TUM

INSTITUT FÜR INFORMATIK

Generating User Interfaces with the FUSE–System

Frank Lonczewski
Siegfried Schreiber

TUM–I9612
Februar 1996

TECHNISCHE UNIVERSITÄT MÜNCHEN

TUM-INFO-02-1996-I9612-200/1.-FI

Typescript: ---

Druck: Mathematisches Institut und
Institut für Informatik der
Technischen Universität München

Generating User Interfaces with the FUSE–System

Frank Lonczewski
Siegfried Schreiber

Institute of Computer Science, Munich University of Technology,
Arcisstr. 21, 80290 Munich, Germany
email: {lonczews,schreibs}@informatik.tu-muenchen.de
WWW: http://www2.informatik.tu-muenchen.de/research/ui/ui.html

Abstract

With the FUSE(Formal User interface Specification Environment)–System we present a methodology and a set of integrated tools for the automatic generation of graphical user interfaces. FUSE provides tool–based support for all phases (task–, user–, problem domain analysis, design of the logical user interface, design of user interface in a particular layout style) of the user interface development process. Based on a formal specification of dialogue– and layout guidelines, FUSE allows the automatic generation of user interfaces out of specifications of the task–, problem domain– and user–model. Moreover, the FUSE–System incorporates a component for the automatic generation of powerful help– and user guidance components. In this paper, we describe the FUSE–methodology by modeling user interfaces of an ISDN phone simulation. Furthermore, the two major components of FUSE (BOSS, PLUG–IN) are presented: The BOSS–System supports the design of the logical user interface and the formal specification of layout guidelines. PLUG–IN (PLan–based User Guidance for Intelligent Navigation) generates task–based help– and user guidance components.

1 Introduction

Even with the most advanced layout oriented user interface construction tools (UI–toolkits, UI–builders, UI–Management Systems) the task–based and user–oriented development of graphical user interfaces remains a time–consuming and difficult process. Therefore tools for the formal specification and automatic generation of user interfaces (model based UI–tools) have gained rising research interest. Regarding the evolution of model based tools from the early approaches (e.g. MIKE, MIKEY, HIGGENS) to the most recent ones (e.g. MASTERMIND, TRIDENT, TADEUS) we recognize that more and more phases of the user interface development process are supported.

The FUSE (Formal User interface Specification Environment)–System described in this paper belongs to this new generation of model based interface tools. The main goals and properties of the FUSE–System are:

- Tool–based support for all phases (task–, user–, problem domain–analysis, design of the logical user interface, design of user interfaces in a particular layout style) of the user interface development process
- Generation of working prototypes in early phases of the development process
- Standardization of user interfaces by formal specification of user interface styleguides
- Generation of powerful help– and user guidance components

The paper is organized as follows: In section 2 we describe the overall methodology and architecture of the FUSE–System. In section 3 we discuss related work in the area of automatic UI generation. In section 4 we discuss the capabilities of FUSE for the generation of user interfaces in different layout styles and of help– and user guidance components by using the example of an ISDN phone simulation. Section 5 describes the stages in the development process of the ISDN interface using the FUSE–System. In section 6 we discuss practical experience with FUSE and directions of further research.

2 The FUSE-Methodology: An Overview

The overall architecture of the FUSE–System is shown in figure 1. The FUSE–System [15] consists of the four components BOSS (BedienOberflächenSpezifikationsSystem, the german translation of "user interface specification system" [20, 21]), FLUID (FormaL User Interface Development, [3]), PLUG–IN (PLan–based User Guidance for Intelligent Navigation [14, 13]) and FIRE (Formal Interface Requirements Engineering). Each of these tools may also be used independently of the FUSE–System.

The user interface development process with FUSE consists of the phases requirements analysis, design and evaluation. For the user interface part of an interactive application no implementation phase is needed, as FUSE generates running user interfaces from design–level specifications. Some activities in the development process have to be carried out only once for a whole class of user interfaces. As these activities mainly refer to the definition of software–ergonomic guidelines (dialogue– and layout guidelines, see figure 1), they belong to the "Guideline Definition Layer" (GDL). The other activities have to be carried out in each user interface development process. As these activities consist mainly of the application of the Guidelines defined in the GDL, they belong to the "Guideline Application Layer" (GAL).

