Computer Support by Knowledge Enhancement, Constraints and Methodology

de Greef, H.P.

Citation for published version (APA):
Amsterdam: UvA
Part II

Model-Based Design of Cognitive Support
Chapter 5

Analysing System–User Cooperation in KADS

This chapter provides the feedback from the StatCons case-study to the KADS project. There are two main problems in the development of a knowledge based system (KBS). The first is the modelling of domain expertise. The second is the modelling of the application of this knowledge to tasks that future users want to perform. This paper discusses how the second problem can be addressed in a systematic way. Our argument is that the second problem is at least equally important and if it is given serious attention, the first main problem will become simpler, because efforts can be directed at the subset of expertise that is actually required.

This Analysis of Cooperation helps to arrive at a consistent set of functional requirements for a future KBS and a population of intended users. It comprises (i) a theoretical framework for system development, (ii) a technique for constructing a model of cooperation and (iii) a recommendation to use the Wizard of Oz technique for validating a model of cooperation in experiments with future users. In such an experiment, users attempt to perform tasks with the help of a mock-up of the future system operating according to the model of cooperation.


Introduction

It is difficult to develop an operational knowledge based system. Even if one succeeds in modelling and implementing some expertise using some AI paradigm, it may very well turn out that the intended users cannot use it or have no use for it in their
everyday work. Introducing operational knowledge-based systems (KBS) has subtle, but further-reaching consequences than automation by conventional systems in professional organizations.

Conventional systems are largely viewed as semi-passive tools under control and command of the user, whereas a KBS may take the role of intelligent, active agent. Therefore, a “careful” specification of how the user and the artificial agent should and can cooperate becomes far more decisive in knowledge engineering than it was thus far in software engineering. “Careful” means that empirical tests can be made about the functional feasibility before the KBS is designed and implemented. “Careful” also means that the functionality itself is based upon a distribution of tasks between users and system, rather than on the default assumption that a KBS should play (one of) the role(s) of an expert.

The cooperation between a KBS and a user is mediated via the user interface. Most user interfaces of KBS look less sophisticated than those of many advanced conventional systems, which use graphics with direct manipulation, etc. The typical “advisory” KBS interacts via canned text, and keyword dialogues. The system takes the initiatives and these initiatives are driven by the current state of the problem solving process, rather than by the pragmatics of discourse. From the user’s point of view the dialogue is unstructured, because the current topic may have no relation with a previous one, and, moreover, the user cannot take initiatives to redirect the discourse. He can only ask for some explanations.

Although not all KBS simply mimic the role of a human expert, and not all KBS exhibit this stereotypical MYCIN-like scenario of advice giving (e.g. Miller 1984), in knowledge engineering there are no methods which allow a specification of the role of a system by successive refinement. Neither is such a methodology available from software engineering. Moran’s (1981) framework for specifying passive (command based) user interfaces can be viewed as a pioneering step (see also Card et al. 1983), but it did not lead to widespread application, due to its complexity and lack of integration with software engineering methods. In the KADS project (Wielinga, Schreiber and Breuker, 1992) we have taken Moran’s view, in particular about layers of abstraction, and extended it to active systems (KBS). This paper presents the result which we call Analysis of Cooperation. It provides a framework that is both applicable to software and knowledge engineering, thus supporting an integration of both in practical projects. The framework helps in the specification of functional requirements, but it does not address the design of the user interface.

5.1 Functions, Cooperation and Communication

Our method for the analysis of cooperation comprises a framework and a specification language for constructing models of cooperation and we recommend a partial ordering of knowledge engineering activities. This ordering is the consequence of dependencies that exist in the top-down framework between the more abstract and more specific layers. This framework is to a large extent inspired by notions from distributed artificial intelligence (Bond & Gasser, 1988; Gasser, 1991), job analysis (Adam & Ebert, 1982), organization theory (Mintzberg, 1983), the design of user interfaces (in particular Moran 1981), user-centered system design (Norman 1986; Gould 1989), and conventional software engineering (e.g. Wallace, Stockenberg and Charette 1987,
The analysis of cooperation is based upon the notion that user and system should cooperate to achieve a common goal. Cooperation is based upon three elements:

A distribution of tasks or a distribution of labour. A distribution of tasks is a task decomposition in which sub-tasks are assigned to different agents. An agent may not know or accept the goal, but may only be committed to a particular sub-goal, that can be accomplished by his sub-tasks. Commitment is the typical term used in distributed artificial intelligence where we will use “assignment” or “allocation” (see e.g. Gasser 1991, Cohen and Levesque 1990).

