PLUS Plan-based User Support

Final Project Report

Frank Berger, Markus A. Thies, Wolfgang Wahlster
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3
W - 6600 Saarbrücken 11, Germany

Thomas Fehrle, Kristof Klöckner, Volker Schölles
IBM Laboratory Böblingen
Schönaicher Str. 220
W - 7030 Böblingen, Germany

Abstract

This paper presents the results of the project PLUS (**Pl**an-based **U**ser **S**upport). The overall objective of PLUS was the design and the implementation of a plan-based help system for applications that provide a graphical and direct-manipulative interface.

The design of graphical user interfaces is based on the principle that *"the user is always in control"*. This means that the user is responsible for performing his tasks according to his own strategy. This leads to a great degree of flexibility in task execution as opposed, for instance, to menu-oriented user interfaces. Usually, neither a definite sequence of interactions nor a fixed number of actions are required to accomplish a specific task. In addition, modeless user interfaces allow the user to work on different tasks in parallel and to arbitrarily switch between them.

Within the project PLUS we developed various help strategies, including graphical representation of the current interaction context, tutoring modes, and animated help, to support novice and occasional users during their work with applications that provide graphical user interfaces.

Contents

Foreword

PLUS belongs to a new generation of user interfaces which possess some understanding of what the users are trying to do, and how they need to go about doing it. An intelligent user interface like PLUS mimics some of the key capabilities of a human assistant: observing and forming models of the user, inferring user intentions based upon those observations, and formulating plans and actions to help the user achieve those intentions.

The results reported here grew out of an effort to determine whether plan-based help technology can survive outside the research laboratories.

The gap that exists between research and development needs to be bridged if innovation is to be achieved. One of DFKIs challenges is finding new ways to spin research results into new software developments of its shareholder companies.

For us at the DFKI, the PLUS project is a showcase effort of teaming applied research and development in order to speed up the technology transfer process. PLUS is also an excellent example of what we call a tandem project at DFKI, i.e. an application-oriented project that exploits results from more basic research in another strategic DFKI project funded by the German Ministry for Research and Technology (BMFT). I was very pleased about the fruitful interaction and cross-fertilization between the PLUS project and the PHI (Plan-based Help Systems) project which is sponsored by BMFT.

Transferring technology between a research organization like DFKI and a development lab like IBM Böblingen Software Systems requires a concerted effort along many dimensions.

Special thanks to Volker Schölles and Dr. Thomas Fehrle from IBM for making our journey through this technology transfer process an enjoyable one.

I would like to thank Markus Thies and Frank Berger from my research division at **DFKI,** who did a tremendous job meeting all the deadlines for the various milestones and finally delivering a piece of software, which surpassed the expectations of the industrial partner and pleased the sponsors. I would also like to thank Dr. Kristof Klöckner and Dr. Teufel from IBM for their excellent management and support of this project. Finally, I owe a great deal of gratitude and appreciation to Prof. Endres and Prof. Glatthaar from IBM, who initiated this fruitful collaboration and fostered a sense of technological excitement about the project inside their company.

I think that the PLUS project was a breakthrough in making plan-based help systems a demonstrable technology ready for widespread application.

Prof. Wolfgang Wahlster

Preface

At the end of this year a fruitful cooperation between the German Research Center for Artificial Intelligence (DFKI) and IBM Böblingen Software Systems came to an end. As manager of the department participating in this partnership I would like to look back at the past two years and give a brief assessment of its importance to us.

From the PLUS project we expected an exploration of context (i.e. task) sensitive help for direct manipulation user interfaces, a problem that came to our attention in usability evaluations of system management applications with graphical frontends. Consequently, our people from advanced software development, human factors and product development took part in this joint effort.

We chose the DFKI as a project partner because of its excellent reputation in knowledge based user interfaces due to prior work by Prof. Wahlster and others on plan-based help systems. Therefore we felt, we could expect a significant transfer of technology to the lab. Our expectations were surpassed, even if ultimately no direct introduction to a product could be achieved.

All technical project goals were achieved on schedule and additional aspects that came up during the investigations (like animation or tutor support) were also able to be covered. In retrospect, this success is due to a great extent to a development process of iterative refinement of prototypes which was facilitated by an object oriented methodology. Being able to demonstrate the power of the plan-based approach through early prototypes was helpful in converting initial scepticism in the product areas into enthusiastic support.

The experiences gained with the PLUS project have been influential beyond the immediate project context, both within the lab and without. Several publications as well as demonstrations and presentations within the IBM technical community and 3 masters' theses attest the scientific success.

I wish to thank the project participants Markus Thies and Frank Berger from the DFKI and Volker Schölles and Dr. Thomas Fehrle from IBM who have set an excellent example of cooperation between advanced product development and applied science. I would also like to thank all supporters who made this project possible, Prof. Glatthaar, who provided additional funds from IBM Germany, Dr. Teuffel, the first IBM project manager and especially mentors Prof. Endres and Prof. Wahlster.

Dr. Kristof Klöckner, Mgr. End User Products Development 3

1 Overview

The project PLUS (PLan-based User Support) was a joint project between the IBM Laboratory Böblingen, the IBM Germany GmbH, and the German Research Center for Artificial Intelligence (DFKI), Saarbrücken. PLUS was carried out from 1 October 1990 to 31 December 1992.

The following research scientists were involved in the PLUS project:

- Prof. Dr. Wolfgang Wahlster (project leader DFKI)
- Dr. Thomas Fehrle (initial project leader IBM Lab)
- Dr. Kristof Klöckner (following project leader IBM Lab)
- Dipl.-Inform. Frank Berger (DFKI)
- Dipl.-Inform. Volker Schölles (IBM Lab)
- Dipl.-Inform. Markus A. Thies (DFKI)

There has been a close and very productive cooperation between the two groups at the DFKI and at the IBM Laboratory. Results from the work were frequently exchanged during periodical meetings held alternately at the DFKI and at the IBM Lab. In addition, further information and code was exchanged as required via Internet. Intermediate results were examined twice by a review board consisting of members from the three joint parties. The first review took place in May 1991 at the IBM Lab, the second review was held in December 1991 at the DFKI. With regard to the planned integration of the PLUS System into Screen View, a code inspection concerning the quality of the produced Smalltalk code was conducted in December 1991 at the IBM Lab (cf. section 5.1.2). Within the periodical SAB1 Review at the DFKI, the project PLUS was reviewd four times and it constantly received a very positive feedback.

The following resources with regard to the hardware and software environment have been provided by IBM:

Hardware: IBM PS/2 Model 80 workstations with 6 (initally) to 10 (final stage) MB main storage.

Implementation: Smalltalk V/PM, an object-oriented programming environment running under OS/2.

Design Rules: IBM's SAA/Common User Access (cf. [IBM 91]).

First Application Domain: HCD (cf. [IBM 92a]), a hardware configuration tool running under Screen View[2]. HCD was developed at the IBM Laboratory Böblingen.

Second Application Domain: The Screen View sample application OrgChart which displays the organization of an enterprise (cf. [IBM 92b], pp. 79-86).

[1] The Scientific Advisory Board is composed of well-known international research scientists.
[2] ScreenView is a set of services aimed at the development and running of applications with a consistent user interface (cf. [IBM 92b]).

2 Objectives

The overall objective of the PLUS project was the design and the prototypical implementation of a plan-based help system[3]. Rather than carrying out basic research, the state-of-the-art methods in several fields of Artificial Intelligence including *Knowledge Representation* and *Plan-based Systems* should be incorporated. Unlike previous help systems that were mostly developed for command language environments (see, e.g., [Finin 83], [Fischer et al. 85], [Wilensky et al. 88], [Wahlster et al. 93], [Bauer et al. 91]), PLUS was designed to cope with applications which offer graphical user interfaces (GUI), whose main interaction principle is based on a user-directed dialog by means of direct manipulation — so-called **D**irect **M**anipulation User **I**nterfaces (DMI) (cf. [Shneiderman 83], [Shneiderman 87]).

The design of graphical user interfaces is based on the principle that *"the user is always in control"*. This means that the user is responsible for performing his tasks according to his own strategy. This leads to a great degree of flexibility in task execution as opposed, for instance, to menu-oriented user interfaces. Usually, neither a definite sequence of interactions nor a fixed number of actions are required to accomplish a specific task. In addition, modeless user interfaces allow the user to work on different tasks in parallel and to arbitrarily switch between them. The flexibility provided by these graphical user interfaces from a human factors point of view, makes the use of software products easier on the one hand but more difficult on the other hand depending on the user type. It will be easier and more productive for an expert user to work in such an environment. Novice and occasional users, however, may easily get confused and they need assistance in performing their tasks. Usability tests conducted in this area have shown that test participants, who are traditional host users, need advice, in order to work with objects, actions, views, and settings in an object-oriented user interface. Available online information could not be used to solve their problems, because

- by presenting help information using hypertext information is split into units, which are too small,
- static help information does not take into consideration the current system state or the previously performed user actions, and
- textual help is not adequate in presenting information concerning the dynamic behaviour of graphical user interfaces.

Rather than asking for static offline (i.e., manuals) and online help, the user might wish to ask an experienced colleague for advice. Plan-based help systems satisfy the user's need for task-oriented help, which is generated at runtime in order to reflect the current dialog context.

We wished to fulfill the following aims with the PLUS System:

1. **Offering help which reflects the current dialog context and system state**
   User actions are mapped to typical user tasks, hypotheses of intended user goals are formed, and sequences of actions to reach these goals are deduced and presented to the user.

[3] See [Fehrle 90] for the initial project description.

2. **Increasing the acceptance of online help**
   The acceptance of online help is quite lowly rated by its users. In general, they miss out on a short and dear solution to their present problem, a solution which can be offered by PLUS.

3. **Offering suitable help in graphical user interfaces**
   Graphical presentation and/or animation is the best way of explaining how to use graphical user interfaces. People tend to deal more and more with other media rather than text.

4. **Reducing the effort of learning**
   Users are curious. They wish to run software immediately after installation and without reading manuals. Plan-based help systems act as an aid to this behaviour of exploring and the process of learning by doing.

The following help strategies should be incorporated into PLUS, in order to meet these demands:

- Passive help:
  The user explicitly requests help.
  Context-sensitive help information is generated.

- Active help:
  The user receives help without explicitly requesting it.
  For example, the system offers the user an optimized interaction sequence in order to reach a specific goal.

- Cooperative help:
  The user receives help when he makes errors.
  The system suggests possible corrections or recommends alternative solutions to the user.

- Implicit help:
  The system adapts itself by, e.g.,
  - changing the screen layout,
  - focusing the user's attention,
  - setting defaults.

