

# 1 Computers as Assistants

## Introduction and Overview

*Peter Hoschka*

"The next major metaphor shift in computing will be toward programs that act like assistants rather than tools: they will show more initiative, assume responsibility for larger subtasks, and take appropriate risks." This is the summary of a Microsoft Research project of summer 1994. In a report of the American Association for Artificial Intelligence (AAAI) for the U.S. Congress of August 1994, the presidents of AAAI, Barbara Grosz and Randall Davis, identified "Intelligent Associates: Smart Collaborators" as one of four grand challenges to future research: "To realize them, a common set of capabilities for intelligent systems need to be developed, like... reasoning about the task being performed by each system..., learning from previous experience and adapting behavior accordingly, collaborating..." In July 1994 the *Communications of the ACM* devoted a special issue to "Intelligent Agents," which discusses the basic ideas, views, and insights in this field of research.

Obviously, the possibility of building computer support systems capable of assistance is a compelling one for computer scientists. At GMD, the National Research Center for Computer Science in Germany, a large-scale research program on assisting computers (AC) was begun as early as 1988. The goals of the program were to study and develop the principles and methods for assisting systems, and to demonstrate assistance properties and capabilities in various prototype systems.

Research on computers as assistants means looking for new ways of dividing the labor between humans and computers. On the one hand, future systems should take on more tasks than existing systems do, especially those that seem tedious or difficult for humans. On the other hand, they should *not* automate tasks completely. The basic paradigm is that of assistance. In many fields of application the problems are either too complex or simply too

numerous for any attempt to develop a machine with complete problemsolving competence to succeed. What is called for instead is a set of calibrated tools that the user can combine, adapt, and employ as he or she sees fit. Exhaustive treatment and coverage of a problem is in fact *not* the goal of computers as assistants.

There are many characteristics of effective assistance. A human assistant, for instance, is expected to be competent in his or her domain of expertise, to know his or her limitations, to be able to process inexact instructions, to adjust to a client and to learn from the client, and to be able to explain his or her behavior and suggestions. The facilitation of communication and cooperation is a central function of an assistant.

If computer systems are to offer assistant capabilities, they must be supplied with domain knowledge and knowledge about the user. There is an additional requirement: Computer systems need knowledge about themselves, that is, they need to know about the way they function. Only if a system can observe its own behavior, and reflect on it, will it be able to correctly evaluate its own competence and explain its behavior.

The AC concept does not entail building a duplicate of a human assistant. What we tried to do in the AC program was replicate some of the functions of good assistance in computer systems-with no claims to cognitive adequacy. The most important assistant properties we studied in the AC program were the following (Fig. 1.1):

- Domain competence: Assisting computers should be equipped with domain knowledge in certain areas of importance to their users; they should be able to support problem-solving processes in these areas.
- Competence assessment: Within their domain, assisting computers should be able to assess their own competence and their limitations. The user should be able to engage in a dialogue with the system to find out which problems it can solve, which not, and why not.
- Learning and adaptive behavior: Assisting computers should be able to adapt both behavior and functions to a user's individual needs and personal style. The system should learn from the user by monitoring and analyzing his work.
- Processing imprecise instructions: Assisting computers should be able to interpret incomplete, vague, ambiguous, and even contradictory instructions on the basis of knowledge about the user and the current task.

- Explaining abilities: Assisting computers should be able to explain and give reasons for each of their actions, conclusions, and suggestions-in terms the user can understand.
- Cooperation support: Assisting computers exist not only to support the isolated work of individuals, but also the work in teams and organizations. They should help coordinate tasks of a group and provide the organizational knowledge required for cooperation and coordination.