Dependencies. In a task distribution, there is also a network of dependencies where one sub-task may require the output of another sub-task as an input. In our terminology the ingredients (objects) produced in a sub-task have to be transferred to another task, thus leading to requirements for communication. The way the communication is performed is, strictly speaking, not part of the cooperation; it is a further refinement that can be postponed to a later stage in system development.

Control. Agents involved in the task distribution must at least know which sub-tasks they have to perform when. In man–machine cooperation there usually is no separate supervisory control and neither does it seem feasible to implement a supervisory agent in the machine, for the simple reason that the machine has too small a communication channel. It can hardly intelligently monitor the user (Suchman, 1987). Instead the control can be driven by local recognition of “turn taking” – e.g. the prompt of a machine– or be based upon common knowledge of the task distribution – a shared plan and its current state of execution.

These elements are sufficient when the task distribution has become fixed. A task distribution is fixed when the work process and its outputs are standardised and do not require adaptive coordination mechanisms (Mintzberg, 1983). A fixed task distribution, and the specification of the dependencies and control issues, form the model of cooperation. Ideally, a knowledge-engineering project would start from a fixed task distribution which reflects the prospective distribution of labour between systems and human agents in an organization. As is also the case in software engineering projects, the analysis of functions in the organization is preliminary to the functional specifications of the systems to be built, and both activities are often performed by the same team. Arriving at a task distribution, i.e. a specification of roles and functions of various types of agents, is a first step in all automatization, and therefore included in the analysis of cooperation.

In distributed artificial intelligence (DAI) the modelling of cooperation involves more elements, because the aim is the uncovering of principles for designing dynamic task distributions. These are assumed to be more flexible than standard rational architectures with top-down, rather than distributed control. Confronted with new types of problems, autonomous agents propose and negotiate a task decomposition, or distribution (e.g. Hewitt 1991, Gasser 1991). A similar process of negotiating and planning between human agents can be involved in a social organization, for instance in making decisions about functions to be automated. The first step in a knowledge or software engineering project consists of “negotiating” the role and the functions of applications.
In general, in building systems, the objective is to assign the most repetitive and invariant roles to machines. Multi-agent planning capabilities (Stuart, 1987), would only be a last option to be built in the machine, if the organization of labour also required flexible coordination capabilities of the machine. This may change as a consequence of introducing KBS technology, where roles requiring more flexibility can also be automated; even then, this may be accomplished in a top-down manner rather than by the bottom-up nature of typical DAI systems. In KADS the flexibility of a KBS is specified at the task and strategy levels and the flexibility of the cooperation between a KBS and a user can also be specified at these levels, i.e. as top-down control. In other words, the full DAI paradigm in which there is bottom-up multi-agent planning, is not necessarily relevant to modelling cooperation between user and KBS. DAI is merely concerned with intra-KBS cooperation.

The role of the analysis of cooperation in KADS is shown in Figure 5.1. Any knowledge engineering project starts concerning intelligent automatization of some functions, but initially there are no real commitments or assumptions about the role and use of the system. An encompassing real world task is the starting point for creating a task model (Figure 5.1). This real world task must have a scope large enough to contain all sub-tasks with which the (to be) automated functions have input or output dependencies. For instance, the basic task of a statistical consultancy system includes not only the design, execution and analysis of quantitative experiments, but also the reporting of the results (de Greef & Breuker, 1989). In practice the identification of the encompassing task poses few problems.

The creation of a task model involves decomposition of the task, identification of interdependencies among sub-tasks and the distribution of sub-tasks over the agents "system" and one or more "user types". This initial activity results in one or more fixed task distributions. Tasks allocated to the system are input to the KADS analysis of expertise, or, depending on their nature, to conventional analysis methods.

The second and final activity of the analysis of cooperation is a refinement of
the task model into a model of cooperation. Because of the interdependencies among sub-tasks, each task distribution in the task model implies a number of transfer tasks. The refinement into a model of cooperation adds a specification of the control that is needed to synchronize system activity and user activity. This specification is surprisingly simple: for all transfer tasks in each task distribution one has to specify which agent -user or system- will have the initiative for the transfer.