As stated above, one of our main goals was to provide graphical help, because this seems to be the most adequate way of supporting users working with graphical user interfaces. In order to provide the user with a 'common look and feel' concerning the application and the help system, the PLUS System should be integrated into the graphical environment of the applications.

3 The Design of PLUS

3.1 The Modeling of Plans

There exists a series of plan-based help systems for which a plan language has been defined that is suitable for the problems arising within their respective domains. We took concepts used within the plan languages of the systems REPLIX (cf. [Dengler et al. 87]), MATHILDE (cf. [Hirschmann 90]), and PLANET (cf. [Quast 91]) and extended the language to adapt it to our needs (cf. [Berger & Thies 92] for a comprehensive overview of all properties that we used for the definition of plans).

We decided to choose a hierarchical plan base as the basis for the plan processor. There are three main reasons for this decision:

(1) From a simple point of view, a plan consists of a series of actions that have to be performed in order to successfully complete the plan and thus to reach the goal associated with that plan. But if we take a closer look at common tasks a user is performing when he is working with an application, we notice that small sequences of actions are often part of several plans. To avoid redundancies, it is sensible to combine such sequences to separate plans. These plans, or rather their associated goals, can be included as *subgoals* within more abstract plans. We thereby obtain a plan hierarchy with several layers.

(2) Another reason for working with a hierarchical plan base is our aim of offering the user an adequate assistance on a suitable abstraction level. A typical help scenario might look like the following: A user starts working on a task consisting of several steps, but after reaching a certain point, he does not know how to proceed. If he asks for help in such a case, he certainly does not want to get instructions about the whole plan he is pursuing, but only for the part (the subplan) he has problems in. Moreover, if he can identify parts of a larger plan as logically independent subplans, it is then easier for him to reuse what lie has learned about a subplan, if this subplan occurs in a second task.

(3) Obviously, a plan recognition process working on a plan hierarchy is generally much more efficient than one working on a *flat* plan base. Firstly, the amount of memory needed to store the plan hypotheses may be considerably smaller because of the redundancies (cf. (1)) that occur within a flat plan base. Secondly, performing inference and search processes in a plan hierarchy is much more efficient than in a flat plan base.

3.1.1 Actions

We use the term *action* within the PLUS System for pulldown choices which are selectable within the application. However there are two additional types of actions within a graphical interface environment. We will define them in the following paragraphs.

Generic Actions The PLUS System is designed to run with applications that are running under Screen View. Usually, these applications offer both application-specific actions and so-called *generic actions* which are common to all Screen View applications. These generic actions essentially comprise actions for clipboard management (i.e., *Create*

and *Paste)* and for the visualization of application objects within the different windows (e.g., *Include).*

Navigational Actions Apart from the actions which are selectable via pulldown choices within the application, there are also actions for the navigation within the graphical user interface. The term *navigational action* denotes actions like *scrolling, restoring windows,* and *selecting objects.*

We believe that the exclusion of generic actions and navigational actions from the plan recognition process is sensible. Goal recognition based on navigational or generic actions is not possible, because these actions are usually part of any plan that a user may have in mind to reach a goal. To overcome this restriction of the plan recognition process, the user must have the opportunity to access help concerning generic actions and/or navigational actions. Adequate presentations of generic and navigational actions would be a tutor-like mode telling the user what actions to perform and how to perform them, and an animated help showing the user how to perform actions on the current user interface.

3.1.2 Plans

In the context of plan-based help systems, a *plan* is a sequence of actions that have to be executed to perform a give task, and thereby to achieve specific *goals*. Given the reasons above, we explicitly distinguish between plans and goals. A goal can be achieved in different ways, each of them represented by an alternative plan. Each plan, however, leads to exactly one goal.

Plan Types We allow the assignment of a type to each plan, identifying it as an *optimal, suboptimal,* or *wrong* way to reach the goal associated with the plan. This information can be used by the different components of the PLUS System to decide what kind of help is suitable for the user (e.g, active help or cooperative help, cf. section 1).

Parameter Constraints Usually, the steps of a plan work on a common set of application objects. Each step has a number of parameters. The parameters of an action are placeholders for the application objects that are provided with the action when it is written to the dialog history. In addition, we allow goals to have parameters. The *goal parameters* are placeholders for the application objects that are substantial for the achievement of the goal. Goal parameters are used for the definition of parameter constraints, if the goal is used as a subgoal within higher level plans (see below). Moreover, goal parameters can be used within the descriptions of a goal to establish a context sensitivity of the descriptions.

In order to reflect the relationship between the application objects involved in a plan, it is necessary to define the constraints between the parameters of the plan's actions and goals. We offer the possibility of defining *Equality* and *Inequality* constraints. Due to the fact that applications addressed by the PLUS System deal with object hierarchies, we offer a third kind of constraint, the *Dependent Of* relation.

Sequence Constraints One of the major benefits of graphical user interfaces — in contrast to command-oriented or menu-based user interfaces — is the possibility of processing tasks in parallel and of performing actions in (almost) any order independently from each other. That is, plans in PLUS enforce no strict sequence of

actions to be performed. Therefore, we basically view plans as a set of steps without any total ordering. However, there are usually some temporal relations between the steps of a plan that have to be maintained in order for the plan to be meaningful. We distinguish between two kinds of sequence constraints:

Absolute Positions It might be necessary for a certain step to occur at a particular position when a plan is being performed by the user. A typical example is a plan working on a file. The first action of this plan is to open the file, and the last action is to close it. Therefore, we offer the possibility of assigning an absolute position to each step of a plan.

Relative Positions It might be necessary for a particular step to occur before other steps, as a plan is being accomplished by the user. For example, before any action can be performed, on an application object, this object has to first be created. Therefore, we offer the possibility of defining a set of predecessors for each step of a plan which specify the steps that have to be performed beforehand.

As an additional feature, it is possible to define whatever a step of a plan is compulsory or optional. In contrast to compulsory steps, optional steps do not necessarily have to occur in order to achieve the goal associated with a plan, however their occurrence strengthens the hypothesis that a plan is being followed by the user.

3.1.3 The Input of a Plan Base

Tools for the application or information developer in order to model the plan base should be part of the system. These tools should offer an easy mechanism of interactively specifying plans without requiring a deep knowledge of the formal description of plans. Therefore a plan language which is easily used by applying concepts of an interactive graphical environment should be designed instead of a pure syntactical plan language.

3.2 The Processing of Plans

The main module of a plan-based help system is a plan recognizer. While the user interacts with the application, the plan recognizer tries to map the performed actions to plans, thereby making assumptions about the user's goals. These plan hypotheses form the basis for offering various kinds of help to the user.

Two different approaches exist for plan-based systems. On the one hand, there are systems that generate plans during run-time using a *plan generation system.* This approach is also called plan recognition from *first principles.* On the other hand, there are systems that use a predefined plan-base as an input for the plan recognition component (plan recognition from *second principles).* In the last few years, a lot of research has been done within the area of plan recognition from first principles (see, e.g., [Bauer et al. 92], [Koehler 92]). However, the plan recognition components developed within these projects are far from being suitable for use within help systems which are intended to be integrated into sophisticated applications, since they require a complete axiomatization of an arbitrary application domain. Therefore, we decided to employ a plan recognizer that is based upon the second principles approach. Plan recognition from second principles exploits predefined plan libraries.

In order to cope with the different DMI events, we planned to realize a two-level plan recognition approach. The first level should process low-level events like mouse-clicks and keystrokes. It was planned that an ATN-based parser to do the low-level processing should be employed. The second level processes the application actions performed by the user, e.g., by selecting pulldown menu items. With this two-level approach, we are able to process the low-level events without stressing the actual plan recognition process.

In the first level we protocol the user's favorite interaction styles (i.e., does he mainly use the mouse, or does he prefer 'short-paths') and we build up a simple user model to reflect the user's preferences (for user modeling see, [Wahlster & Kobsa 89], [Rich 89]). Firstly, this simple user model can be employed in adapting help information to the user's habits by considering his preferred interaction styles, and secondly, it allows the detection of alternative interaction principles that are unknown to the user. Moreover, while generating help sequences, the first level of the plan recognition can be used in order to determine the most efficient interaction technique for performing a specific action. The results of this first plan recognition level are the application-specific actions performed by the user. These actions are recorded within a *Dialog History* that serves as an input for the second level plan recognition process.

The second level plan recognition process is based upon a hierarchical plan base called *static plan base* as described in section 3.1 above. We decided to use a *spreading activation* algorithm for the plan processing. A similar algorithm has been employed within the system PLANET (cf. [Quast 91]). The plan recognition component tries to map actions stored in the dialog history to plans contained in the plan hierarchy. A so-called *dynamic plan base* is thereby built up at run-time. The dynamic plan base contains all hypotheses concerning plans and goals being pursued by the user at a certain state of the dialog. Together with a knowledge base containing common help strategies extended by rules and facts about generic interface concepts, these hypotheses serve as the basis for the various help components realized within PLUS (cf. section 4).

3.3 Controlling the Plan Processing

As stated earlier, DMI environments allow the user to act in a very flexible manner. As the user keeps on working with the application, the dynamic plan base may quickly grow and may thus contain plan hypotheses which are no longer plausible. Therefore, additional mechanisms which keep the dynamic plan base clear by rejecting unlikely hypotheses are required. Within PLUS, the following focusing methods are employed:

(1) For each plan, it is possible to specify a list of cancel actions and/or goals (briefly called *cancels*). The execution of a cancel action or the achievement of a cancel goal immediately dismisses the respective plan hypothesis. A typical cancel action is the closing of a window whose presence is essential for the successful execution of a plan.

(2) A special kind of cancel action is the deletion of an application object which has been used by a plan's previously performed actions. We therefore introduced the concept of so-called *generic cancels*. This mechanism causes every plan hypothesis to be immediately dismissed from the dynamic plan base, if one of the involved objects is deleted.

(3) We introduced a *Time Frame* concept (see figure 1) that enables the PLUS System to categorize plan hypotheses into different states depending on the number of user actions that have been performed since a plan hypothesis was last activated (i.e., since the last assignment of a step to a plan hypothesis). As soon as an action activates a plan hypothesis, we call this plan *focused*. If more than Tl actions (Tl is called *Time Frame Focus*) are performed without a new activation of the plan hypothesis, it changes its state to *sleeping*. If it gets no further activation for another T2 steps (T2 is called *Time Frame Sleep*), then the plan hypothesis is dismissed from the dynamic plan base (it is unlikely that the user will continue to carry out this plan).