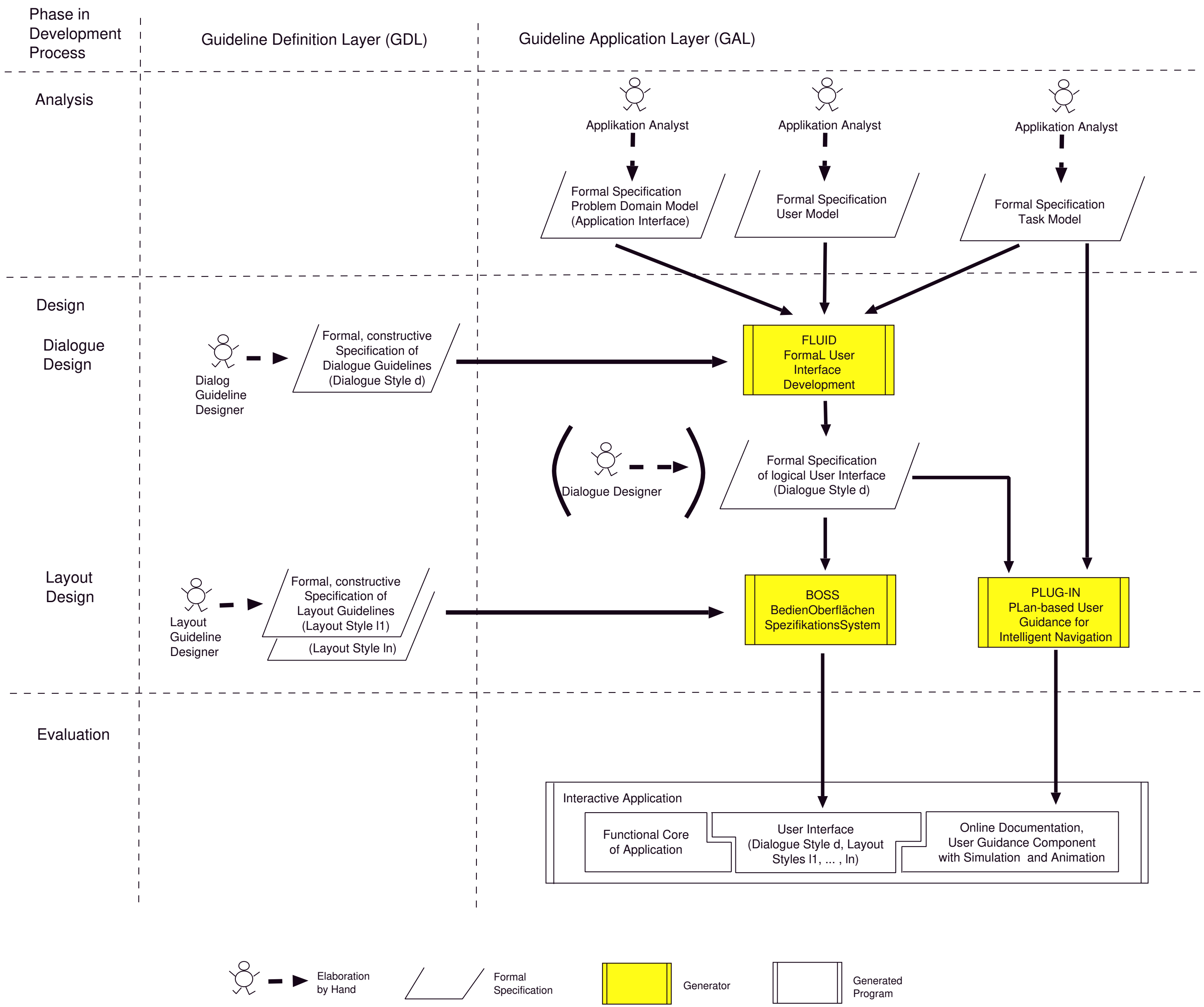

Figure 1: Overall Architecture of the FUSE–System

In the analysis phase of the process, an application analyst defines the requirements for the user interface by setting up the formal specification of three models. The specification of the task model describes the task–hierarchy of the interactive application. The problem domain model (application interface) consists of an algebraic specification of the functions and data structures of the UI–relevant part of the functional core of the interactive application. The user model is a description of static and dynamic properties of user groups and individual users which influence not only the generation process of the user interface, but also the kind and depth of the help offered by the user guidance component on a 'per user' basis. To support the requirements analysis phase the FUSE–System contains a component called FIRE (not shown in figure 1). FIRE provides graphical editors for setting up the task–, problem domain– and user model and a tool for the generation of a very–first UI prototype.

In the design phase of the UI development process, software–ergonomic guidelines are formally specified by human factors experts in the roles of dialogue– and layout guideline designers. Dialogue guidelines describe the transformation of the task–, problem domain– and user models of interactive applications into formal specifications of logical user interfaces in a particular dialogue style. At the abstraction level of logical user interfaces, static and dynamic properties of interfaces are described without considering presentation issues. Layout guidelines describe the transformation from specifications of logical user interfaces into specifications of user interfaces in a particular layout style. The formal specifications of dialogue– and layout guidelines, which can be regarded as the formal specification of a user interface styleguide, belong to the GDL–layer in the development process, as they have to be carried out once for a whole class of user interfaces.

Within the FUSE–architecture, the FLUID–System plays the role of an automatic dialogue designer. From the specifications of dialogue guidelines for a dialogue style d and the specifications of task–, problem domain– and user model of an interactive application, FLUID generates the specification of static and dynamic properties of a logical user interface in the dialog style d. This specification may be modified by a human dialogue designer. For the representation of the design of the logical user interface, FUSE employs a specification technique called HIT (Hierarchic Interaction graph Templates), which is based on attribute grammars and dataflow diagrams. Besides the automatic generation with the FLUID–System, a human dialogue designer can elaborate the specification of the logical user interface by hand.

From the specifications of layout guidelines for the layout styles $l_1, ..., l_n$ and the specification of a logical user interface (generated automatically by FLUID or specified by a human dialogue designer) in a dialogue style d the BOSS–System generates an implementation of a user interface in the dialogue style d, in which the end–user can switch at run–time between the layout styles $l_1, ..., l_n$. For the formal specification of the layout guidelines the BOSS–System also uses the HIT specification technique. The separation between logical UIs and interfaces in particular layout styles in the design phase (see figure 1), which is typical for model based UI tools, can be found also in related research domains like document architecture (see e.g. [5, 22]).

Based on the specification of the task–model and the specification of the dynamics of the logical user interface generated by FLUID the PLUG–IN–System generates a component for intelligent user guidance, which is bound (i.e. "plugged in") to the user interface implementation generated by BOSS. This user guidance component supports the end–user during his work with the user interface by context–sensitive hypertext help–pages. These provide information about the current state of the user interface. Moreover, the generated user guidance component uses animation sequences to demonstrate how complex tasks can be accomplished by the user.

3 Related Work

MIKE [18] (Menu Interaction Kontroll Environment) und MIKEY [19] generate user interfaces with menus and dialog boxes based on a description of the functions (argument– and result parameters) and the data structures in the application interface. In HIGGENS [10] a semantic data model of the application interface is used as the base for deriving views as abstract descriptions of the user interface layout.

The ITS (Interactive Transaction System)–System [24] offers a frame–based language for the specification of user interfaces in its logical structures ("dialogue content"). Moreover, ITS allows the specification of style rules, which describe the mapping between logical user interfaces and user interfaces in a particular style.

In the UIDE–System (User Interface Design Environment) [8], the UI development process consists of the description of two models. In the application model, the logical user interface is described in terms of application objects and –tasks. The UI–model describes the coupling of the application model to a user interface layout by linking application tasks to interface tasks, interaction techniques and –objects. The links between the models are used by a runtime engine to provide animated help.

HUMANOID [16] divides the UI–development process into the activities application design, dialogue sequencing, action side effects, presentation design and manipulation design. In the first three design dimensions the logical structure of a user interface is described in terms of the structure and the behaviour of so called application objects. The mapping of the state of the application objects in an logical user interface to a UI–layout is described in the design dimensions presentation– and manipulation design through presentation and manipulation templates. Based on the model described above, HUMANOID is able to provide textual help. Recently the research on UIDE and HUMANOID was joint in the MASTERMIND project.

In the ADEPT–System [12], a process–algebra–like specification technique (TKS, task knowledge structures) is used for the specification of the task model of an interactive application. In the design phase of the UI–development process, the task model is transformed into the specification of the so called AIM (abstract interface model), which corresponds to the term “logical user interface” in figure 1. Based on design rules in a user model, the ADEPT–System derives a concrete interface model (CIM) from the AIM by replacing the abstract interaction objects in the AIM by the appropriate concrete interaction objects in the CIM.