The distribution of tasks not only embodies an expectation about the feasibility of automating the tasks allocated to system, but it also embodies an expectation concerning the capabilities of the prospective users. Therefore, the final activity in the analysis of cooperation also involves a validation of the cooperation model in experiments with prospective users. These experiments help to identify problems in the cooperation. Small problems may be amendable by adding functions to support the user. Severe problems may give rise to a revision of the task distribution.

After the specification and testing of the model of cooperation, it can be integrated with the model of expertise. This integration into the KADS conceptual model (see Wielinga et al. 1992), can occur at the task or at the strategy layer of the model of expertise.

5.2 The Task Model: Decomposition and Distribution

Tasks can be decomposed in endless varieties. One option is to take the current distribution of labour in an organization. However, this has as an important disadvantage that the task distribution will not fully exploit the specific capabilities of human and artificial agents. For instance, tasks which require common sense reasoning are preferably assigned to users, while tasks with a high ratio of information management, repetition, or well-known routines are rather performed by systems. Therefore, some iterations between decomposition and distribution are likely to optimize the effective exploitation of capabilities and preferences of machines and people. Because the new decomposition and distribution of sub-tasks may drastically differ from the current situation, an appropriate term for these activities is "job design" (Adam & Ebert, 1982). In the task distribution one does not have to decide whether the automation of sub-tasks allocated to the system should be accomplished by KBS or by conventional systems. The fact that there is no a priori commitment to knowledge or software engineering approaches enables an integrated development from the start of a project. The analysis of cooperation is indifferent for both approaches.

Although task decomposition is a major "divide and conquer" principle in designing problem solving systems (e.g. Steels 1990) the DAI literature hardly contains any explicit guidelines or techniques for task decomposition (see Bond and Gasser 1988, p. 10). We suggest the following heuristics (see also Breuker 1991):

Object decomposition. If parts can be distinguished in the product (the output) of a task, each of these parts may be produced by different sub-tasks.

Object refinement. If, in the output of a task, levels of abstraction can be distinguished, sub-tasks may consist of a sequence of refinement steps. The framework for analysis of cooperation is in itself an example, and is typical for design tasks,
which start with an abstract “skeletal” structure, that can be refined in the next stages (Brown & Chandrasekaran, 1989).

**Functional sequencing.** This is more or less the same principle as for object refinement, but in this case the production of the final output is viewed as a sequence of operations or transformations on the same object. An example is painting, where the object has to be cleaned, sanded, prepared, etc.

**Knowledge typing.** If the knowledge required to perform a task model is “strongly typed”, this may suggest a decomposition according to the type of knowledge required. Each sub-task is then viewed as a “specialist” agent, and may be reflected in the modularization of the architecture of the KBS.

These suggestions are not exhaustive. In practice the design of one or more decompositions poses little problems, even on the basis of a simple interview with the client, or for a relatively inexperienced knowledge engineer. However, the effectiveness and validity of a task distribution is hard to assess. Task distributions are, rather, objects of a process of negotiating between client, experts, prospective users, and knowledge engineers, in a similar way as it is in DAI-systems between agents before a cooperation can be executed.

The specification of the decomposition is represented as an AND/OR tree. OR branches reflect different decompositions, that is, different methods (cf. the GOMS model, Card et al. 1983). The need for explicit control constructs (conditionals; iteration) has been negligible in our experience. However, nothing prevents a knowledge engineer from annotating a task distribution with traditional control constructs. What should be avoided is to hide control information in the input/output dependencies (see below). When interdependencies between sub-tasks are taken into account, a hierarchical data flow diagram can be used. The conventions for the diagram of the decomposition are rather standard. The nodes in the tree are bubbles containing the name of the (sub-)task. The name is often a verb and some object, and the most neutral term is “achieve (object)”. In general, the object is the output of the task. Dependencies between tasks, i.e. the flow of data (objects), are indicated by arrows labeled with the name of the object. If there is a dependency between two sub-tasks of different governing tasks there is also a dependency implied between the governing tasks. Because these dependencies of sub-tasks aggregate upward, there is also an implicit hierarchy of objects. A sub-task may have more than one input, because it may come from various other sub-tasks. In general, a sub-task has a single, intended output, but there may also be side-effects. If there is more than one major and intended output, a further decomposition seems appropriate. Note that the same output of a sub-task can be the input of several others. It is assumed that the inputs and outputs are not control or coordination information, because control is either included in an abstract and semi-fixed way in the AND/OR tree of sub-tasks, or pushed to the lower levels.