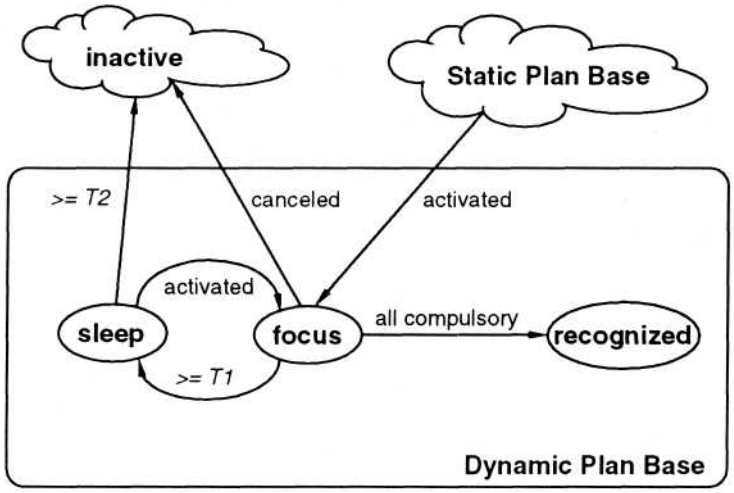

Figure 1: State Transitions using Time Frames

3.4 Animated Help

Object-oriented graphical user interfaces entail new demands in providing the user with adequate help. Static and knowledge-based help systems with a pure textual help (cf. [Wilensky et al. 84], [Breuker 90], [Bauer et al. 91], [Wahlster et al. 93]) reach their limits as soon as the user needs assistance in performing interactions. For example, if the user addresses a question like: *"How do I include object A into container-object B ?'* a generated textual help could possibly sound like: *"Move the mouse to the position of object A and press the left mouse button. Now move the mouse with the left mouse button still pressed to the position of the container object B. Then release the mouse button.*[77] We think that an animated presentation of these interaction steps is more adequate than a pure textual description.

As soon as the user needs assistance in performing interactions within the graphical interface, an animated sequence demonstrating the necessary interaction steps on top of the current interface seems to be the most adequate way of supporting the user.

In contrast to earlier approaches to animated help (cf. [Neiman 82], [Sukaviriya 88], [Sukaviriya & Foley 90]), the animation system of PLUS generates animated presentations of interaction steps in the context of the current task which a user is carrying out.

The animation presentation comprises both the movement of the mouse on the interface and the manipulation of objects (e.g, menus, scrollbars, windows, application objects) with the mouse. In addition, the shape of the mouse changes in order to reflect mouse actions like single-click or double-click with the left or right mouse button.

In order to provide the user with a better understanding of the reason why the animation system performs the current mouse action, a text describing the goal of the animation and the current mouse action is presented in an adequate form (e.g., through speech output from a speech synthesizer).

4 The Realization of PLUS

4.1 The Architecture of PLUS

Figure 2 shows the overall architecture of the PLUS System. PLUS can be divided into three functional parts:

(1) The **Plan Processor** including the *Plan Recognition*, *Plan Completion*, and *Plan Generation* components.

(2) The **End User Interface** including the modules *InCome*$^+$ and *AniS*$^+$, and a context-sensitive entry to a hypertext-based help facility.

(3) The module *PlanEdit*$^+$ as a tool for application developers for specifying plans.

These modules work on four different data resources:

- The *Dialog History* containing information on the user interactions recorded by the application. The Dialog History is shared by the application and PLUS via the OS/2 Dynamic Data Exchange (DDE) mechanism.
- The *Static Plan Base* containing typical user tasks. The static plan base is generated by an application specialist using PlanEdit$^+$.
- The *Dynamic Plan Base* containing hypotheses about the plans and goals the user is currently pursuing.
- The *Generation Knowledge Base* containing rules that model the interface syntax, the application semantics, and generic interface concepts (e.g, how to perform navigational interaction steps).

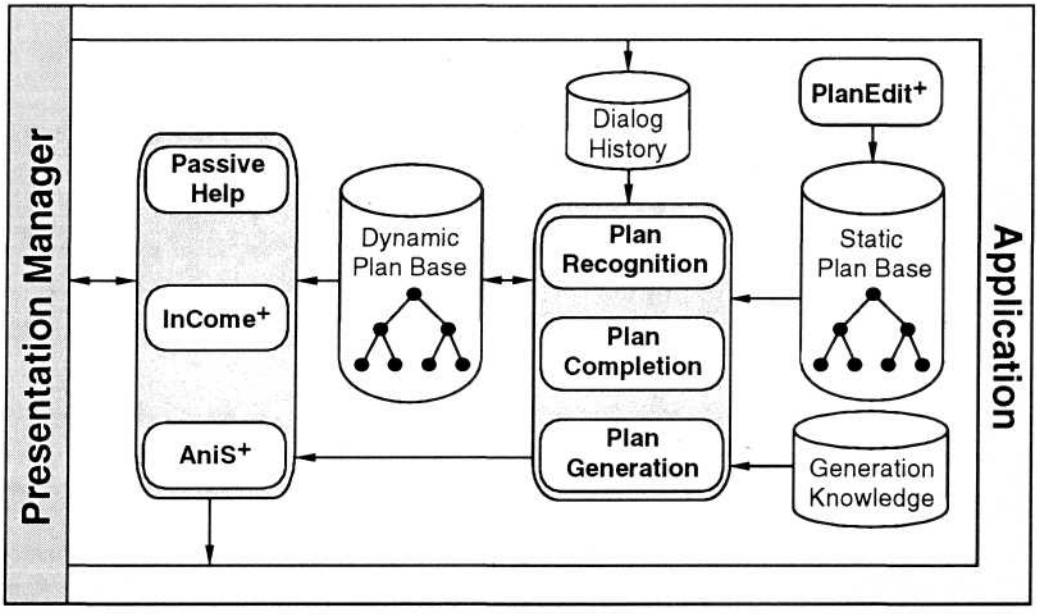

Figure 2: Architecture of PLUS

4.2 The Definition of a Plan Base

For each application running with the PLUS System, a separate plan base has to be built up. Typical tasks performed by a user when he is working with the application should be modeled within this plan base. We describe user tasks in terms of actions, plans, and goals. These objects are contained within a plan hierarchy called static plan base that is structured as follows (see figure 3):

- The lowest layer consists of the *actions* representing the application actions that can be performed by the user via pulldown or popup menu choices or by direct manipulation interactions. Actions are part of plans.
- A *plan* represents one way of reaching a specific goal. It consists of a set of actions and/or subgoals (i.e., goals on a lower hierarchical level). Each plan leads to exactly one goal.
- A *goal* is a system state that a user wants to achieve while interacting with the application. A goal may be reached in different ways, each of them represented by an alternative plan. Goals may be contained as subgoals within higher level (i.e., more abstract) plans.

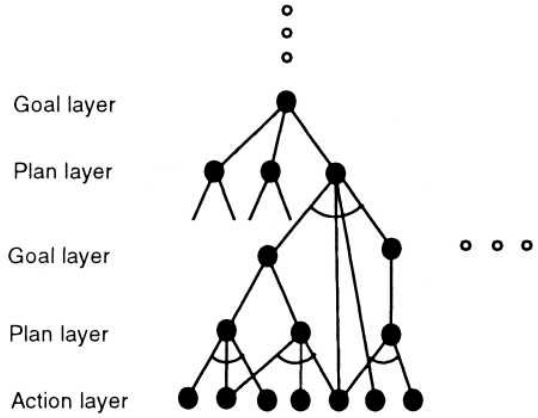

Figure 3: The Structure of a Plan Base

For the definition of the static plan base, we developed the language *GPL+* (**G**oal **P**lan **L**anguage) that provides mechanisms to build hierarchical structures. GPL$^+$ has been designed to cope with specific features of graphical user interfaces like *multiple selection, optionality, parallelity, object hierarchies,* and *multiple views* on objects. In addition, features common to plan recognition like *parameter* and *temporal constraints, plan cancellation,* and *plan interactions* can be modeled with GPL$^+$. [Berger *h* Thies 92] contains a comprehensive summary of all properties that can be defined for the elements of a plan base.

PLUS offers a convenient tool for specifying a plan base without the need for a deep knowledge of the formal description of plans. The module **PlanEdit**$^+$ (cf. [Berger & Thies 92]) provides a graphical user interface that allows the plan designer to build up the plan base interactively by means of direct manipulation, and to generate the appropriate **Smalltalk** objects that are used by the plan processor for the plan recognition and plan completion processes.

Figure 4 shows the PlanEdit^{+} main window, in which most of the interaction takes place. The elements of the plan base are displayed as graphical objects. Each object consists of an icon representing the element's type and the element's name. The *Type Box* in the lower left corner of the main window contains icons for the three types of elements contained in the plan base: actions, plans and goals. These icons can be used to generate new elements of the respective types. The properties of the elements may be defined within a series of dialog boxes.