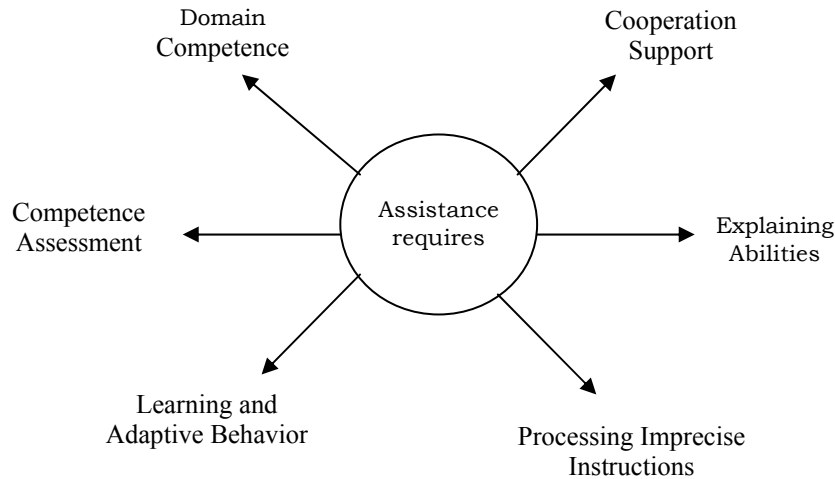


FIG. 1.1 Fundamental assistance properties.

The following chapters describe the concepts and methods we have used in our endeavor to equip systems with assistance properties and capabilities, and the results we attained. This introductory chapter gives a short overview.

## COMPETENCE ASSESSMENT

According to the basic paradigm of computers as assistants, we do not try to build systems with exhaustive and perfect competence. But if systems are not perfect, they should be able to assess their own competence. Competence assessment means judging the potential ability of a system to solve a particular problem. More specifically, competent behavior means:

- Before jumping into the solution process, a competent system checks whether it understands the problem, which might be incomplete or ambiguous.

- It does not try to tackle problems that are unsolvable in principle, for instance, if the problem statement is in itself contradictory.
- Nor does it try to solve problems surpassing its capabilities and resources. If necessary, it is able to negotiate the problem statement.
- A competent system is able to detect and remove redundancies and can therefore reduce complexity.
- It monitors its progress and changes focus by adapting strategies or redistributing resources in case progress is behind expectations.
- Finally, it evaluates its solutions in retrospect, so as to store its problemsolving experience. Later, when confronted with a similar case, decisions can be made based on this experience.

Competence assessment requires stepping back and viewing the system from the outside in order to detect its malfunctions. Hence, competence assessment may be regarded as reflective behavior-the inspection of the system by itself. In order to represent the system itself, a reflective system has a meta-level architecture; that is, the part of the system that is reasoned about is represented in a more abstract model at the meta-level (the self-representation). Reflective components inspect, observe, and possibly correct their "object" system from a higher, meta-level. They do not operate on the actual implementation of the object system but on an abstract model of the underlying system. They are thus generic and applicable to a broader class of object systems.

According to this scheme, we built several components for the class of assignment and constraint satisfaction problem solvers. They can recognize overspecified and overcomplex problems, cope with redundancies and contradictions, decompose complex problems, schedule problem-solving steps with respect to time limitations, and switch to a propose-and-revise strategy in case constraint satisfaction fails. A detailed discussion of these problems and solutions can be found in chapter 2, section 2.1.

## **ADAPTIVE AND ADAPTABLE SYSTEMS**

An important requirement for assisting systems is the ability to adapt to users' individual styles and tasks. Two forms of adaptation should be distinguished here: one that occurs on the user's initiative (adaptability), and the other initiated by the system itself (adaptivity or, better, auto-adaptation). Although the call for adaptability is uncontroversial in principle, there are differing positions on auto-adaptation of systems. Our own research has

shown that in practice there are only few opportunities for auto-adaptive services. Those that have been implemented so far (adaptation of parameter defaults, control of dialog queries, offering of abbreviated commands) are based only on the evaluation of simple frequencies in user behavior.