One of the criteria for an effective task decomposition is to keep the number of dependencies as low as possible. Another one is to optimally exploit resources: time, effort, available problem-solving competence (knowledge and skills of agents) etc. In general, distribution according to competence is the major issue. If the decomposition into sub-tasks distinguishes well between human and machine capabilities, allocation is not a problem, and almost contingent upon the task decomposition.

The competences of users may differ, and also their roles in an organization. For
instance, both nurses and doctors may use a medical advisory system, but they may use it for different purposes. The tasks of a nurse are different from those of a doctor. This means that there are two different types of user, and that for each type of user the task distribution may be different. Of course, there should be sufficient overlap in tasks to warrant the construction of a KBS that switches “mode” between types of users, instead of constructing two different KBS. Users (agents) may also strongly differ in competence. For instance, users who are not familiar with the domain require other ways of cooperation than experts in the area. In general, for novice users the system will perform more tasks, so that the task decomposition may be the same as for experts, but the distribution is different (de Greef & Breuker, 1989).

The allocation of sub-tasks to user (types) and system(s) should result in unique assignments. There are several reasons why this may not be the case.

**Insufficient refinement.** The task decomposition may not be sufficiently refined so that it seems that both the system and the user participate in the same sub-task. This problem is solved by further decomposition of the task.

**Dynamic assignments.** It cannot be determined in a fixed way who is going to do what. The allocation should occur in a dynamic way, after some “negotiation” between the user and the system. It means that the system should be able to configure a new distribution, or should have at least an alternative available. For instance in statistical consultancy, the user’s problem may not be the question as to what is the statistical model that is applicable to his experimental design, but the user may want to know whether the experimental design is in accordance with the model he has selected. The medical advisory KBS may have a “nurse” and a “doctor” mode of operation.

**Parallel sub-tasks: coaching and critiquing.** The same sub-task can be performed “in parallel” by the system and the user. This may look very redundant, but it is in fact required for coaching, and for critiquing (Miller, 1984). In this case the system has an extra sub-task: comparing the (intermediary) results of the user and its own. This comparison, or monitoring, is for instance the basis for coaching activities, where a discrepancy between the system’s “correct” results, and those of the user (student) are used to diagnose the cause in terms of lacking knowledge or know-how (Breuker et al. 1987).

**Instruction and execution.** Two agents may participate in the same sub-task, when one agent instructs how to do the task, and the other one simply executes these instructions. In fact this is the way most passive, conventional systems work, because the user issues the instructions in the form of commands. That is also a reason why in conventional systems the notion of task distribution is unclear, and why Moran’s (1981) framework had to be split in two columns to be applicable to active KBS. In KBS these roles may be switched, where the KBS tells the user what to do (see the cooking planner example in section 5.4).

Except for the first one, these reasons justify maintaining shared sub-tasks in the distribution, but in the analysis it should be made clear what roles “system” and “user” have in these shared tasks.
Once the task distribution is accepted and fixed, the first refinement step is taken. For all tasks allocated to the system decisions are taken on which ones are candidate for conventional approaches, and which ones are knowledge intensive, requiring knowledge engineering. The latter are to be analysed using the standard KADS analysis of expertise (see Breuker and Wielinga 1989, Wielinga et al. 1992).

The next refinement step in the analysis of cooperation is the specification of the model of cooperation and the empirical validation in experiments with users. This can proceed in parallel with the KADS analysis.

5.3 The Cooperation Model

The KADS model of expertise can be viewed as an autistic problem solver, that does not know how to cooperate and communicate with the user. Therefore, the KADS control layers (task and strategy) can be extended with elements that manage cooperation and communication with the user. This provides top-down control of the cooperation, as depicted in Figure 5.2.