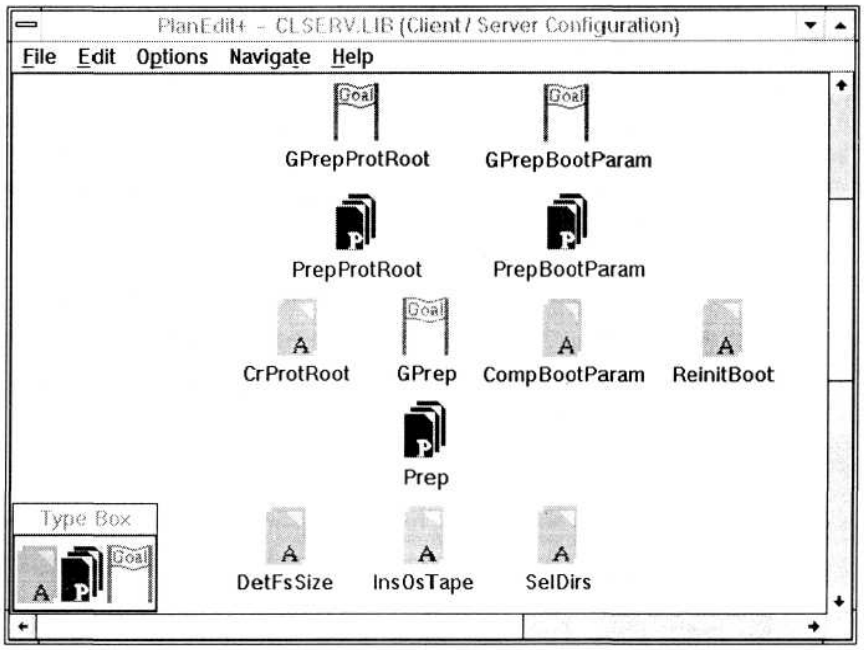

Figure 4: PlanEdit^{+} Main Window

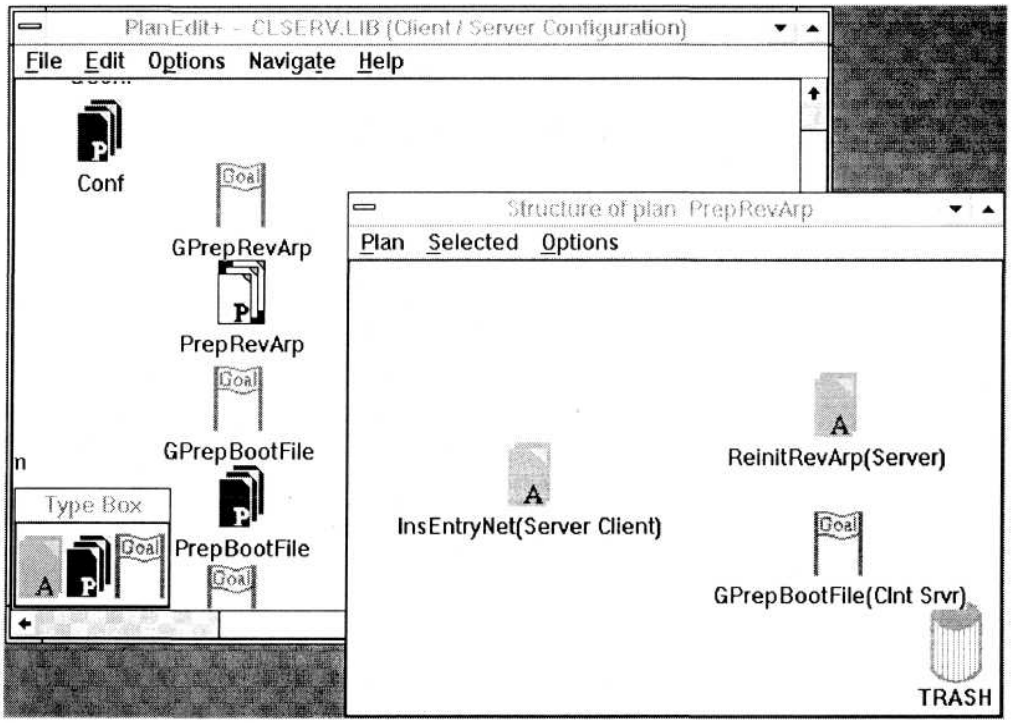

Figure 5: PlanEdit^{+}: Various Window Types

The contents of the main window may become confusing for the user as the plan base grows. Therefore, we added a second window type enabling the separate examination of the structure of previously defined plans and goals, and the easy modification of their properties. Figure 5 shows an example of a plan window on top of the main window. In the main window, the elements of the plan base may be arranged arbitrarily without considering the structure of the plan base. Within a plan window, however, the layout of the objects corresponds to the logical sequence of the elements within the plan, as defined through the sequence constraints.

A second tool for the generation of a plan base has been developed at the IBM Lab as part of a masters thesis (cf. [Braune 92]). This tool introduces a textual format for the definition of an AND-OR tree representing the structure of a plan base, and for the specification of the properties of the elements contained in a plan base. The textual format is based upon the *Abstract Syntax* (AS) developed by IBM. The AS has been extended to meet the requirements necessary for the definition of a plan base.

The textual description of a plan base can be entered and edited within a conventional text editor. Additionally, tools for the mutual conversion between the textual format and the internal format used within PlanEclit$^+$ have been developed. This means that both formats can be employed in parallel and that the plan designer can use whichever tool he prefers, depending on the current situation.

4.3 The Plan Processor

The plan processor is the core of the PLUS System. It consists of three parts, the plan recognition component PlanRecognizer$^+$, the plan completion component, and the plan generation component. The plan completion component and the plan generation components serve as the basis for the visualization of possible future actions within InCome$^+$ (cf. section 4.4) and for the animation component AniS$^+$ (cf. section 4.5).

4.3.1 PlanRecognizer$^+$

Within the PLUS System, PlanRecognizer$^+$ plays the part of the plan processor. It receives input from the application via the dialog history.

The dialog history is updated each time the user performs an action within the application. Each update triggers PlanRecognizer"[1]" which works as follows (cf. figure 6) assuming the new entry within the dialog history is action **a**:

1. The corresponding action within the static plan base is identified and a new instance is created.
2. The dynamic plan base is looked up for existing plan hypotheses to which a could be assigned.
3. All constraints that are defined for **a** within the static plan base are verified for each plan hypothesis that is determined by the previous step.
4. If the verification has been successful, **a** is assigned to the corresponding plan hypotheses.

5. In addition, for each plan to which **a** could be assigned, a new plan hypothesis is created and again all constraints defined for **a** are verified. Each new hypothesis to which **a** could not be assigned is dismissed.
6. If a plan hypothesis is completed by the execution of **a** then it changes its state to *recognized.*
7. Each plan hypotheses that changed its state to *recognized* spreads its activation to all plans to which it could belong by using this algorithm, modified by replacing action **a** with plan hypothesis.

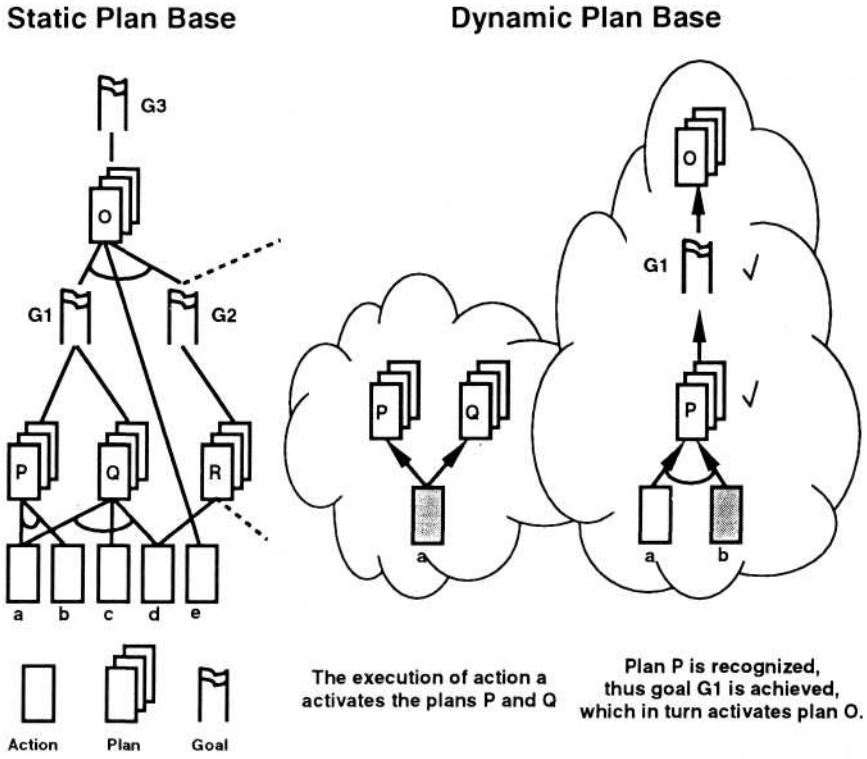

Figure 6: Spreading Activation

During the plan recognition process, PlanRecognizer$^+$ keeps a record of the plan recognition process. When an entry within the dialog history has been processed, PlanRecognizer$^+$ sends the record to the context visualizing component InCome$^+$. Based on the information contained in the record, InCome$^+$ builds an internal representation of the plan recognition process and of the interaction context (cf. section 4.4).

4.3.2 Plan Completion

The term *Plan Completion* describes the generation of a sequence of actions that perform a specific plan. The execution of the generated actions leads to the goal associated with the plan. The sequence is generated according to the definition of the plan. The definition includes various constraints that had been defined for the elements of that plan during the design of the static plan base. The sequence is called to be *valid*, if the constraints are solved. This is done by considering sequence constraints, minimum/maximum iteration constraints, and parameter constraints.

A special case appears when the elements of a plan have minimum iteration constraints with 0 value. The 0 value means that these elements are *optional.* An element that is defined to be *optional* within a plan must not necessarily occur when persuing that plan. Consequently, during the generation of a valid sequence, the *plan completion* component considers only elements that are mandatory for the plan to be completed.

A speciality of the *spreading activation* algorithm must be handled by the plan *completion component:* The *spreading activation* algorithm allows spreading of activation only for elements that have just been recognized or for actions that have just been executed by the user, i.e. the user can start a plan *(focused plan)* without its corresponding goal being activated (due to the unrecognized plan). If the *plan completion* component then generates a sequence for another plan that starts with a goal, it must consider all *focused plans* that could lead to that starting goal. All constraints that are defined for the starting goal must be satisfied as far as the parameters of the focused plan are already known. If the plan completion component did not consider this *focused* plan, the generated sequence would include steps that had already been performed by the user. In order to suppress this misleading information, the plan completion component examins focused plans which are in the dynamic plan base during generation of a sequence (cf. step 6 of the plan completion algorithm).

The *plan completion* component may generate valid sequences for plans that are already in the dynamic plan base and may generate valid sequences for plans that are not yet activated by the plan recognizer. Recall that a plan in the *dynamic plan base* is activated by the plan recognizer due to the assignment of actions performed by the user to that plan (during the *spreading activation* phase).

The generated sequences are used by InCome$^+$, its tutor, and the stand-alone tutor. The elements of the sequence will be visualized to provide the user with the information he needs to resume or to finish his work.

The plan completion component is activated by a request for it from InCome$^+$ or from the stand-alone tutor. The algorithm for the plan completion works as follows :

1. Determine the appropriate plan.
2. Determine the steps of all elements of the plan specification that are not yet performed and that are mandatory *(missing steps).*
3. Determine the absolute positions of the *missing steps,* using the sequence constraints *Absolute Positions* for them, and place the *missing steps* within the new sequence reflecting their absolute positions.
4. Determine the relative positions of the *missing steps,* using the sequence constraints *Relative Positions* for them. Order the *missing steps* according to their relative positions within the new sequence.
5. Place all remaining steps (i.e. not yet placed) within the new sequence considering the positions already occupied.
6. Take the first step within the sequence arid try to assign a plan that has already been activated to that position by verifying the defined constraints.

7. Replace each step in the new sequence with a newly created instantiation of the class corresponding to that step. Steps assigned during step 6 are omitted.
8. Propagate parameter types and values within the new sequence.
9. Answer the new sequence.