The GENIUS–System (GENerator for user Interfaces Using Software Ergonomic Rules) [11] generates user interfaces for data–base oriented applications. In GENIUS, the problem domain model is represented by an ERA (entity–relationship–attribute) diagram. Based on this ERA–diagram static aspects of the logical user interface are described in terms of so called views, which can be regarded as abstract representations of user interface windows. For the representation of the dynamics of the logical interface, GENIUS employs a petri–net–like specification technique (“dialogue–nets”). For each view in the logical user interface, the static UI–layout is generated by applying software–ergonomic guidelines, which are described as decision tables (e.g. for the selection of interaction objects).

In the TADEUS–System (TAsk based DEvelopment of User interface Software) [6], the UI–development process is divided into the phases requirements analysis, dialogue design and realisation. During the phase requirements analysis, a task model (hierarchic goal structure), a problem domain model (class hierarchy) and an user model is specified. In the phase dialogue design, static and dynamic aspects of the logical user interface are described in terms of views and dialogue–graphs (an extension of the dialogue–nets of GENIUS). In the phase realisation, the logical user interface is transformed into an user interface description for an UIMS by applying software–ergonomic guidelines specified through decision tables.

The TRIDENT(Tools foR an Interactive Development ENvironmenT)–System [4] consists of a methodology and a support environment for developing UIs for business–oriented interactive applications. TRIDENT uses ERA–diagrams for the description of the problem domain model. For the representation of the task model TRIDENT provides a data–flow–graph–like specification technique (activity chaining graphs, ACGs). Each ACG is structured into presentation units. From these presentation units, the static user interface layout can be generated by applying rules for the selection of abstract interaction objects (AIO), rules for mapping AIO to concrete interaction objects (CIO) and rules for the placement of CIO.

The JANUS–System [2] uses OOA (Object–Oriented Analysis) for describing the problem domain model (i.e. application interface) of a database–oriented interactive application. Moreover, JANUS allows the specification of software–ergonomic guidelines, which describe the mapping between OOA–models to the UI–description language of a UIMS. JANUS does not provide means for the explicit specification of the UI–dynamics.

PLUS [7] is a task-oriented help system for domain-specific interactive applications. It uses a database of hierarchical plans described by an application analyst. With this database the system reasons about the hypothetical tasks the user currently performs.

In the task description knowledge about application– and user interface specific knowledge is combined. Therefore the given help is tailored to a predefined layout style. If the application functionality or layout guidelines are changed, the database has to be changed accordingly by hand.

The differences between FUSE and the approaches presented above include the flexibility of the WWW–based help– and user guidance components generated by PLUG–IN. Moreover, FUSE achieves a high degree of integration and tool–based support over the whole UI–development process. The use of an encompassing, intuitive specification technique (HIT) in the UI design phase (see figure 1) eases the use of the FUSE–System.

4 User Support for an ISDN Phone with FUSE

In this section we present examples of different ISDN phone user interfaces generated with the BOSS system. The ISDN phone simulation is an interactive application for the simulation of main operations that are possible with a real ISDN telephone simulation described in [1]. Furthermore we look onto the problems that the user can possibly have when using one or more of these user interfaces. We also show the various kinds of help that PLUG-IN provides for the ISDN phone simulation.

4.1 User Interfaces of an ISDN Phone

With the ISDN phone simulation the user can accomplish a number of tasks with different complexity. An example of a simple task is $Create1stConnection$. This task can be decomposed into the subtasks $Start1stConnection$ and $DialTelephoneNumber$. If the user at the other end responds, the two parties are connected to each other afterwards. Other example tasks of the ISDN phone simulation are: $DefineDirectCallButton$, $EstablishConference$ and $EstablishConnectionBetweenOtherParties$.

In figure 2 and figure 3 we can see two different user interfaces of the ISDN phone simulation. In figure 3 the direct manipulation interface is displayed. The elements of