![Figure 5.2: Problem solving and communication capabilities are controlled by a strategic meta level.](image)

The major element for specifying control to switch between problem solving and communication is initiative. Initiative states which agent is responsible for starting communication. From the point of view of the system it means that if the user has the initiative, its problem solving process may be interrupted; if the system has the initiative, it goes into communication mode.

Requirements for communication can be directly derived from the dependencies in the task distribution. Whenever there is a dependency between sub-tasks of different agents, the objects involved have to be transferred. We use here the term transfer because it is neutral with respect to how the transfer occurs. For instance, if the objects were physical objects, the way to transfer these objects would be called “transport”. If the objects are symbolic ones, the activity of transferring these is generally called “communication”.

The objects to be transferred are called ingredients, so each dependency between tasks performed by different agents becomes an ingredient, that has to be transferred. In general, the agent who produces an ingredient is also the owner of the ingredient. We mean by ownership not “possession”, but control. The combinations of ownership
of ingredients and initiative lead to four types of transfer tasks, for which we use the terms: receive, obtain, provide and present (see Figure 5.3 for an explanation).

We have distinguished three major types of ingredients: information, knowledge, and skill.

**Information** refers to specific states in the world or in the mind. Typical categories of information are:

- *Data*, such as the particular value of some variable (individual data), or structures of data, such as case descriptions.
- *Problem statements* are requested types of information.
- Other types of information may refer to specific internal, or mental states, such as: intention, evaluation, history or the current state of a problem solving or communication process. Many of these states may coincide with meta-classes in the model of expertise, in particular those meta-classes which are input and output of an inference structure (see Wielinga et al. 1992).

**Knowledge** is not situation specific, but generic. Knowledge, as an ingredient for transfer, is used for explanation or teaching purposes.

**Skill** is transferred where the objective is to instruct the other agent on how to perform some sub-task.

The task distribution and the interdependencies imply a minimal set of information ingredients that must be transferred in cooperation. In addition, knowledge and skill may need to be transferred to support the user in doing his sub-tasks, when his competences are less than assumed in the design of the task distribution. Other types of additional ingredients may emerge for two reasons.
• Users may require explanation facilities which only have an indirect, supportive function because it increases their faith in the KBS, rather than bearing upon the strict cooperation itself.

• The process of communication itself may become the object of communication. For instance, there may be misunderstanding, or the user may want to change the topic of communication.

Beside the four standard transfer tasks there may be an additional transfer task: negotiate, which is aimed at transferring information about the cooperation, or the problem solving. Negotiate is typically used when a task distribution has to be selected, or when the user wants to correct information he has provided to the system.

5.3.1 Integration with the Conceptual Model

The task layer of the model of expertise (Wielinga et al. 1992) describes the more or less fixed task decomposition and top level control of the system. It indicates when a sub-task is started, and one of these sub-tasks can be a transfer task. When the user has the initiative and the transfer task may range over a number of sub-tasks, we use the monitor construct to indicate that the system can be interrupted over this range. An example of a task structure is presented in Figure 5.4.

```
PROCESS (problem, solution)
   NEGOTIATE (user_type, task_structure)

  task_structure_1
   SOLVE (problem, solution)
      RECEIVE (problem)
      MONITOR (userInterrupt1) => PROVIDE (problem_solving_trace)
         SPECIFY (system_model)
      ...
      OBTAIN (data)
      PROVIDE (purpose_of_data)
      MONITOR (userInterrupt1) => NEGOTIATE (data)
      ....
   PRESENT (solution)
      PROVIDE (justification_problem_solving_method)
      CLASSIFY (problem)
      ABSTRACT (problem_features, problem_type)
      SPECIFY....etc.
```

Figure 5.4: A typical task structure of a KADS Conceptual Model, highlighting its transfer tasks.
5.3.2 User Analysis: Testing the Model of Cooperation

"Wizard of Oz" Experiments

The model of cooperation is an idealized one, in which both the system and the user pursue the task goals in a rational way. However, the assumptions about the competences, preferences and intentions of the user may not correspond to reality. There are two ways to find this out.

The first one is the standard practice in software and knowledge engineering. Build a (prototype) system and evaluate the performance of the user. Methodologies for evaluating knowledge based systems describe how to collect data and to analyze these data (Hollnagel, 1989).