4.3.3 Plan Generation

The *plan generation* component is used by the *How to...* tutor and the animation system AniS^{+}. The plan generation component works upon a knowledge base using a simple backward chaining algorithm. The knowledge base is split into an application part, a generic part and an interface part. The application part includes information about application-specific actions, the generic part includes information about actions that are common to all applications (due to the SAA/Common User Access) and the interface part contains information on how to access application-specific runtime information. Within the knowledge base, pre- and postconditions for actions are defined along with specifications for the tutor and the animation system. The knowledge is packed into so called *chunks.* These *chunks* include, depending on the part of the knowledge base for which they are defined, various slots: *name, preCond, postCond, stepsAS, stepsPG,* and *builtln*. The slots *stepsAS* and *stepsPG* contain information that is collected when the corresponding chunk is processed successfully. The information collected builds up a sequence of steps that must be carried out to reach the specified goal. The slot *stepAS* is used for AniS"[1]" and the slot *stepsPG* is used for the *How to...* tutor. The contents of the slot *huiltln* is a function that queries runtime information from the connected application.

In order to understand the dependencies between the preconditions, the postconditions, and the chunks defined in the knowledge base, a closer look at the inference process performed by the *Backward-Chainer* (BC) is required.

InCome^{+} notifies the BC about an application action that has to be performed by the animation system or that has to be explained to a user, who asked a 'How to...' question. For example, two application objects should be connected using the menu function "connect". The BC tries to map the menu function "connect" together with the provided arguments to the name and the placeholders of an application chunk. If a suitable chunk is found, the BC checks the preconditions of that chunk by trying to verify the conditions defined in the slot *preCond.* The verification is done as follows: if there is an interface chunk with the same postcondition as the precondition to be verified, the built-in method defined in this interface chunk is performed. If no such interface chunk exists, or if the built-in method answers *false,* the BC searches in the generic knowledge part of the knowledge base for chunks with a postcondition that is identical to the precondition to be verified. Thereby a list of chunks is created and sorted according to the sequence of the chunks as defined in the knowledge base. The first entry in the list is taken and the BC process recurses to reach the new goal. If the derivation fails, the next entry in the list is taken, and so on. If the list is empty, the BC fails to reach the specified goal and terminates. Otherwise, the inference process has been successful and the BC returns a list of (either animation or generation) steps necessary to perform the application action.

4.4 The Module InCome$^+$

One of the central components for graphical help within the system PLUS is the *Interaction Control Manager* InCome$^+$ (cf. [Thies 90], [Fehrle &, Thies 91]). It provides a graphical visualization of the current dialog context, the dialog history, and possible future interactions. InCome$^+$ gives the user a quick and helpful reminder of the system state to resume suspended tasks. It supports the user in leaving system states unfamiliar to him and in exploring actions (cf. [Paul 89]) that can next be executed when completing unfinished tasks. InCome^{4}" meets the following demands:

- Adequate visualization of user interactions,
- Display of different levels of abstraction selectable by the user,
- Visualization of possible future interactions,
- Graphical navigation services, and
- Display of plan interactions, like embedded, overlapping, and interrupted plans.

PlanRecognizer$^+$ and the plan completion component form the backbone of InCome$^+$. The plan completion component generates, on demand, a valid sequence of actions for plan hypotheses that are contained in the dynamic plan base. Several constraints defined within the hierarchical plan base are satisfied. For example, sequence constraints are solved and parameter values that are already known are propagated according to parameter constraints (cf. section 3.1).

PlanRecognizer$^+$ notifies InCome$^+$ about the ongoing plan recognition process. On receiving the incoming data, InCome$^+$ generates an internal representation of the interaction context and displays it as a graph structure on the screen (see figure 10). The instances of the object classes *action, plan,* and *goal* are represented as nodes. An *action* is represented by an icon that looks like a single sheet of paper, a *plan* is represented by a stack of papers, and a goal is represented by a goal banner (see figures 7-9).

Figure 7: Action

Figure 8: Sequence of Actions

Figure 9: Goal

The visualized structure resembles a directed graph reflecting the chronological order of the performed interactions from top to bottom. Objects belonging to the same plan are connected by arcs. The sequence is ended by a goal banner representing the associated goal (cf. figure 10). InCome$^+$ runs in its own window. The presented nodes are selectable via mouse clicks. User actions provided by InCome$^+$ can be divided into four categories (cf. [Thies 92] for a comprehensive description of the functionality offered by InCome$^+$):

- Graphical Navigation,
- Hierarchical Navigation,

- Tutor Activation, and
- Remote Application Interaction.

Graphical Navigation includes actions like *scrolling*, *including*, *excluding* and *removing nodes*, and *searching* for specific nodes.

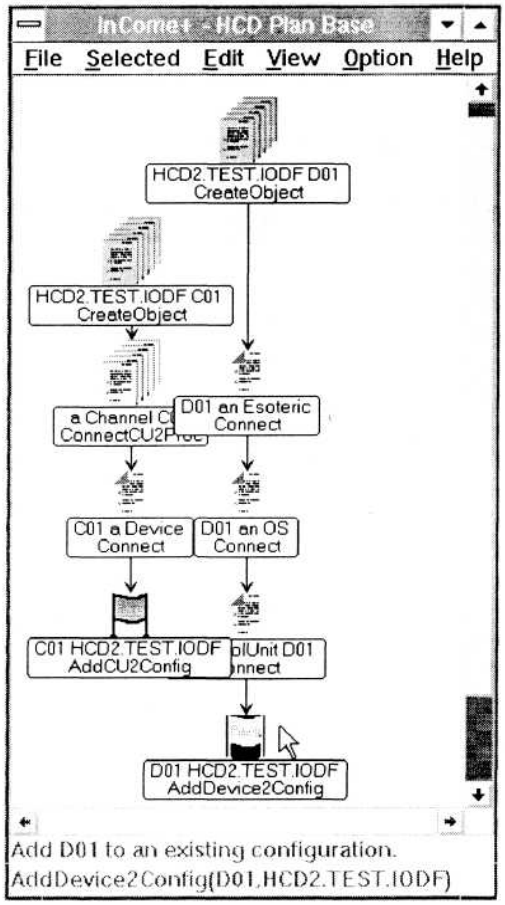

Figure 10: InCome+

Hierarchical Navigation supports the user in viewing plans on different abstraction levels. InCome+ offers actions for *expanding* and *collapsing* plans. Expanding is equal to a downward movement in the hierarchy and collapsing is equal to an upward movement in the hierarchy. Expanding and collapsing of plans are realized within InCome+ by grouping together sequences of actions into plans or by replacing plans with their sequences of actions.

In addition to the navigation through the hierarchy, InCome+ is able to visualize various plan interactions like *plan interruption*, *plan embedding*, and *overlapping of plans.* Figure 11 shows a snapshot of an interaction context where two plans, namely *AddCU2Config(C01, HCD2.TEST.IODF)* and *AddDemce2Config(D01, HCD2.TEST.IODF),* overlap each other and where both plans include embedded plans, e.g., plan *AddCU2Config* includes two embedded plans: *CreateObject* and *ConnectCU2Proc.* Both plans overlap at the action *Connect(C01, D01).*

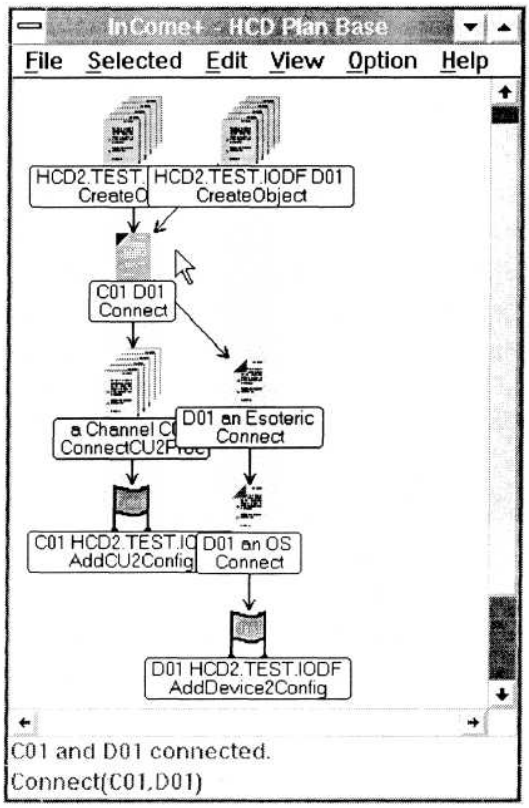

Figure 11: Plan Interactions

Tutor Activation is carried out by selecting a goal and activating the tutorial mode. The user is guided by the system to reach the chosen goal. After activating the tutorial mode, InCome$^+$ requests an optimal sequence of actions in order to reach the goal selected from the plan completion component. In this context, *optimal* means the most efficient sequence of actions carried out in order to reach a goal. The attribute *optimal* is defined at the plan level within the static plan base and is therefore predefined. The plan completion component generates this sequence by considering various constraints (cf. section 3.1) defined in the hierarchical plan base. Known argument values are propagated. The sequence of actions is textually represented in a separate window like a *to-do-list* (see figure 12).

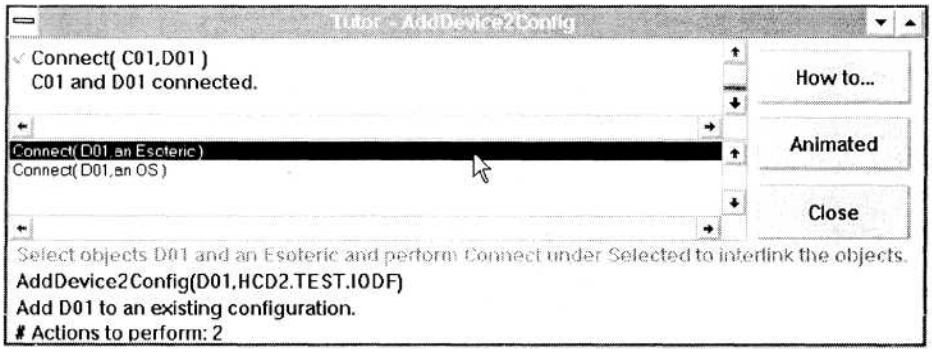

Figure 12: Tutorial Mode and Actions Performed

The Tutor lists each action necessary to reach the selected goal and supervises the actions performed by the user. The user receives feedback from the Tutor by marking the corresponding entry with a check mark, if the action performed is part of the sequence of steps

required (see figure 12). If each action listed is performed, the user receives notification that the chosen goal has been reached. If the user has made a mistake by performing an action that hinders the achievement of the selected goal, the Tutor informs the user about this.

To offer the user help concerning generic and navigational actions, we implemented a second tutor-like mode that conveys *how to perform* an action within the current interaction context. After the *Row to* mode has been activated, a window pops up, listing from top to bottom, navigational actions that have already been performed by the user, and navigational actions that are still necessary for the execution of the selected action on which the *How to* mode has been activated. Navigational actions which have formerly been executed are marked by a sign in front of their respective entries.