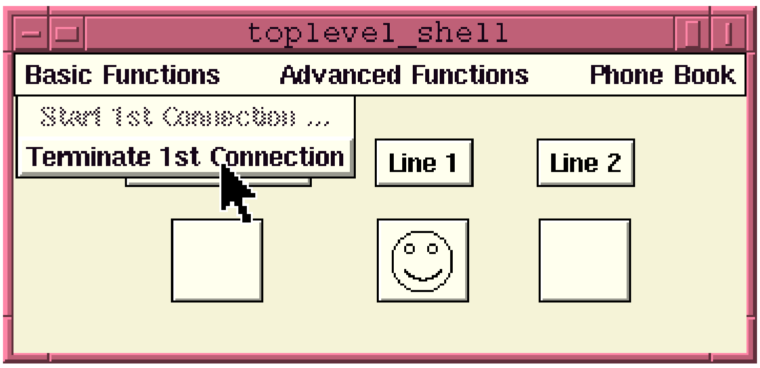

Figure 2: menu interface

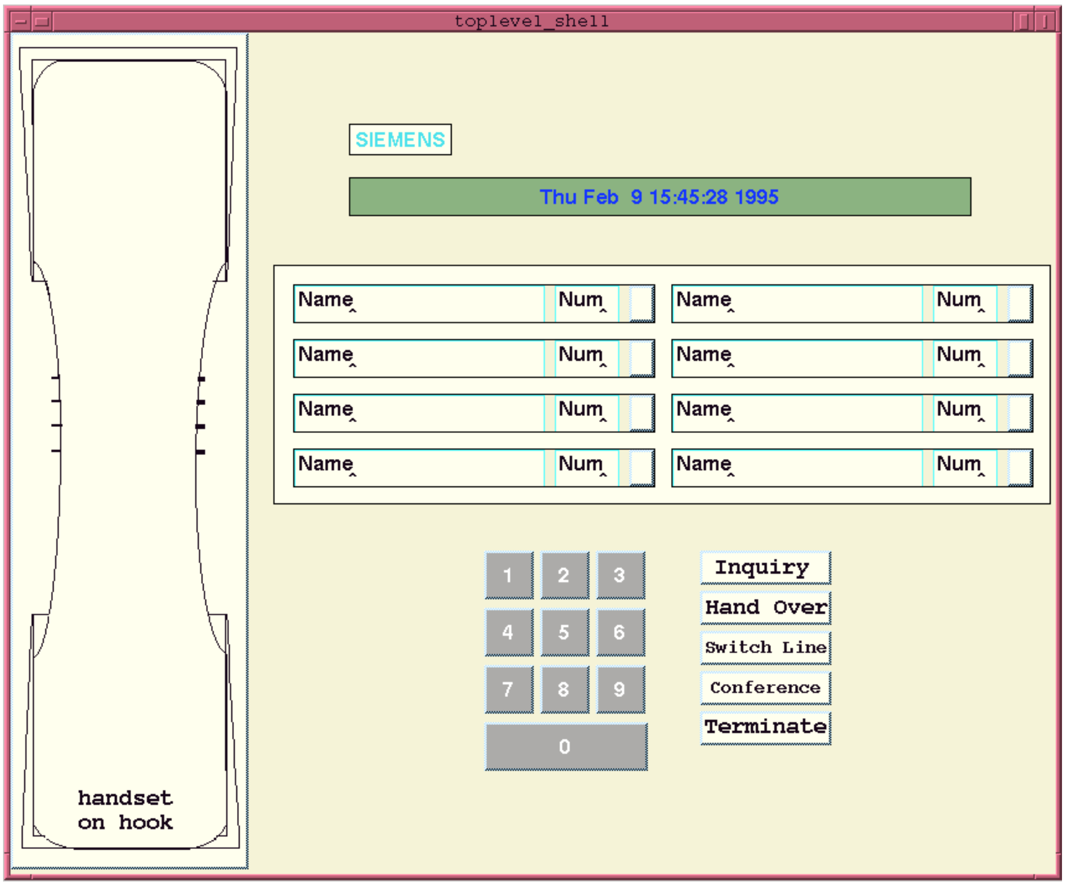

Figure 3: direct manipulation interface

the user interface are a handset button, a liquid crystal display, eight direct call buttons (each one with name and phone number label), a digit block and five special function buttons. A phone number can be entered by using the digit block or one of the direct call buttons. If a direct call button is used, a predefined phone number associated with the button is dialed. Figure 2 displays a functional equivalent menu interface for the phone simulation. Both user interfaces are generated with the BOSS system. The user can change the layout between the two styles presented above during runtime.

The alternative user interface of the ISDN phone simulation in figure 2 consists of a menupane with three menus named *BasicFunctions*, *AdvancedFunctions* and *PhoneBook*. In the first one the basic phone functions (e.g. to start a phone call) are listed, whereas the more complex functions (e.g. to create a connection to a second party while already connected to a first one) can be found in the second menu. With the third menu the phonebook of the ISDN simulation can be administered. In comparison to the direct manipulation interface the menu interface displays the state of the phone simulation more explicitly by displaying icons under the three labels *ExternalLine*, *Line* 1 and *Line* 2. These are helpful for the user as the state of the phone simulation can be deduced from them. As a smiling face is displayed for *Line* 1, the user is currently connected to a party on the first of two available phone lines. If the state of the phone simulation changes, the displayed icons change accordingly (e.g. if the user terminates the phone call, the smiling face will disappear).

One of the more complex functions of the ISDN phone simulation is *StartConference*. In an ISDN phone conference the three participating parties can talk and hear each other simultaneously. Despite of the fact that a *Conference* button is available on the direct manipulation user interface (and similarly a *StartConference* menu entry in the menu *AdvancedFunctions* of the menu interface), it is a complex task to establish a conference with the phone. As the interactive phone application simulates the behaviour of a real ISDN phone [1], it is not as easy as just pressing the conference button on the user interface. If the phone is not in an appropriate state, only the message "Conference not possible" is shown on the LCD. In this situation the user would look into the reference guide of the phone trying to find out how to establish the conference. While working with an interactive simulation, a user guidance component can offer even more than an on-line reference guide in hypertext form.

PLUG-IN supports the user of interactive applications by dynamical on-line help and task-based user guidance. For this purpose all interactions of the user are observed.

4.2 Task–Based User Support with PLUG-IN

For the task-based user guidance PLUG-IN tries to determine the current tasks of the user while she is working with an interactive application. If the observed interactions can be matched with parts of a task valid in the current state, a way is searched to solve the identified task. A task can be accomplished if its task–goal can be reached. Typically a unique (sub)state of the user interface is associated with each task goal. If the task can be performed in the current state, the user guidance component helps the user by:

- generating an animation sequence (upon user request) that simulates the necessary user interactions to accomplish the given task
- updating and visualizing a list of tasks that the user is currently performing out of the view of the user guidance component

To provide the task-oriented help, an application analyst first describes the tasks that can be performed with the interactive application by creating the task model. It contains a layout independent description of the tasks that the user can accomplish with the application. A task description of the ISDN phone simulation is presented in section 5.1.

A list of ISDN phone tasks is shown in figure 4. If the user selects a task from the list, the necessary interaction sequences are simulated on the user interface. Entries of the list are highlighted if PLUG-IN has identified a corresponding task as currently being performed by the user.

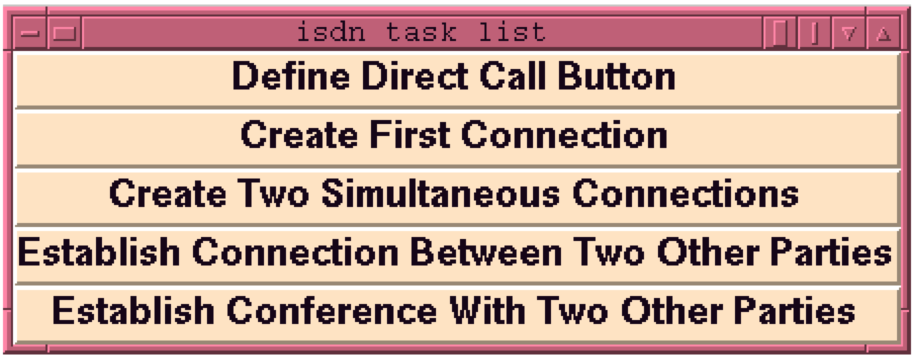

Figure 4: task list for ISDN phone simulation

4.3 Dynamical On-Line Help with PLUG-IN

The dynamical on-line help is based on the various possible states and state transitions of the interactive application. As not all possible application states and transitions are interesting from the user's point of view, only those relevant for the user support are taken into account. This subset can be derived by using the information coded into the task–, problem domain– and application–model and can be represented as a set of finite state automatons. The information contained in these can be used to:

- generate help pages (see below)
- visualize the set of finite state automatons as State Transition Diagrams (STDs)
- generate animation sequences that simulate the necessary user interactions to change the state of the application to another state selected by the user.