System adaptivity is especially desirable in the offering of help when the user is having difficulties. Help facilities should be adjusted according to the current dialogue situation and the individual user. We have developed such a context-sensitive help system for the spreadsheet program Excel. Our HyPLAN system consists of two modules, a plan-recognition program and an interactive multimedia help environment. The plan recognition unit gets a continuous input protocol, listing the commands entered by the user. Guided by a knowledge base of hierarchical action nets, dynamic state models of the user's probable goals are instantiated and incrementally extended as new protocol data comes in. When the user calls for help, the system selects a context-specific help offering based on the goals currently marked. The offerings themselves are realized as voice commented illustrations and animated scenes.

Empirical studies have shown that the possibilities of user-initiated system adaptation are not exploited nearly as intensively as system designers intend. We concluded that the proper exploitation of adaptability requires special support by the system, and thus we looked into the possibilities of using an auto-adaptive component to point out to the user the adaptability capabilities of a system. With FLEXCEL we have devised some new ways of helping the **user** to utilize the adaptation features of a system. The example application again was Excel. In the meantime, our solutions were incorporated by Microsoft in the new version of Excel (cf. chapter 2, section 2.2).

## PROCESSING INEXACT KNOWLEDGE

Assistance systems should be able to perform meaningful actions even if the instructions or the available pieces of information are imprecise and the action to be executed is not definitely prescribed. In this context, imprecision means:

- The available information can be incomplete, so that decisions must be based on plausible additional assumptions (defaults), which may have to be revised later.
- The available information can be vague or ambiguous, for example, because certain relationships are true only with some probability, or

because terms used in an instruction have no precise meaning (draw a triangle *near* the rectangle).

- The available information can be contradictory, which means that parts of it must be faded out and left out of consideration.

As an illustrative field of application, we took the problems of processing imprecision during the construction, editing, and search of graphics. The user can incrementally specify the attributes of graphical objects. The assistance function consists of completing or making more precise and consistent user specifications that are imprecise in the foregoing sense. The methods we used here were mainly concentrated on nonclassical methods of inference, such as nonmonotonic (i.e., based on revisable assumptions) and associative reasoning (cf. chapter 2, section 2.3).

Nonmonotonic inference mechanisms are of interest because they allow the assisting system to complete imprecise problem specifications with the help of standard assumptions. Default rules make it possible to infer plausible conclusions, which can lead to meaningful decisions in the case of incompleteness.

Associative reasoning techniques based on neural networks are used for the storage and retrieval of known problem specifications and solutions. The solution of a current problem is supported by looking at similar, already solved problems. These are reactivated in a content-driven, associative manner (cf. chapter 2, section 2.3 and chapter 3, section 3.5).

## EXPLANATION ABILITIES

If assisting systems process imprecise instructions, make suggestions for system adaptation, and offer domain competence for solving problems, then these capabilities inevitably give rise to a further requirement: The systems must be able to explain their own behavior and suggestions-in terms the user can understand. Assisting systems have no chance of being accepted as black boxes. It is well known of existing expert systems that the explanation components actually explain very little. They are usually limited to confronting the user with a more or less flexible presentation, giving the protocol for each of the the problem-solving steps executed. Explanation ability in the assisting computer sense, furthermore, entails not only making a system more transparent, but also implementing pedagogical competence so that the user's understanding of a problem can be estimated and improved in the course of dialogue. Consequently, a system must have tutorial faculties, and

not be limited to simply showing the formal structures of a knowledge-based system.

As a first step toward better explanations, we worked on improving knowledge representation methods so as to make them more suited to explanation purposes. We examined the possibilities of using conceptual models that represent knowledge in several levels of abstraction to determine the focus and level of detail for an explanation to be produced (cf. chapter 2, section 2.4).