Three dots (...) are a special sign, indicating that the system can not predict subsequent navigational actions because the result of the navigational action listed above the three dots cannot be anticipated. If the navigational action above the three dots is executed, the three dots disappear and the next navigational actions can be anticipated by the system. After the user has performed the first navigational action that has no mark before its entry, the system anticipates the next navigational actions necessary. The window is updated, the executed navigational action is marked with a sign, and the next navigational actions are added to the list.

The *How to...* window is closed if no more navigational actions are necessary for the execution of the selected action. The user is notified by a message about the successful execution.

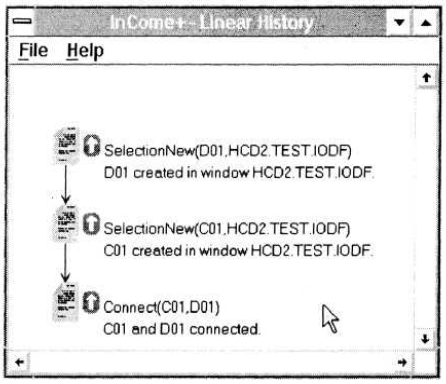

Figure 13: Linear Dialog History

Remote Application Interaction is provided by the animation system AniS^{+} that can be activated within InCome^{+}. In addition to AniS"[1]" (cf. section 4.5), some ideas were developed in order to provide access to the *undo*- and *redo-mechanisms* of an application. InCome^{+} could provide an interface to these mechanisms. In order to be able to deal with two different principles for undo *(function-oriented* vs. *state-oriented;* see also [Rathke 87], [Rathke 89], [Yang 90]), InCome"[1]" uses an extended function-oriented approach by handling *freezing-points* (cf. [Paul 89]). Freezing-points are snapshots of system states that are saved within the application. It is possible to reset the application state to one of these freezing-points by activating an application function. By representing the interaction context in a more abstract way than by a linear dialog history, the user can perform undo-actions and redo-actions on plans rather than actions. This is called *undoing on a*

semantic level. An undo applied to tasks without reversing successor tasks is not supported *('freies undo' (unrestricted-undo),* cf. [Rathke 87]).

In addition to the visualized interaction context, a window is provided that presents the linear dialog history. The visualization emphasizes *reversible actions* and *freezing-points* that are set within the application. The lower left window in figure 13 represents the linear dialog history. Within figure 13, the arrows on the right side of actions denote *reversible actions.*

4.5 The Module AniS$^+$

As a substantial extension of the graphical user assistance, we integrated the presentation of animated help within the PLUS System. Within the PLUS System, animation is performed by the component AniS$^+$ (cf. [Thies 93]). AniS$^+$ generates animated presentations of interaction steps in the context of the current task being performed by a user. The animation presentation comprises the movement of the mouse on the display and the manipulation of objects (e.g, menus, scrollbars, windows, application objects) with the mouse. The shape of the mouse is varied to reflect mouse actions like single-click or double-click with the left or right mouse button (see figures 14 and 15).

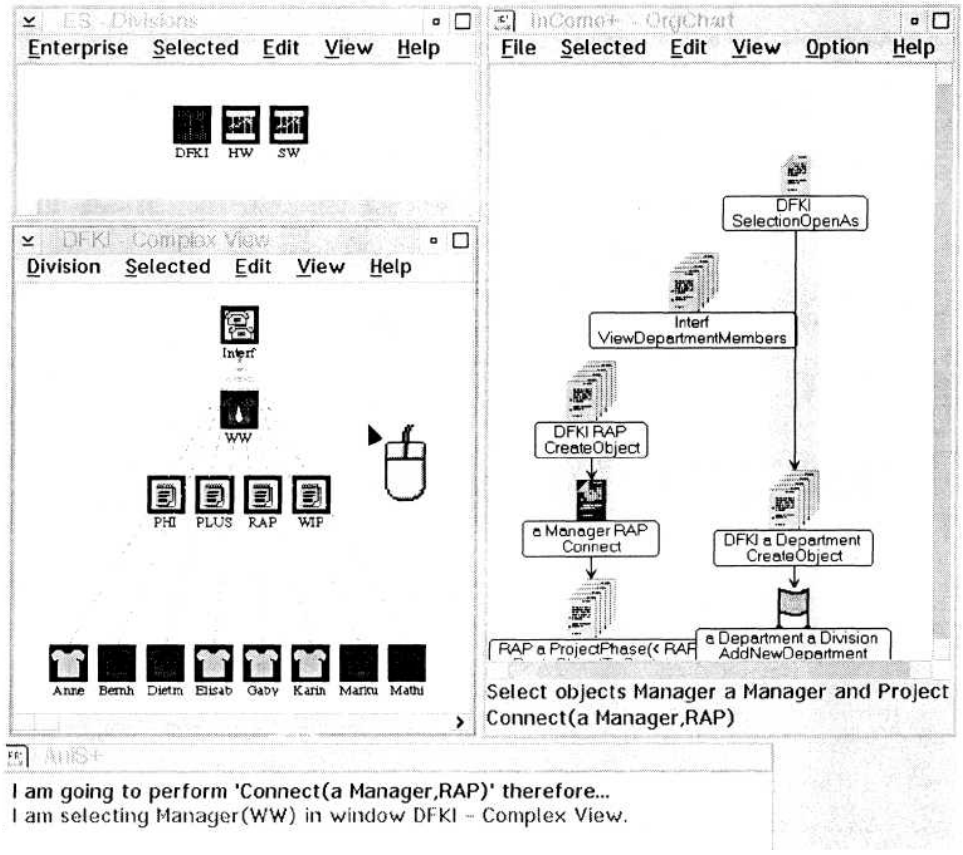

Figure 14: AniS$^+$ generates navigational actions...

The mouse movements and clicks are simulated by sending corresponding mouse events to the interface in such a way that the interface and also the application are acting on these events as if they were performed by the user. Thus, the actions are really executed within the application.

A text describing the goal of the animation and the current mouse action is displayed in order to provide the user with a better understanding of why AniS$^+$ performs the current mouse action. By variable substitutions, the prestored text fragments are adapted to the current application context.

An action sequence generated by the plan completion component serves as an input to AniS$^+$. AniS$^+$ works with a two phase planning loop to incrementally generate the interaction steps (e.g., mouse movements and clicks) necessary for the execution of the generated action sequence. The inner loop considers the changes within the interface context (e.g., selecting an object, scrolling the window) and uses a backward-chaining algorithm. The outer loop considers the changes of the application context that take effect after the execution of an action and involves both the plan recognition process by reacting upon the performed action and the plan completion component by reflecting new parameter values provided by the user.

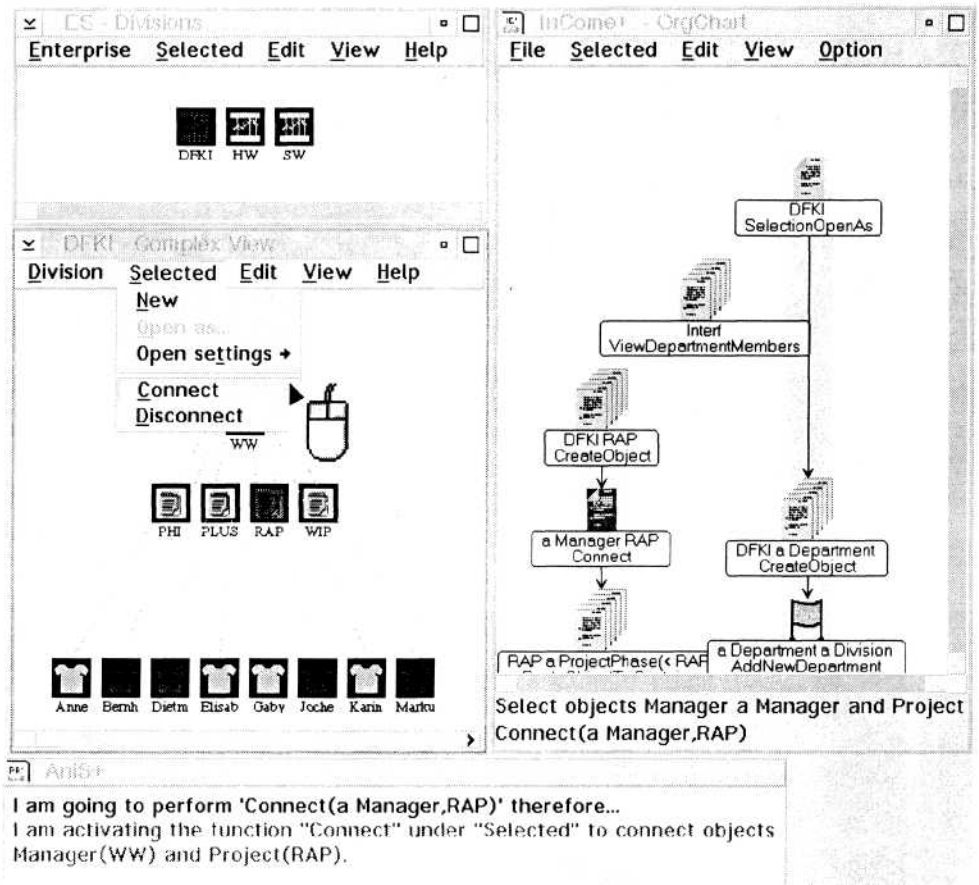

Figure 15: ...and varies the mouse shape

During the backward-chaining process, AniS$^+$ accesses a knowledge base that defines specific pie- and postconditions for each action. Informal examples of such preconditions are [a]to apply an action to an object, it must be selected" and [a]an object can only be selected if it is visible". The representation of generic interface concepts allows us to generate navigational interaction steps (e.g., steps to scroll the visible area of a window). In addition, the knowledge base models the interface syntax (e.g., clicking on an object changes its state to be selected) and the application semantics (e.g., which objects can be visualized in which types of windows and which actions are applicable to which objects).

There is an interface to the application for accessing information about, e.g., selected objects, visibility of objects and the applicability of actions within specific types of windows. Although selected objects are considered as replacements for missing parameters during the execution of the animation, not every parameter can be anticipated from the result of the plan completion process. For that reason, the user is prompted to provide missing parameters.

Animation as part of a plan-based help system is a sensible extension for supporting the user in performing interaction steps in an interactive graphical environment. It fills the gap between the concepts of an interactive graphical interface and a textual representation of help. Although animation can be valuable, merely using animation in help does not deliver a perfect help system. Minimal textual explanations are presented with the animation to help a user to generalize concepts (see the lower part of figures 14 and 15).