The task-based and the dynamical on-line help are closely coupled. Both of them are used for the provision of the dynamical on-line help.

As an example a STD for the ISDN telephone application is shown in figure 5. The highlighted node *NoConnection* describes the current state of the phone. The actions that the user can perform in the different states are denoted by directed arcs. In the current state the user can only start a phone call. PLUG-IN uses the set of state transition diagrams to generate dynamical on-line help pages and animation sequences. In contrast to other approaches to user guidance [7, 17], PLUG-IN generates dynamical on-line help pages in HTML format that can be inspected with a World Wide Web browser like Netscape or NCSA Mosaic. One dynamical on-line help page is displayed in figure 6.

Each dynamical on-line help page is typically divided into four regions and contains:

- information about the current state of the application from the user's point of view

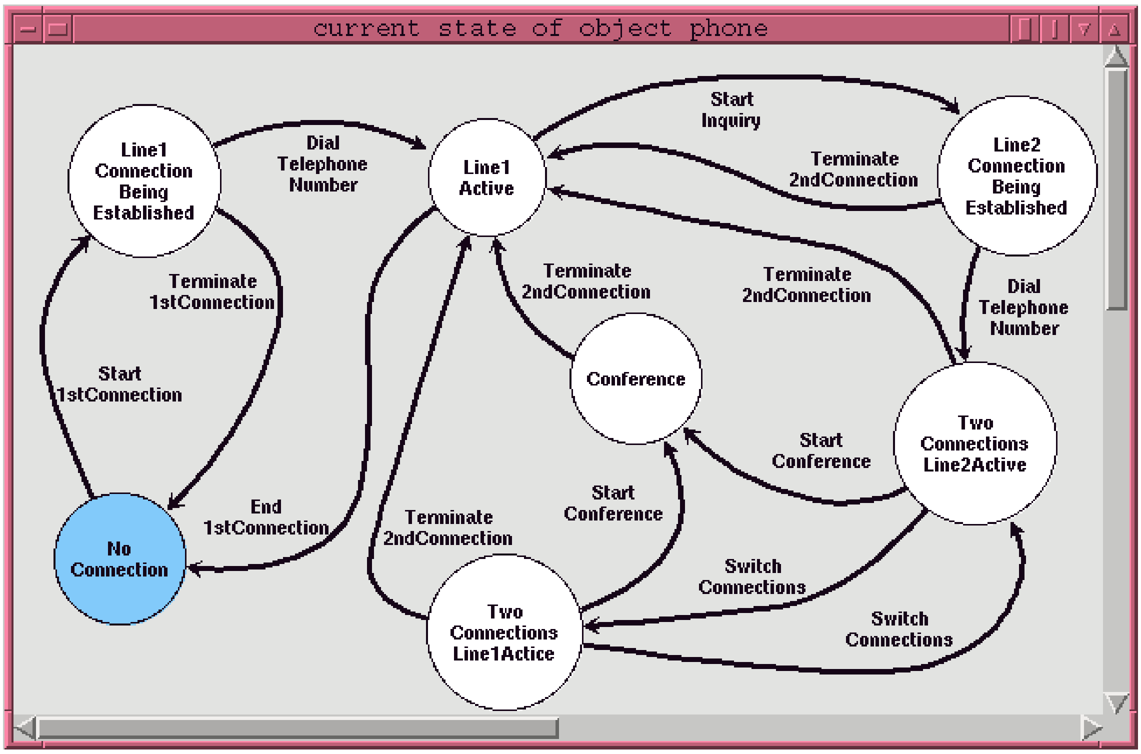

Figure 5: STD for ISDN phone simulation

- information about the set of possible actions the user can perform in the current state
- for each of the possible actions: information about the necessary user interactions to perform the action.
- information about further documentation material, e.g. references to a hypertext version of the user manual of the application

As all operations the user can perform on the original user interface can also be triggered through the WWW browser, it can be regarded as an alternative user interface of the application. In contrast to the original user interface the goal of the WWW-based user interface is to guide the user during the work with the application. The information displayed helps the user to accomplish a given task. Furthermore, the user can learn how to interact with the original user interface through the means of the simulation capabilities of PLUG-IN.

Depending on the current state of the application and the chosen layout style for the user interface, PLUG-IN generates different on-line help pages on the fly. The generated on-line help page corresponding to the current state of the phone's user interface shown in figure 3 is displayed in figure 6. If the user selects the light bulb icon on the page, the