## **SYSTEMS WITH DOMAIN COMPETENCE**

Supplying systems with domain knowledge is the goal of conventional expert systems. A number of examples that deviate somewhat from the usual expert system scenario show how assisting computers can incorporate some of the knowledge that must, at present, be supplied by the user. The major difference between conventional expert systems and assisting computer systems involves the roles of the human and computer. Most expert systems ask the user for input, make all decisions, and then return an answer. In assisting computer systems, the user is an active agent empowered by the system's domain competence. This means that the systems and the person are bringing complementary strengths and weaknesses to the job. Rather than communicating with computers, users should perceive themselves as communicating with application domains via computers. To shape the computer into a truly usable and useful medium, we must let users work directly on their problems and their tasks.

We have created (chapter 3) five paradigmatic assisting systems with domain knowledge:

- An assistant for knowledge acquisition that supports the development of a knowledge base (section 3.1).
- A statistics interpreter that helps to detect interesting findings in statistical databases (section 3.2).
- A graphics designer that assists the design of business and presentation graphics (section 3.3).
- A design assistant and a graphical search tool that use case-based memory and principles of gestalt theory to support the reuse of graphical objects (sections 3.4 and 3.5).

- A user interface design aid tool that assists the designer of graphical user interfaces during the design process (section 3.6).

All these systems incorporate domain competence. They also, however, contain elements of other assistance properties. The knowledge acquisition system, for instance, is able to learn from the user and identify and manage imprecise instructions. The graphics designer system can also process imprecise instructions: The user develops a graphic to a certain point and then passes on this raw version to the system, which takes over the details of producing an aesthetically pleasing end product.

### **Assistant for Knowledge Acquisition**

A system to support the acquisition of knowledge cannot be limited to offering the user some formalisms for the representation of knowledge. Acquiring knowledge in a new domain is not a straightforward task that can be easily planned out for computer treatment. It is a creative process with frequent mental leaps and many iterations. The system must be flexible so that the user can work in his or her own manner and is not forced to adapt to the system's. The system must be able to supply the user with the consequences of new knowledge that the user adds to the system. Since a domain model is built in the course of work with the system, it must be able to accept contradictory, incomplete, and preliminary inputs. We call this the paradigm of "sloppy modeling" (Morik 1989). Finally, the system should help the user complete and rectify the model, for instance by suggesting new rules and concepts to supplement the existing knowledge. These requirements have guided the conception of the knowledge acquisition system MOBAL (model-based learning system).

MOBAL is a workbench offering several tools for knowledge acquisition that can be used individually or in combination. The heart of the system is the Coordinator & Inference Engine, which manages the knowledge base and coordinates the system's various functions. MOBAL uses facts and rules to represent a domain model. Existing facts and rules can be modified at any time; the inference engine makes sure that entries relying on the changed ones are modified as well. If entries are incomplete or erroneous, they do not have to be corrected immediately. The system maintains an agenda of such "open ends," and the user can address these at any time he or she sees fit. Even contradictory entries can be processed; the system recognizes contradictions and offers a special tool, the Knowledge Revision Tool, with the help of which a contradiction can be analyzed and resolved.

The other tools that comprise MOBAL serve the purpose of supporting the user with various machine learning techniques. They produce suggestions that can help resolve incompleteness, improve knowledge-base structure, or recognize and point out regularities in the facts. The user can use these suggestions or decide to make similar (or other) entries. All machine learning techniques in MOBAL play the role of an assistant to the user. Modeling is understood as a cooperative and balanced process between human and computer.

### **Statistics Interpreter**

A second example of an assistance system with domain competence is the statistics interpreter Explorer. It discovers interesting findings in statistical data and aide the user in interpreting them. The system hers knowledge about the domain and about methods for the analysis of statistical data. To this end, the objects, relations, and queries used in the statistical analysis of a data set must be represented in the system. From these, the system constructs a proposition space for these data and searches it for interesting findings. In traditional statistics packages, the user must specify each hypothesis individually. In Explorer, this is different: Knowledge-based and systematic processing of the proposition search space allows the system to discover results that might have been overlooked in traditional analysis. The statistics interpreter usually hers to cover a very large search space of potential findings. However, this space can be pruned substantially with the help of redundancy filters and generalization methods. This assures that only the "strongest" findings are presented to the user and the user is not flooded with redundant statements.

