces. In Luc Steels, Guus Schreiber, and Walter Van de Velde, editors, *A Future for Knowledge Acquisition: Proceedings of the 8th European Workshop on Knowledge Acquisition*, pages 318–340, Berlin, 1994. Springer.