4.6 Stand-Alone Tutorial

A stand-alone tutorial largely based on the PLUS System has been implemented at the IBM Lab as a master's thesis (cf. [Scheidel 92]). The tutorial is a framework allowing the information developer to integrate plans and add further information as well as hints in a more didactic way. Learning information is structured by lessons consisting of a number of paragraphs. Each paragraph describes a goal and a corresponding plan to reach this goal. A paragraph is displayed in a separate window with several areas containing

- a summary of the task,
- a detailed textual description,
- preconditions which have to be satisfied, and
- the graphical visualization of a plan (according to the visualization used by InCome^{+}, extended by icons representing navigational actions).

In contrast to the PLUS System the stand-alone tutorial does not communicate with the application. It invokes the plan processor by a handle identifying a specific plan and receives the complete interaction sequence needed to accomplish this plan.

4.7 Steps towards Integration

4.7.1 Changes in the Objectives

During the project period, we decided to shift the focus of the PLUS System towards a possible integration of the PLUS System into an IBM product. Due to the switch towards the product integration, we had to cut the initial PLUS activity plan:

- The first level of our plan recognition concept (ATN-based event handler) has not been implemented. Therefore, the information contained within the dialog history is directly provided by the application.
- Different help strategies (active, cooperative, implicit help) could not be realized.

However, little extra effort is necessary in implementing the active help component, because the concepts of optimal, suboptimal, and wrong plans are already incorporated within the PLUS System.

4.7.2 Activities for the Integration

As a result of the intended integration of the PLUS System into the Screen View product, PLUS had to adhere to some rules and standards used therein.

Error Handling If a Screen View module detects a bad return code of another Screen-View service or an operating system service, then an error message is written into the Screen View error log. This message may also be presented to end users. Within Screen-View, the error handling DLL is implemented as a multi-threaded DLL. Therefore, we implemented a server process which is able to communicate with multi-threaded DLLs. This server communicates through a pipe with a client, who, in turn, is called from Smalltalk V/PM. This client is implemented as a single-threaded DLL.

National Language Support The concept of National Language Support (NLS) is realized within the PLUS System. All text strings appearing at the surface are internally coded by unique ids. At runtime, these ids are substituted by the respective strings contained in a dictionary that is filled at startup time from a corresponding DLL. For each target language, a separate DLL containing the language-specific dictionary will be supplied with the PLUS System.

In the current version of PLUS, the services that must be delivered from the application are implemented within the PLUS System and within the Smalltalk prototypes of the target applications. On the one hand, these services transmit information about the objects and the actions used within the applications and about their relations (e.g., which objects are includable in which types of windows, which actions can be applied to a particular object). On the other hand, dynamic information required by the plan generation component at runtime concerning the current state of the interface (e.g., which windows are visible, which objects are selected) is transmitted. In the future, the former are to be substituted by services accessing information contained within the *Abstract Syntax Table* that exists for each Screen View application. These services have been implemented as part of a masters thesis at the IBM Lab (cf. [Braune 92]).

5 Results of the PLUS Project

5.1 Integration of PLUS into ScreenView

In the following subsections, we will briefly describe the platform ScreenView in which the PLUS System will be implemented, the results of a code inspection of the PLUS System, and the state of the integration.

5.1.1 A Short Sketch of ScreenView

ScreenView (cf. [IBM 92b]) is the central platform implementing the End-Use Dimension of SystemView. ScreenView is an integrated environment for developing and running applications in the area of system management products. The implementation of Screen-View follows a strict separation of interface logic and function logic. While the interface logic resides on a workstation, the function logic can be distributed between the host and a workstation. ScreenView services and tools support user interactions as follows:

- A work area provides application access by means of a graphical user interface.
- A generic navigation, object and view handler — called GenOVHa — enables the user to navigate through complex object structures using a graphical object-oriented user interface.

5.1.2 Code Inspection

Due to the planned integration of the PLUS System into ScreenView, a code inspection concerning the quality of the produced Smalltalk code has been carried out at the IBM Lab in December 1991. For that purpose, a comprehensive specification of the PLUS System has been supplied (cf. [Thies fc Berger 92c]). The architecture of PLUS has been presented, and the object-oriented design of PLUS and the Smalltalk code have been inspected by experienced IBM employees from various departments that are related to PLUS.

The following is a summary of their remarks:

- The high quality of the documentation was appreciated.
- The PLUS architecture was essentially approved.
- The design was accepted completely, and its functionality was considered to be adequate. It was suggested to point out known limitations.
  Some sensible recommendations concerning possible code improvements were realized thereafter.
- It was confirmed that the code is completely readable.
- Some work items, necessary for the integration into ScreenView, were listed: Error Handling, NLS, and integration into the User Interface Services of ScreenView.
- The expected costs for tests have been estimated differently due to their limited experience to date concerning the testing of software written with an object-oriented programming language.

5.1.3 The Current State of the Integration

So far, no real integration of PLUS into Screen View has been achieved. Rather, the PLUS System has been successfully tested with Smalltalk prototypes of the two Screen View applications HCD and OrgChart. The communication between the PLUS System and the applications is realized using the Dynamic Data Exchange (DDE) concept provided by OS/2. There are different 'communication paths' that follow a defined protocol. It should be possible to take over these protocols almost unchanged when the integration is performed.

The actual integration of PLUS into Screen View will be carried out at the IBM Lab at a later date. To assist this integration as far as possible, a comprehensive documentation of the PLUS System, including a full specification of the implemented Smalltalk classes, the external and internal interfaces, and known limitations, has been provided (cf. [Thies & Berger 92c]).

5.2 Usability Evaluation

To obtain some qualitative data about the user value of PLUS, we exploited a usability test of the Screen View product and demonstrated PLUS to several test participants. Following is a summary of their remarks:

- They request a task-oriented system introduction and confirm that PLUS is a good vehicle.
- They confirm that the dynamic concept of PLUS supports users in all interaction states. In addition, they appreciated having the choice of a completely user-driven dialog, a completely system-driven dialog, or a mixed dialog form.
- During animation sequences, they like having to enter parameters for functions interactively, because this gives them an active learning role.
- They claim that PLUS supports their way of learning a new application — to play around interactively without reading much hard-copy information.
- They think that PLUS allows a quick revision of 'how to work with an application', if users had not worked with that application for a long time.

The test showed that the users were able to correctly apply the strategies that they had learned during the PLUS demonstration. In general, we can conclude that PLUS meets many requirements and demands of users that are familiarizing themselves with a new application.

Beside this usability test, the PLUS System has been tested very extensively by the PLUS project members and by several research assistants during the design and implementation phases, so that a lot of improvements and rectifications could be conducted beforehand.

6 Publications, Talks and Presentations

6.1 Publications

The following papers about PLUS have been published:

- *InCome: A System to Navigate, through Interactions and Plans* by T. Fehrle and M.A. Thies, in: Human Aspects in Computing: Design and Use of Interactive Systems and Information Management, Proceedings of the HCI International '91, Stuttgart, Germany.
- *Plan-Based Graphical Help in Object-Oriented User Interfaces* by M.A. Thies and F. Berger, in: Proceedings of the workshop on "'Advanced Visual Interfaces", May '92, Rome, Italy.
- *Planbasierte graphische Hilfe in objektorientierten Benutzeroberflächen* by M.A. Thies and F. Berger, in: Innovative Programmiermethoden fur Graphische Systeme, Proceedings of the GI-Fachgespräch, June '92, Bonn, Germany.
- *Perspektiven zur Kombination von automatischem Animationsdesign und planbasierter Hilfe* by W. Graf (member of the WIP project at the DFKI) and M.A. Thies in the KI journal, Volume 6, Number 4, 1992.
- *Task-Oriented User Assistance for Interactive Graphical Environments,* by M.A. Thies and F. Berger, in: Proceedings of the 5th International Conference on Human-Computer Interaction, HCI International '93, August, 1993, Orlando, Florida, USA (cf. [Thies *h* Berger 93]).
- *Animated Help as a Sensible Extension of a Plan-Based Help System,* by M.A. Thies, in: Proceedings of the 5th International Conference on Human-Computer Interaction, HCI International '93, August, 1993, Orlando, Florida, USA (cf. [Thies 93]).

Furthermore, some working papers summing up results of distinct areas of PLUS have been written:

- *PLUS System Specifications* (cf. [Thies fe Berger 92c]).
- *PlanEdit*$^{+}$ *User's Guide* (cf. [Berger & Thies 92]) — planned to be published also as DFKI Memo.
- *InCome*$^{+}$ *User's Guide* (cf. [Thies 92]) —- planned to be published also as DFKI Memo.

An article about the PLUS project has been published within the IBM Nachrichten, Number 309, June '92. The paper *A Knowledge-based Help Environment for Task-oriented Assistance in Graphical User Interfaces* has been submitted to appear in the IBM Information Development Newsletter, 1/93.

6.2 Talks

The following conference talks have been given by members of the PLUS project, partially combined with publications within the respective conference proceedings:

- *Intelligente Benutzerschnittstellen* by W. Wahlster at the BTW Tagestutorium, March '91, Kaiserslautern, Germany.
- Tutorial *User Modeling and Plan Recognition* by W. Wahlster at the International Summer School on AI, July '91, Prague, CSFR.
- *InCome: A System to Navigate through Interactions and Plans* by M.A. Thies at the HCI International '91, Stuttgart, Germany (cf. [Fehrle & Thies 91]).
- *PLan-based User Support — an Implementation of a Knowledge-based Help Environment for Graphical User Interfaces* by T. Fehrle at the workshop on "Future Trends of User Interface Technology", organized by the IBM Academy, April '92, Somers, New York.
- *Planerkennung als Grundlage für intelligente Benutzerschnittstellen* by W. Wahlster at the DEC-Symposium, November '91, Koln, Germany.
- *Plan-Based, Graphical Help in Object-Oriented, User Interfaces* by M.A. Thies at the workshop on "Advanced Visual Interfaces", May '92, Rome, Italy (cf. [Thies & Berger 92a]).
- *Planbasierte graphische Hilfe in objektorientierten Benutzeroberflächen* by M.A. Thies at the GI-Fachgesprach "Innovative Programmiermethoden für Graphische Systeme", June '92, Bonn, Germany (cf. [Thies & Berger 92b]).
- *Intelligente Multimodale Benutzerschnittstellen* by W. Wahlster at the Siemens AG, October '92, Munich, Germany.
- Keynote lecture *Intelligente Benutzerschnittstellen als Grundlage erfolgreichen Informationsmanagements* by W. Wahlster at the opening of the "Saarländische Technologiemesse", October '92, Saarbrücken, Germany.
- *PLan-based User Support (PLUS) - a Prototype of a Knowledge-based Help Environ ment for Graphical User Interfaces* by V. Schölles at the "Interdivisional Technical Liaison (ITL) on Expert Systems", October '92, Yorktown Heights, New York.
- *Experiences with a Smalltalk Implementation of a Plan-based Help Environment (PLUS)* by V. Schölles at the European Object-oriented Software Symposium, Oc tober '92, Böblingen, Germany.
- Keynote lecture *Perspektiven intelligenter, plan-basierter Benutzerschnittstellen* by W. Wahlster at the IBM-Kolloqium for Prof. Endres, December '92, Böblingen, Germany.