Explorer accepts propositions of an arbitrary type which are useful to formulate knowledge about data and incorporates them in a "discovery system." In other words, the system is not limited to specific types of hypotheses (as, for instance, rule learners are). Explorer also supports the user in navigating through the space of potentially interesting findings by being able to refine, specialize, condense, and generalize propositions. Explorer is an early (and working) system in the area discussed nowadays under the headings of data mining or knowledge discovery in databases.

### **Graphics Designer**

A further example of success in incorporating domain knowledge in a computer system is the graphics designer. Today's graphics packages allow anyone to produce graphics easily and comfortably. Current systems do not,

however, offer any assistance where aesthetic beautification or the choice of suitable graphical means of expression is concerned. It is here that the graphics designer aims to provide assistance.

In creating graphics, certain rules of design that govern expressiveness must be observed. These rules state, for instance, which diagram type is suited for time series, and which for rank orders, or which colors make a good background and which a good foreground. The layperson is usually unfamiliar with such rules and sometimes they elude even professionals. We have developed a system that helps the user in producing a suitable business graphic.

In the fine calibration of a picture, graphical elements must be aligned to each other and spread out evenly across the available space. The more precisely a graphic can be printed, the more conspicuous discrepancies in layout and slight inaccuracies in size and position become. The graphics designer can describe typical errors in graphics in a "situation language" we have developed, and can automatically find these errors. A knowledge-based criticism module decides which corrections are to be made in which order, and plans out the individual steps in such a way as to ensure that later corrections do not reverse previous ones.

### **Graphical Search Assistance**

More and more products are designed with graphical editors, such as simple drawing tools, CAD (computer-aided design) tools, or graphical programming language editors, and are then archived electronically. The logical next step is to reuse stored designs for other products, either as a source of inspiration, or directly via copy, paste, and adaptation. Retrieval of archived designs is a problem if there is no foolproof filing scheme and if one does not exactly know what to look for. One may be forgetful, designs may have been produced by other persons, or designs from different contexts can be mistakenly appropriated.

Here, the most suitable support would be an assistant that can semantically interpret graphical queries and perform a so-called content-based retrieval. Such an assistant could be given a sketch and asked to retrieve something similar and hopefully more complete. One challenge in interpreting such fuzzy queries is to detect structures and forms that were not explicitly drawn, so-called emerging shapes. Techniques for that purpose often rely on principles of gestalt psychology, such as the one presented in chapter 3, section

3.4. Another challenge is to find a fuzzy notion of similarity, which typically is domain dependent. Two such techniques are discussed in chapter 3, section 3.5. All approaches have in common that design documents need not be indexed manually, because the same mechanism that automatically interprets a query can also interpret the designs archived. Thus, they free the designer from boring work in order to concentrate on the more creative aspects.

### **User Interface Design Assistant**

In the effort to provide high ergonomic quality graphical user interfaces, designers have acquired knowledge about human factors and expressed it in standards, style guides, and guidelines. The volume of these available sources is already huge. Simultaneously, user interface designers need more and more competence, knowledge, and experience to handle this great amount of human factors knowledge. This means for most of them that optimally performing their jobs requires taking into account far more information than they can possibly keep in mind or apply. This results in a need for user interface development tools with domain competence based on human factors knowledge that may be encountered, learned about, practiced, and improved during ongoing use—in other words, tools with which users learn and use on demand. An important research goal in the development of user interface tools is, thus, to discover helpful, unobtrusive, structured, and organized ways to integrate human factors knowledge into the tools without stifling creativity—and to provide the designer with assistance in understanding, searching, and applying this knowledge. This goal was the starting point for the project IDA (User Interface Design Assistant). The primary purpose was to explicitly incorporate domain competence in user interface development tools to empower the user interface designers. The following benefits for user interface designers using the IDA development tools are expected:

- Designers will be able to acquire human factors knowledge during their daily work using their development tools ("learning and using on demand").
- Designers will be enabled to apply ergonomic style guides and guidelines ("usability").
- Designers will be able to use predefined ergonomic user interface components ("reusability").
- Designers will be able to evaluate the ergonomic quality of their design during the design process ("quality assurance").