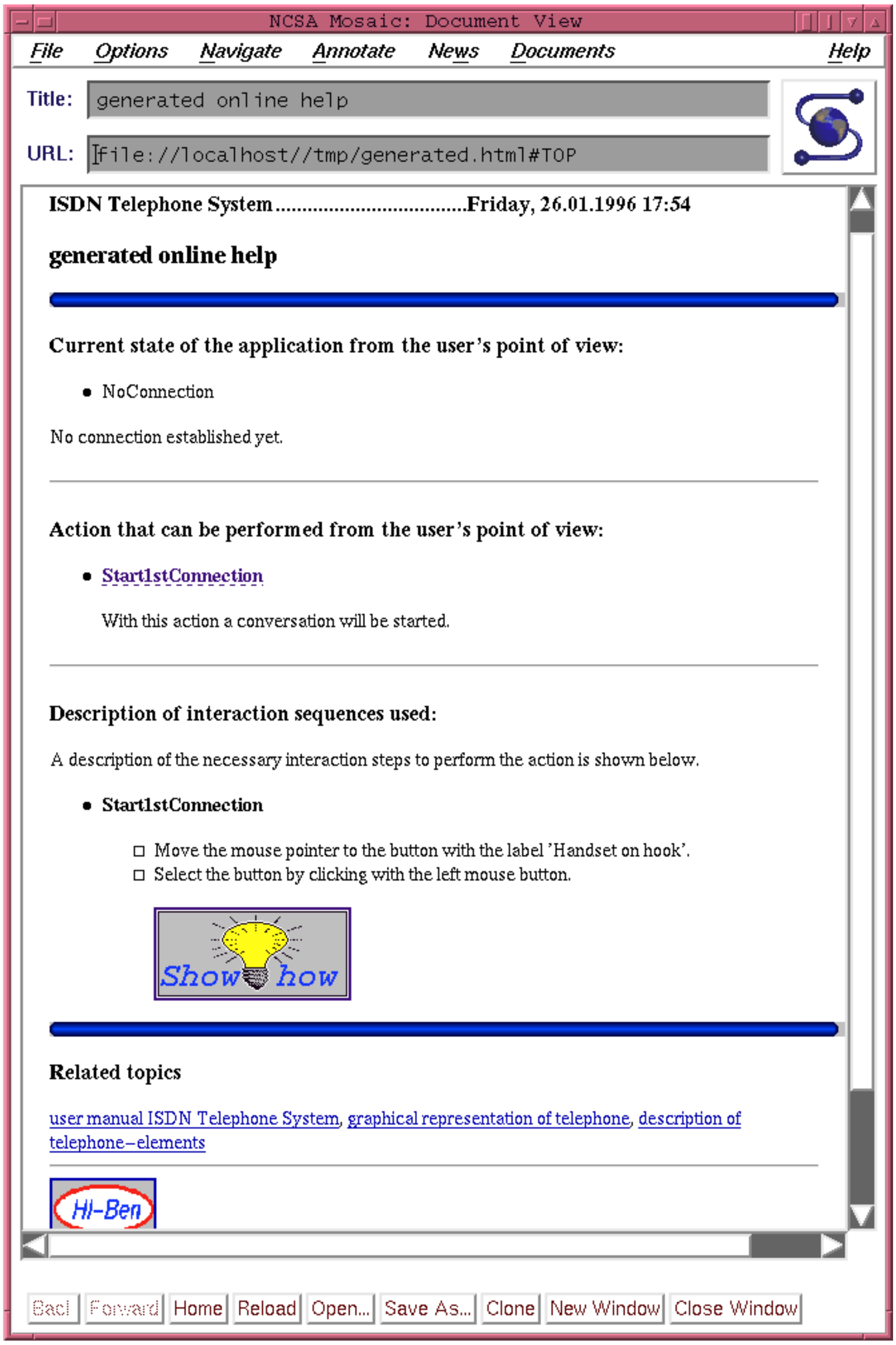

Figure 6: one dynamical on-line help page generated by PLUG-IN

described user interactions are animated on the user interface. In this example PLUG-IN would take control over the mouse pointer, then changes the shape of the mouse pointer to provide visual feedback for the user. Afterwards it moves the mouse pointer to the handset button on the user interface and selects the button by simulating a click with the left mouse button. Finally, a new on-line help page is generated and displayed with the WWW browser.

The user can also interact with the displayed STD. Here he simply selects a state node and PLUG-IN searches a path to the selected node. If a path can be found, the corresponding user interaction are animated on the displayed user interface.

PLUG-IN has the capability to deal with different user interface layouts with regard to the generated on-line help pages and animation sequences. If the layout style of the user interface changes during runtime, the description of the necessary interaction steps on the dynamical on-line help pages are altered correspondingly. Also the generated animation sequences are tailored to the new layout style. It would be very hard to build a help system by hand that provides the various kinds of help offered by PLUG-IN, because the designer has not only to take into account the various possible states of the interactive application, but also the various layout styles that can be changed during runtime. The approach used within PLUG-IN is very flexible, because the help offered adapts itself automatically to the runtime context.

It is worth to be mentioned that PLUG–IN can be used independently of the FUSE–System. In this case, PLUG–IN provides a comfortable environment (e.g. a graphical editor) for creating the required STDs. Within FUSE the FLUID–System [3] will provide the automatic generation of these STDs by using the information from the task–, problem domain– and user model.

5 Modeling the ISDN User Interface with FUSE

In the following we describe the systematic development of the ISDN user interfaces described in section 4. In section 5.1 we demonstrate how the task– and problem domain models are represented during the requirements analysis. In section 5.2 we focus on the design of the ISDN UIs using the BOSS–System. In this context we show how the logical ISDN UI is designed by a human dialogue designer "by hand" without using the FLUID–System. The use of the FLUID–System for generating an initial design of the logical ISDN UI can be found in [3].

5.1 Defining the requirements for the ISDN UI

During the phase "requirements analysis" in the UI–development process (see figure 1) the application analyst defines the requirements for the UI in terms of a problem domain–, task– and user model.

In the FUSE–System, the conceptual objects and functions of the problem domain model (Application Interface, AI) are represented as an algebraic specification $Spec_{AI} =< \Sigma_{AI}, Ax_{AI} >$. The signature part Σ_{AI} consists of the definitions of sorts with associated constructor– and selector functions for the description of the conceptual problem–domain objects. Furthermore Σ_{AI} describes the functionality (argument– and result parameters) of the so called interface functions, i.e. functions which end–users apply to conceptual objects.