The second way is to mimic the prospective task distribution by using the “Wizard of Oz” technique to collect data. A domain expert and a user cooperate while communicating via terminals. Typically, the domain expert is instructed explicitly to follow the cooperation scenario as specified in the task structure of the conceptual model, but he is free to perform the sub-tasks and to communicate in a “natural” way. This set-up resembles the Turing test, but in this case the terminals are not intended to resemble computer output, or the user interface. Any other means of telecommunication – e.g. telexcopiers – that restrict the communication to one channel will do. Communication then proceeds by strict sequential actions and turn taking, which facilitates the analysis of the data. The “Wizard of Oz” technique was introduced by Stevens and Collins (1977), and further worked out by Carrol and McKendree(1987) and Sandberg, Breuker and Winkels (1988).

This method of user analysis has the advantage that it can be performed before the system or a prototype is actually built, which is most cost-effective. A second advantage is just as important. If the user communicates with a real system the evaluation includes both the cooperation and the communication. A system may fail to be effective and usable for many reasons, and in evaluating the complete system, errors in the design of the user interface can hardly be distinguished from problems in the cooperation. In the “Wizard of Oz” technique the medium for communication is the most universal one: natural language, which can be enhanced by the use of graphics. This means that communication is not a limiting factor in the cooperation, and that therefore, problems are rather due to assumptions about the user’s competences or intentions. In analysing the data, communication issues are not taken into account; only the understanding of the cooperation, and the “when” and “what” in the communication: not the “how”.

<table>
<thead>
<tr>
<th>User Thinking Aloud</th>
<th>Dialogue</th>
<th>System Thinking Aloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>I want to know whether the power is on.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is the power switch on?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I can’t see it</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I don’t know.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.5: Data streams in a Wizard of Oz experiment in the audio domain.

The tracing of problems is facilitated by the fact that there is no face-to-face communication between the domain expert and the user, because they are in different rooms. More importantly, this allows the recording of thinking aloud protocols of both
agents, in parallel with the interaction protocol. In Figure 5.5 an example fragment is presented of a dialogue between a domain expert and a user. Misunderstandings and problems – also in the communication – can easily be traced from these thinking aloud protocols. Another reason that these protocols contain a lot of information is that communication by typing in text is relatively slow, thus allowing the human participants to reflect on current states, histories and intentions.

Additional Ingredients

The relevant information for the design of a KBS that can be elucidated by this form of user analysis are:

Overall task conception of the user. This is where the user understands the (major parts of) the task decomposition of the task model. This knowledge is used to keep track of the current state of the cooperation and its requirements. This may be a problem particularly for novice users. For instance, in NEOMYCIN a task decomposition in the form of a tree is presented to the user, and continuously updated, i.e. has a transparent task structure (Clancey & Letsinger, 1984).

User conceptions of the task distribution. User and system can have different ideas of tasks with mutual user/system involvement. For example the user might have a different idea from the designer about what constitutes “teaching”, or “critiquing”.

Accessibility to information. One of the ingredients in the cooperation is information. User analysis may reveal whether the user really has access to the necessary information.

Domain knowledge and skills of the user. These are the ingredients that the user brings to the task. When the user does not possess the assumed knowledge and skills, he cannot perform the task that is allocated to him and he is therefore unable to satisfy the needs of the system. A second important role of knowledge and skills comes in where the user has to interpret or select external data before transferring them to the system.

Users may lack insight in the task distribution, may not have the required information, and may lack understanding about what the questions and answers of the expert mean; they therefore fail in their part of the cooperation. The system may ameliorate such needs by providing additional ingredients like knowledge, skill and information about the state. To specify these, the cooperation model can be extended with the appropriate transfer tasks. In our experiences, the result of a user analysis which evaluates the assumed model of cooperation is, in general, an extension of the list of the ingredients, rather than a new task distribution (de Greef & Breuker, 1989).