## **SUPPORT OF COOPERATION**

Every form of activity in organizations requires cooperation among its members. Assisting computers should therefore not only support the isolated work of the individual but also help coordinate activities within and across groups, and provide access to the necessary knowledge about the organization. In the AC program, these functions of assisting computers were analyzed and prototypically implemented in the following components (chapter 4):

- A task manager that helps coordinating group work (section 4.1).
- A mediating system for the support of discussion, argumentation, and decision-making in groups (section 4.2).
- An organizational knowledge assistant that offers information about structures and procedures of organizations (section 4.3).
- Experiments with new metaphors to effectively present complex cooperation support environments (section 4.4).

### **Coordination Support by Task Management**

As a contribution to the effort to provide more effective computer assistance for cooperative activity, we have developed the Task Manager, which is especially oriented to supply coordination support for geographically distributed work situations typical of large business organizations and government agencies.

With the help of the Task Manager, users may organize cooperative tasks, monitor their progress, share documents and services, and exchange informal notes while performing their tasks. The Task Manager distributes task specifications, attached documents, and notes to the involved users in a consistent way. It is meant to support the management of work distributed in time and/or space by providing:

- Support of organization and planning of collaborative work (who does what, with whom, until when, using what material).
- An up-to-date overview of collaborative activity and work progress.
- Dynamic modification of work plans during performance.
- Availability and exchange of documents and informal notes within groups of people involved in task performance.

### **Mediating System for Argumentation and Decision Making**

Another cooperation support system that we developed is a "mediating system" for supporting discussion, argumentation, and decision making in groups over electronic networks, the Zeno system. The system is designed to take resource limitations and conflicts of opinion and interest into account. It is a contribution to what might be called "computational dialectics," the study of computational models of norms for rational discourse.

The thesis is that rationality can best be understood as theory construction regulated by discourse norms. Techniques from artificial intelligence are applied for supporting practical reasoning given such pragmatic constraints as limited time and uncertain, incomplete, or even inconsistent information. To avoid the rigidity inherent in formal models of norms and to facilitate an acceptable balance between the individual interests of autonomy and management interests of control and accountability, the space of possible actions is kept cleanly separate from the space of permissions and obligations. That is, our mediating system advises participants in a discussion about their rights and obligations, but does not automatically enforce rules. Using the legal system as a model, Zeno distinguishes between the legislative and judicial functions. Participants have an opportunity to argue about whether some rule should be applicable in a concrete case; the norms evolve and adapt through use.

### **Organizational Knowledge Assistant**

Cooperation in teams and organizations is embedded in an organizational framework. Thus, information about the organizational context in which users work helps to choose the right patterns for communication and cooperation. Information must be provided to answer questions such as: Who is responsible for carrying out a specific task? Whom can I ask for help? Furthermore, the system should provide information as to how particular tasks are handled in the organization. What are the organizational rules one has to consider? Whom do I have to ask first? Which document type do I have to use? All this information is part of the knowledge that is normally not or only very implicitly provided by cooperation supporting applications, although it plays a significant role in cooperation. We developed a system, named TOSCA, that provides this information to users and applications.

The organizational knowledge assistant aims at two goals. The first goal is providing the organizational knowledge required to ease the integration of cooperative applications in an organization. In the service of this goal, it

offers well-defined interfaces and supporting distributed management. In combination with a user interface it is meant to cater to the informational needs of the end user.