Figure 7 shows the sorts, constructor– and selector functions of the problem do-

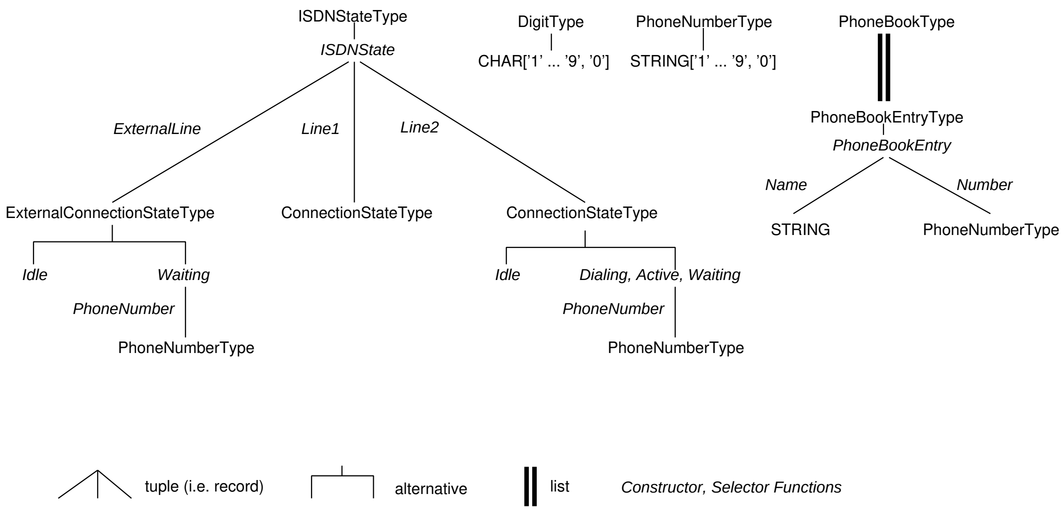

Figure 7: Sorts, Constructor– and Selector Functions for the ISDN Phone

main model of the ISDN phone in the graphical notation used in the FUSE–System. The sort $ISDNStateType$ describes the set of possible states of an ISDN phone. $ISDNStateType$ is defined as a tuple with the components $ExternalLine$ (state of the external line), $Line1$ and $Line2$ (state of the two internal lines). The sort *ConnectionStateType* describes the possible states $(Idle, Dialing, Active, Waiting)$ of an internal line. The sort $PhoneBookType$ describes the phone book, an abstraction of the direct call buttons, as a list of elements of the sort $PhoneBookEntryType$ (tuple with components $Name$ and $PhoneNumber$). The sort $PhoneNumberType$ defines phone numbers as strings out of the ordered character set $'1'...'9','0'$. This character set is also described by the sort $DigitType$. The functionality of the interface functions in the problem domain model of the ISDN phone is shown in figure 8. The function $start\_1st\_connection$ is used to start a phone call on line 1. The argument parameter sb (state before) denotes the state of the phone before, the result parameter sa (state after) the state after calling $start\_1st\_connection$. The function $terminate\_1st\_connection$ is used to terminate phone calls on line 1. With the function $dial\_with\_db$ the user enters a digit (argument parameter d) of the phone number. With the function $dial\_with\_dcb$ a complete phone number (argument parameter n) is entered with one of the direct call buttons.

The semantic part Ax_{AI} of the algebraic specification $Spec_{AI}$ of the ISDN problem domain model describes the semantics of the interface functions in terms of pre– and postconditions. E.g. for the function $start\_1st\_connection$ we demand the precondition

$$is\_Idle(Line1(sb)) \ \wedge \ is\_Idle(Line2(sb)),$$

```
// ... start and terminate connections on the first line

start_1st_connection: ISDNStateType sb -> ISDNStateType sa
terminate_1st_connection: ISDNStateType sb -> ISDNStateType sa

// ... dial with the digit block (db) or the direct call buttons (dcb)

dial_with_db: ISDNStateType sb, DigitType d -> ISDNStateType sa
dial_with_dcb: ISDNStateType sb, PhoneNumberType n -> ISDNStateType sa

// ... receive connection request

receive_request: ISDNStateType sb, PhoneNumberType n -> ISDNStateType sa

// ... start and terminate inquiries and conferences

start_inquiry: ISDNStateType sb -> ISDNStateType sa
start_conference: ISDNStateType sb -> ISDNStateType sa
terminate_conference: ISDNStateType sb -> ISDNStateType sa
hand_over: ISDNStateType sb -> ISDNStateType sa
terminate_2nd_connection: ISDNStateType sb -> ISDNStateType sa
switch_connections: ISDNStateType sb -> ISDNStateType sa
```

Figure 8: Functionality of ISDN interface functions

i.e. the phone is not in use. After calling *start_1st_connection*, line 1 of the phone should be in the state *Active*, if there was a phone call request on the external line. If there is no such request, line 1 should be in the state *Dialing*. This behaviour is expressed by the axioms

$$\begin{aligned}
&\forall n \in \ PhoneNumberType: \\
&\quad start_1st_connection(ISDNState(Waiting(n), Idle(), Idle())) \ = \ ISDNState(Idle(), Active(n), Idle()) \\
&\quad start_1st_connection(ISDNState(Idle(), Idle(), Idle())) \ = \ ISDNState(Idle(), Dialing(<>), Idle())
\end{aligned}$$

In a similar way, the semantics of the other interface functions is described. Overall, the algebraic specification $Spec_{AI}$ of the ISDN problem domain model describes the state transition diagram shown in figure 5.

While the problem domain model defines the requirements for the user interface from the view of the application functionality, the task model describes the UI–requirements from the view of potential end–users. Its content, the task–space, is a decomposition of tasks into subtasks, actions and associated functions of the application interface. An example is shown in figure 9.

The task *EstablishConference* can be decomposed into the subtasks *Create1stConnection*, *MakeInquiry* and the action *StartConference*. The subtasks and the action have to be performed in the order given from left to right, therefore the *sequence* symbol () is displayed above the task *EstablishConference*. Links from actions of the task space to functions of the application interface are denoted by the symbol $\perp$. Besides the *sequence* other constructs define temporal relations in the task space. For each node pre- and postconditions refering to a particular system state can be used to define the context in which a task or an action is applicable. The task *DialTelephoneNumber* (figure 9) uses the *choice*– construct (). Here the user can choose to dial with the digit block or one of the direct call buttons.