5.4 Chez Chef: an Example of Cooking Cooperation

The example presented here is to a large extent fictitious, but easy to understand because most readers will be familiar with the domain task.
A publishing company PC (Pressure Cooker) has published a database system containing recipes for a few thousand dishes. Marketing has revealed that this database is hardly attractive, because books are more fun and do not require expensive hardware. There is no reason to bring the machine to the kitchen, unless it can also be of use during the process of cooking. Here a knowledge engineering firm (SWING) is called in and presents a short orientation in the domain, which can be summarized as follows:

The major bottleneck in preparing a dinner is the detailed planning of the cooking actions, as every amateur cook will admit. The role of a chef in a restaurant kitchen, for instance, is mainly devoted to this type of task. The planning problem is a very complex one, because it should embody the cooking plans of a number of dishes. For each dish, cooking plans can easily be derived from recipes in cook books, but there are no books that allow for "weaving" these separate plans together, if only for the fact that the user wants varied instead of standard dinners. Expert cooks may do all sorts of hierarchical planning, i.e. they have a global plan in mind when they start cooking which is refined during execution. One of the major problems is that during cooking not only a complex plan should be kept in mind, but that it is also entangled with execution and monitoring tasks. Experienced cooks leave their plans sufficiently underconstrained to allow for minor adjustments. Moreover, their execution and monitoring skills do not put such a high demands on their processing capacity compared to amateur cooks.

Task Decomposition

In order to discuss the role of this bottleneck, SWING shows to the PC board a task decomposition of the "make dinner" domain. The making dinner task is easy to isolate, because it has a recurring nature (see Figure 5.6).

The first level of decomposition is relatively easy; the sub-tasks are successively: compose menu, obtain ingredients, make cooking plan, cook meal, serve meal and eat meal. At the next, more interesting level, these tasks are decomposed into two or more sub-tasks. Some of these tasks are split on the basis of a decomposition of the object. For instance, "compose menu" is split into the selection of a menu frame (e.g. a lunch, a four-course meal) and selection of dishes that can fill the slot. Similarly, the cooking plan is decomposed in the separate plans for each dish, and the configuration of these separate plans into an overall cooking plan for the meal.

At the more detailed level, the decomposition appears to become somewhat arbitrary: the sub-tasks at the first level are clearly separated. They can be easily distinguished because they are observable as behaviour, or can be easily inferred as a prerequisite task (e.g. making the cooking plan). However, the decomposition of making the cooking plan is far more complex, because we cannot observe "in-between products or states". The decomposition is based upon a decomposition of the object (plans/dish) and the constraints the plan should satisfy. The cooking itself may be easily observable and have many in-between products, but here a further task decomposition is not very interesting. First, because there can be a multitude of too detailed tasks. Secondly, the decomposition would exactly coincide with the outcome of the planning, and thirdly, this task is only interesting if we want to make a cook-robot. PC thinks this will be too ambitious as long as our kitchens have such a deplorable state of electronic interfacing capabilities!
Therefore, the task decomposition is an idealized one, i.e. based on object decomposition and "logical" task requirements. There may be alternatives (assume that these have been considered but rejected by "experts"), but the decomposition appears to be a sufficient one. It can be evaluated to some extent by the complexity of the dependencies: these are depicted in Figure 5.7.

Because the decomposition is an AND-tree, the dependencies aggregate upward, i.e. dependencies between sub-tasks are also dependencies between their governing tasks. Expressing all dependencies makes a messy picture; therefore, only the most specific dependencies are shown, not the implied ones at higher levels.

Task Distribution

The activity of specifying the task distribution begins with a specification of the user types. The question is not whether there are distinctive types of users (e.g. children doing the cooking vs. a housewife cooking for a party), but whether these distinctions are relevant for different task distributions. For SWING life is simple; there is only one type of user: the amateur cook (i.e. anybody who has the skills to cook a separate dish from the recipe in a cook book).

The task decomposition appears to be a good basis for the distribution of tasks. The allocation of tasks should be performed at the most specific level of the tasks. Figure 5.8 presents the task distribution.

Although the sub-tasks allocated to the system appear to be very diverse and spread over the full problem-solving domain (having dinners), the sum total of system tasks makes sense. First, the cooking plan planner should have the capability to understand a recipe in terms of activities. This means that such knowledge should either be represented explicitly in the data/knowledge base of recipes, or that it
Figure 5.7: Task decomposition with interdependencies.