6.3 Presentations

A PLUS System demonstration has been performed at the Third International Workshop on User Modeling (UM '92) in August '92 at Schlotf Dagstuhl, Germany (cf. [Andre et al. 92]).

At the following IBM-internal conferences, presentations of the PLUS System have been performed:

- ITL on Expert Systems (see above)
- European Object-oriented Software Symposium (see above)
- ITA Expert Systems, April and December '91, Stuttgart and Böblingen.

Furthermore, a lot of demonstrations of the PLUS System have been carried out both in various departments of the IBM Laboratory Böblingen and at the DFKI.

Bibliography

[Andre et al. 92] E. **Andre,** R. Cohen, W. **Graf,** B. Kass, C. **Paris,** und W. **Wahlster** (Hrsg.). *UM'92, Third International Workshop on User Modeling,* Saarbrücken, Germany, August 1992. DFKI.

[Bauer et al. 91] M. **Bauer, S. Biundo, D. Dengler, M. Hecking, J. Köhler,** und G. **Merziger.** *Integrated Plan Generation and Recognition - A Logic-Based Approach.* In: W. Brauer und D. Hernandez (Hrsg.), Verteilte Künstliche Intelligenz und kooperatives Arbeiten. 4. Internationaler GI-Kongress Wissensbasierte Systeme, Berlin, Heidelberg, 1991. Springer. Also DFKI Research Report RR-91-26.

[Bauer et al. 92] M. **Bauer, S. Biundo, D. Dengler, J. Koehler,** und **G. Paul.** *PHI-A Logic-Based Tool for Intelligent Help Systems.* Research Report RR-92-52, DFKI, 1992.

[Berger & Thies 92] F. Berger und M. A. **Thies.** *Developing a Plan Base for the PLUS Help System with, PlanEdit*$^{+}$. Document, German Research Center for Al (DFKI), Saarbriicken, Germany, 1992.

[Braune 92] H. **Braune.** *PLUS (PLan-based User Support) - Entwicklung and Implementierung von Tools.* Diplomarbeit, Fachhochschule Furtwangen, Germany, 1992.

[Breuker90] J. Breuker (Hrsg.). *EUROHELP, Developing Intelligent Help Systems.* Kopenhagen, Amsterdam: EC, 1990.

[Dengler et al. 87] D. **Dengler, M. Gutmann,** und **G. Hector.** *Der Planerkenner REPLIX.* Memo 16, Institut für Informatik, Universität des Saarlandes, September 1987.

[Fehrle & Thies 91] T. **Fehrle** und **M. A. Thies.** *InCome: A System to Navigate through Interactions and Plans.* In: H.-J. Bullinger (Hrsg.), Human Aspects in Computing: Design and Use of Interactive Systems and Information Management, Amsterdam, London, New York, Tokyo, 1991. Elsevier Science Publishers B.V.

[Fehrle 90] T. **Fehrle.** *PLanbased User Support (PLUS) Projektbeschreibung.* IBM intern, IBM Laboratory, Böblingen, Germany, 1990.

[Finin 83] T. W. **Finin.** *Providing Help and Advice in Task Oriented Systems.* In: Proceedings of the 8th International Joint Conference on Artificial Intelligence, S. 176-178, Karlsruhe, Germany, 1983.

[Fischer et al. 85] G. **Fischer,** A. **Lemke,** und **T. Schwab.** *Knowledge-based Help Systems.* In: Proceedings of the CHF85 Conference on Human Factors in Computing Systems, acm Press, 1985.

[Hirschmann 90] A. **Hirschmann.** *Das Hilfesystern MATHILDE.* Dissertation, Universität Regensburg, 1990.

[IBM 91] **IBM.** *Common User Access, Advanced Interface Design Guide.* Systems Application Architecture. International Business Machines Corporation, 1991. SC34-4289-00.

[IBM 92a] **IBM.** *MVS/ESA SP 4.3 Hardware Configuration Definition User's Guide.* International Business Machines Corporation, 1992. GC33-6457-03.

[IBM 92b] **IBM.** *ScreenView User's Guide.* IBM Systems Application Architecture, Screen View. International Business Machines Corporation, 1992. SC33-6451-00.

[Kobsa & Wahlster 89] A. **Kobsa** und W. **Wahlster** (Hrsg.). *User Models in Dialog Systems.* Symbolic Computation. Berlin, Heidelberg, New York: Springer, 1989.

[Koehler 92] J. **Koehler.** *Towards a logical treatment of plan reuse.* In: Proceedings of the 1st International Conference on Artificial Intelligence Planning Systems, S. 285-286, Washington, D.C., 1992. Morgan Kaufmann, Menlo Park.

[Neiman 82] **D. Neiman.** *Graphical Animation from Knowledge.* In: Proceedings of the 2nd National Conference of the American Association for Artificial Intelligence, Pittsburgh, PA, 1982. AAAI Press.

[Paul 89] H. **Paul.** *Exploratives Agieren in interaktiven EDV-Systemen.* In: B. Endres-Niggemeyer, T. Herrmann, A. Kobsa, und D. Rosner (Hrsg.), Interaktion und Kommunikation mit dem Computer. Informatik Fachbericht 238. Berlin: Springer Verlag, 1989.

[Quast 91] K.-J. **Quast.** *PLANET, Planerkennung mit aktivierten Handlungsnetzen.* Sankt Augustin: GMD, 1991.

[Rathke 87] M. **Rathke.** *UNDO/REDO - Szenarien and Anforderungen für eine anwendungsneutrale Implementierung.* In: M. Paul (Hrsg.), GI - 17. Jahrestagung Computerintegrierter Arbeitsplatz im Büro, Berlin, Heidelberg, New York, London, Paris, Tokyo, 1987. Springer.

[Rathke 89] M. **Rathke.** *Erweiterung interaktiver Anwendungen um Undo-Mechanismen.* In: Software Ergonomie: Aufgabenorientierte Systemgestaltung und Funktionalität, GI Band 32, Stuttgart, 1989. Teubner.

[Rich 89] E. **Rich.** *Stereotypes and User Modeling.* In: Kobsa und Wahlster [Kobsa & Wahlster 89], S. 35-51.

[Scheidel 92] H. **Scheidel.** *Intelligentes Online-Tutorial mittels planbasierter Verfahren.* Diplomarbeit, Fachbereich Informatik, Universität Stuttgart, Germany, 1992.

[Shneiderman 83] B. **Shneiderman.** *Direct Manipulation: A step beyond programming Languages.* IEEE Computer, 16, 1983.

[Shneiderman 87] B. **Shneiderman.** *Designing the User Interfaces: Strategies for effective Human-Computer Interaction.* Massachusetts: Addison Wesley, 1987.

[Sukaviriya & Foley 90] P. **Sukaviriya** und J. D. **Foley.** *Coupling a UI Framework with Automatic Generation of Context-Sensitive Animated Help.* In: Proceedings of the ACM SIGGRAPH Symposium on User Interface Software (UIST'90), New York, 1990. ACM SIGGRAPH, acm Press.

[Sukaviriya 88] P. **Sukaviriya.** *Dynamic Construction of Animated Help from Application Context.* In: Proceedings of the ACM SIGGRAPH Symposium on User Interface Software (UIST[7]88), New York, 1988. ACM SIGGRAPH, acm Press.

[Thies & Berger 92a] M. A. **Thies** und **F. Berger.** *Plan-Based Graphical Help in Object-Oriented User Interfaces.* In: T. Catarci, M. F. Costabile, und S. Levialdi (Hrsg.), Proceedings of the International Workshop AVI'92, Advanced Visual Interfaces, Band 36: World Scientific Series in Computer Science, Rome, Italy, May 1992. World Scientific.

[Thies & Berger 92b] M. A. **Thies** und **F. Berger.** *Planbasierte graphische Hilfe in objektorientierten Benutzungsoberflächen.* In: K. Kansy und P. Wißkirchen (Hrsg.), Innovative Programmiermethoden für Graphische Systeme, Berlin Heidelberg New York, 1992. Springer-Verlag.

[Thies & Berger 92c] M. A. **Thies** und F. **Berger.** *The PLUS System - A Plan-Based Help System.* System specifications, German Research Center for AI (DFKI), Saarbrücken, Germany, 1992.

[Thies & Berger 93] M. A. **Thies** und **F. Berger.** *Task-Oriented User Assistance for Interactive Graphical Environments.* In: Proceedings of the 5th International Conference on Human-Computer Interaction, HCI International '93, Orlando, FL, USA, August 1993. forthcoming.

[Thies 90] M. A. **Thies.** *Interaction Control Manager: Ein System zum Navigieren durch Interaktionen und Plane.* Diplomarbeit, Fakultät Informatik, Universität Stuttgart, 1990.

[Thies 92] M. A. **Thies.** *InCome+ - User's Guide.* Technical memo, German Research Center for AI (DFKI), Saarbrücken, Germany, 1992.

[Thies 93] M. A. **Thies.** *Animated Help as a Sensible Extension of a Plan-Based Help System.* In: Proceedings of the 5th International Conference on Human-Computer Interaction, HCI International '93, Orlando, FL, USA, August 1993. forthcoming.

[Wahlster & Kobsa 89] W. **Wahlster** und A. **Kobsa.** *User Models in Dialog Systems.* In: Kobsa und Wahlster [Kobsa & Wahlster 89], S. 4-34.

[Wahlster et al. 93] W. **Wahlster, D. Dengler, M. Hecking,** und **C. Kemke.** *SC: The SINIX Consultant* In: P. Norvig, W. Wahlster, und R. Wilensky (Hrsg.), Intelligent Help Systems for Unix - Case Studies in Artificial Intelligence. Heidelberg: Springer, 1993. forthcoming.

[Wilensky et al. 84] R. **Wilensky, Y. Arens,** und **D. Chin.** *Talking to UNIX in English: An Overview of UC* Communications of the ACM, 27(6), June 1984.

[Wilensky et al. 88] R. **Wilensky, D. N. Chin, M. Luria, J. Martin, J. Mayfield,** und D. **Wu.** *The Berkeley UNIX Consultant Project.* Computational Linguistics, 14:35-84, 1988.

[Yang 90] Y. Yang. *Current Approaches & New Guidlelines for Undo Support Design.* In: H.-J. Bullinger uncl B. Shackel (Hrsg.), Human-Computer Interaction - INTERACT90, North-Holland, 1990. Elsevier Science Publishers B.V.