The dynamic nature of any organization makes it impossible to develop a single representation that fits all organizations. Thus the second goal is ensuring "tailorability" that allows an adaptation of the system to various organizational settings. TOSCA heightens the visibility of the relevant concepts and provides an object modeling tool that allows users and groups to tailor the object model to their specific needs.

### **Dynamic Interfaces for Cooperative Activities**

The success of computer-supported cooperative work depends to a large part on the quality of the user interface. User interfaces for cooperative applications are demanding because the environment is no longer static and the user has to be aware of other people's actions. This is a significant departure from the single-user desktop metaphor. New metaphors as well as innovative techniques (such as animation) are required to effectively present the complex working environment on a small computer screen.

The DIVA (Dynamic interfaces for cooperative activities) project has identified the following three key problems to improve the interfaces for cooperative applications:

- A new metaphor is needed for cooperative work. The conventional desktop metaphor is no longer sufficient, because the private desktop must be expanded to a space of cooperating people.
- The concept of time becomes more important. In particular, users must be supported by a history of activities because it is not possible to memorize other people's actions.
- Cooperative activity must be visualized. New mechanisms, including three-dimensional structures and animation techniques, are required to provide the level of awareness necessary for successful telecooperation.

We have developed a prototype that tries to mimic on the screen the social behavior in a real office, to facilitate communication between users and the use of cooperative applications. We experimented with a first version of the "virtual office" metaphor that represents the environment as people moving between rooms in a virtual building.

## **METHODOLOGICAL AND TOOL PROJECTS**

The overall AC effort was planned to consist of many components—the assistants—with consistent user interfaces designed according to uniform guidelines. Therefore, a common development base for all projects was defined: the UNIX operating system, the X Window System, the OSF/Motif graphical user interface, and LISP/CLOS (Common LISP Object System) and C++ as programming languages. However, it soon became clear that this decision alone was not sufficient to guarantee consistency, and therefore the project GINA was created in order to define guidelines and to incorporate them into a software tool to be used by those involved in the projects implementing assistants. In chapter 5, section 5.1 we give an overview of GINA.

In the context of the AC program there were several additional projects that focused not on the development of specific user-oriented functionality but more on the solution of basic methodological problems. Thus, we intensively studied the potential of artificial intelligence planning for building assistance systems. Planning can be both a purpose and a means in assistance systems: A purpose whenever the task requiring assistance is a genuine planning task, and a means whenever an assistance task or property requires some internal planning. Section 5.2 contains the results of our work on planning in this context. In section 5.3 we investigate the instrumental use of guiding visions and metaphors, such as the assistance metaphor, for designing computer systems and for preventively oriented technology assessment.

## **CONCLUSION**

Work guided by the concept of computers as assistants began in 1988. The results reported in this book show that we have come closer to our goals and have been able to demonstrate, at least in sample fashion, some of the assistance capabilities we aimed at. New methodological approaches to a number of problems have been found. But for the most part, we are still in the early stages of progress toward our goals. Therefore, we are pleased that so many other research groups have also started work on computers as assistants.

## **ACKNOWLEDGMENT**

A large number of people were involved, directly or indirectly, with our AC research program on assisting computers. Above all I would like to thank all the members of the AC program for their support and constructive work. Particularly the project leaders Franco di Primio, Klaus Kansy, Willy

Klosgen, Thomas Kreifelts, Katharina Morik, Reinhard Oppermann, Horst Santo, Michael Spenke, August Tepper, and Angi Vo13 always supported the basic idea of building computers as assistants. I would like to thank my colleagues Thomas Christaller and Peter Wil3kirchen for their support in directing the AC program; particularly Peter Wil3kirchen always was there with help and advice when problems and impediments arose. Last but not least, I am extremely grateful to Klaus Kansy for his invaluable assistance in the editorial work for this book. Erich Rome has put a lot of effort into polishing the references. Gabi Vezzari patiently kept the editorial process running and helped with the subject and author indices. I gratefully acknowledge the help of Lawrence Erlbaum Associates, particularly of Susan Milmo, during the publishing process.