Figure 5.8: Task decomposition with interdependencies and distribution.
should be inferred from the current database (e.g., based upon script-like or "skeletal" knowledge of typical cooking activities). Second: the monitoring support is conceived as a real-time instruction of what to do when. For instance, the CHEZ-CHEF may list successive instructions with time intervals specified, like

In 5 minutes: start cooking the haricots verts
In 6 minutes: control whether the milk is cooking: keep looking
In 9 minutes: put butter in frying pan / make sure hot milk is turned off, and immediately available.
In 10 minutes: control haricots verts: should be ready in 1-2 minutes

A number of (sub)tasks are allocated to both system and user. These tasks have to be decomposed further. The major reason we cannot specify these sub-tasks further (select dish, specify constraints, and monitor/diagnose) is because we do not know what these tasks may look like. This requires more empirical data about the user and the expertise. There are many possible solutions, depending on the amount of support the user may want and the complexity of the expertise. For instance, in selecting a dish, the system may either be a passive database with query facilities (much like it is now), or it may become actively engaged, and, for instance, check compatibilities, prices, complexity of dishes, or even make expert suggestions, like a real chef de cuisine would do.

The Cooperation Model

From the interdependencies between sub-tasks and the allocation of sub-tasks to "system" and "user" it follows which dependencies there are between sub-tasks performed by different agents. These interdependencies indicate physical objects or information objects that need to be transferred between system and user. The objects to be transferred in cooperation we call "ingredient" (Somewhat confusing because in the cooking domain the term "ingredient" occurs as well).

The next step is to specify the control of cooperation by assigning the initiative for the transfer of each ingredient to "user" or "system". Figure 5.9 provides this specification for a subset of the ingredients.

<table>
<thead>
<tr>
<th>ingredient</th>
<th>type of transfer task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of Dish</td>
<td>obtain</td>
</tr>
<tr>
<td>Ingredient</td>
<td>present</td>
</tr>
<tr>
<td>Constraints</td>
<td>obtain</td>
</tr>
<tr>
<td>Error</td>
<td>receive</td>
</tr>
<tr>
<td>Cooking Plan</td>
<td>present</td>
</tr>
</tbody>
</table>

Figure 5.9: Specification of the Cooperation Model.

This cooperation model now can be tested by hand-simulating it in "Wizard of Oz" experiments with future users. One would have to flesh-out the model for a few example problems (menus and dishes) and one would have to invent an arrangement for conducting "Wizard of Oz" experiments with one terminal in the kitchen. These
experiments may show that the roles of user and system as specified in the cooperation model constitute a working man-computer system, but it may also turn-out that the user needs additional information or knowledge, or that the allocation of initiatives should be revised, or (not likely) that the task distribution must be modified.

5.5 Conclusion

KBS are usually conceptualized as active agents rather than passive slaves. Therefore one should first make explicit what the intended cooperation between the user and the artificial agent(s) should be, before one specifies and designs the user interface. In our view the communication between the user and the KBS is primarily a function to support this cooperation. The top-down framework for analysis of cooperation (see Figure 1) appears feasible. This functional view has the advantage that user-interface specification becomes less dependent on the arbitrary collection of user's wants, wishes and needs in more-or-less structured interviews. The model of cooperation can be used as a model that guides the collection of empirical data. These data also enable the validation of the assumptions of the model. Therefore, the top-down framework provides an effective alternative to current practice, that may consist of some combination of open ended inventory of user's statements and testing a fully running prototype.

The specification of the “what” (content) and “when” (control) of the required communication between user and KBS are also viewed as a further refinement of decisions about the task distribution between user and system, and of the specification of the control layers of the KADS model of expertise (Wielinga et al. 1992). The model of cooperation is completely independent of the medium for expression. This fits well with the general approach in KADS to leave commitment to specific to physical realization (implementation) to later stages of design. In this respect, the analysis of cooperation does not replace the many guide-lines in the literature for user–interface design.

The analysis of cooperation interfaces seamlessly with the KADS major modelling activities in the analysis stage. It also shows the performance of activities in parallel with the development of the KADS model of expertise. However, it can also be used as an independent methodology for designing cooperative user interfaces. Indeed, some software companies in the Netherlands have used this approach to design user interfaces without explicit use of KADS even for conventional software.
Acknowledgements  The research reported here was partly funded by contracts P1098 (KADS) and P5248 (KADS-II) of the ESPRIT programme of the European Commission. The contributions of the UvA- KADS team, Ton de Jong, and in particular Joost Breuker, who had an important role in the development of this framework, are gratefully acknowledged.