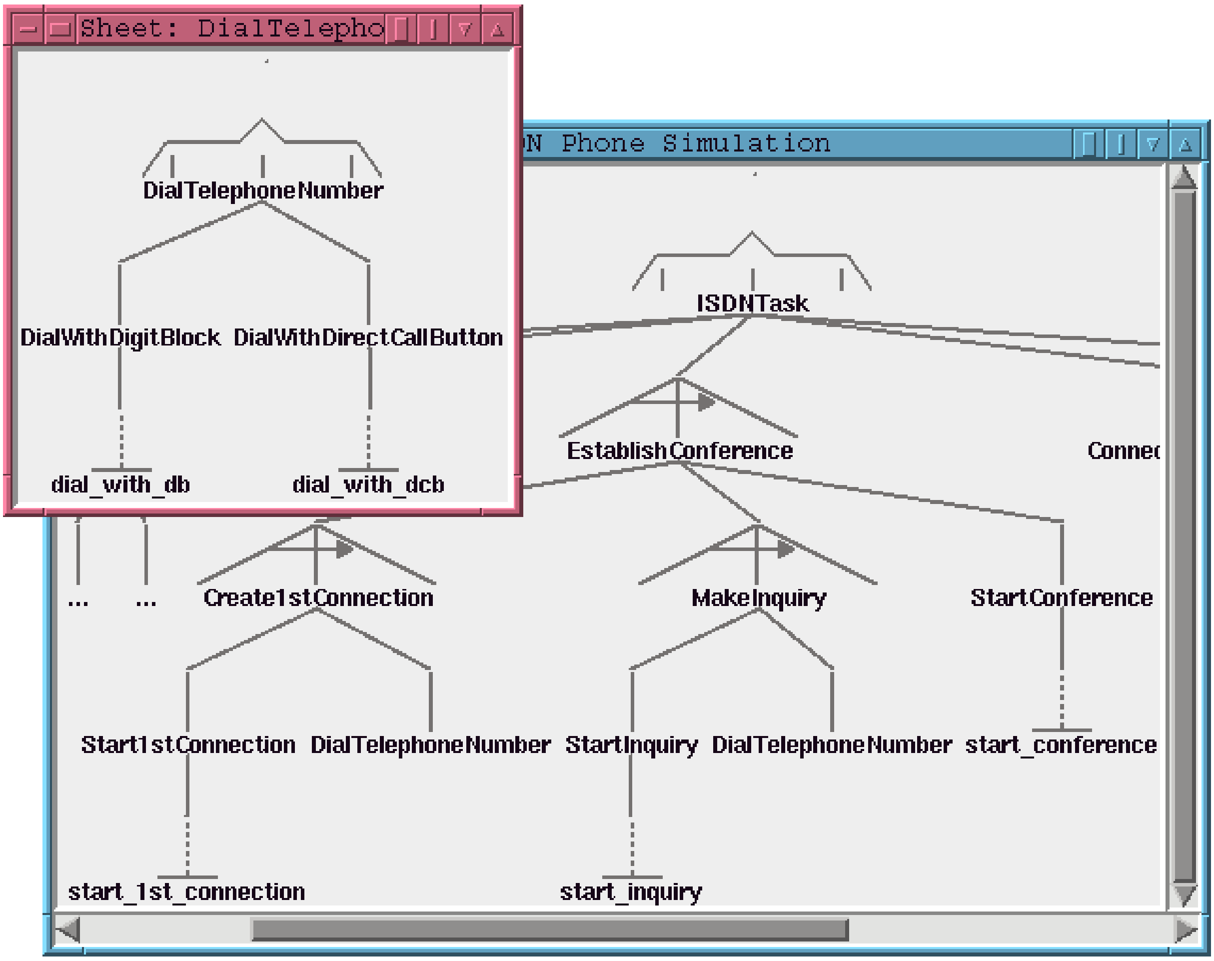

Figure 9: excerpts from task space of phone simulation

In the property sheet of figure 10 the precondition

$$ISDNStateBefore = ISDNState(Idle(), Idle(), Idle())$$

states the fact that this task can be only performed if the phone is not in use. Other properties of the sheet define the behaviour of the user guidance component during runtime.

The requirements analysis phase is completed by defining the static and dynamic properties of the user model. With the static properties of the user model various user stereotypes are modeled. Examples for static properties are "user's motivation", "user's application knowledge" and "user's task knowledge". All of these properties can have one value of the set $\{low, medium, high\}$ and are predefined by the application analyst for a whole user class. In contrast to the static properties the dynamic properties are obtained during runtime. In this way it is possible to give help that is adapted to the user's individual interaction behaviour. One example of a dynamic property is the

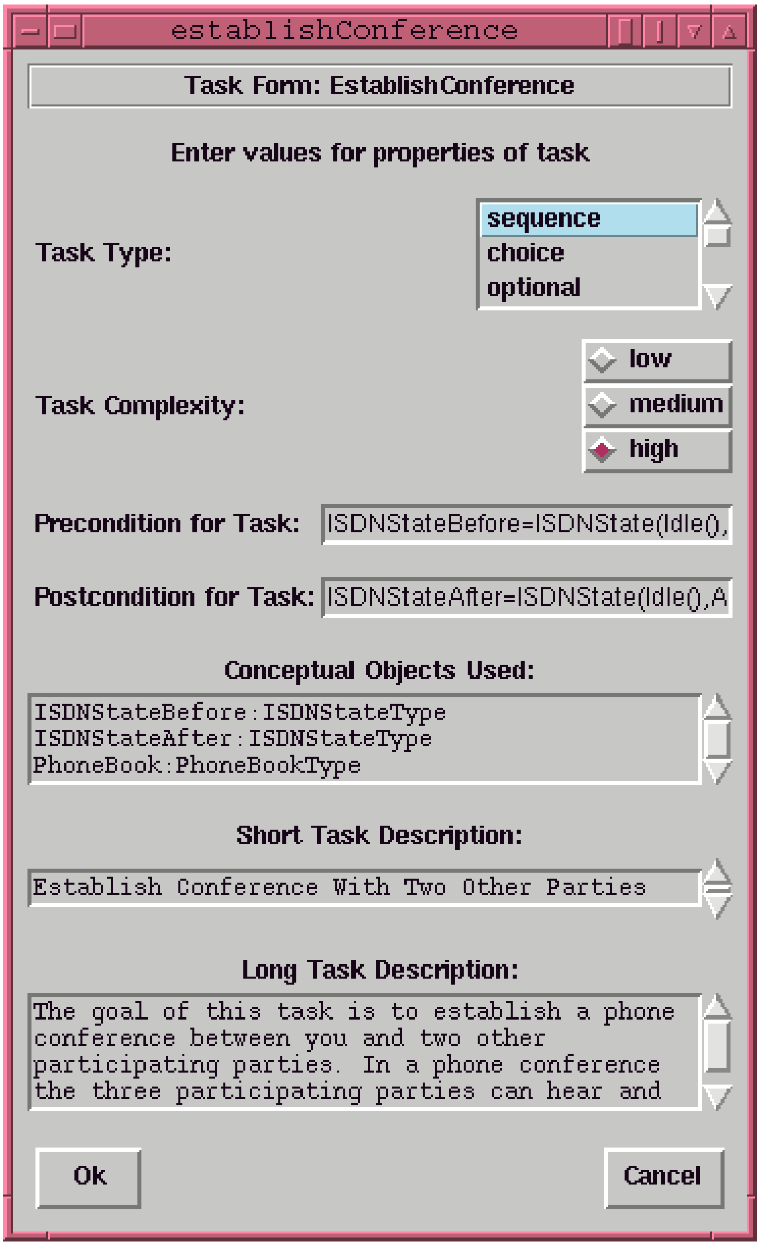

Figure 10: property sheet of example task

"frequency of already solved tasks". With this property it is possible to reason about tasks still unknown to the user. Furthermore the property can be exploited to order the task-based on-line help with respect to the measured task frequency. Overall, the various properties defined in the user model control the behaviour of the user guidance component during runtime.

As mentioned in section 2 the FUSE–System provides a support environment called FIRE, which supports the specification of the three models in the requirements analysis.

FIRE contains not only the graphical editors shown in figures 9 and 10, but also a prototyping tool which generates a very first UI–prototype. This prototype is a direct representation of the task– and problem domain model in terms of menus and dialogue–boxes and provides a good base for discussing the results of the requirements analysis with end–users. Overall the functionality provided by the FIRE–System is similar to the ADEPT–System (see section 3).

5.2 Design of the ISDN UI with BOSS

Within the FUSE–architecture, the BOSS–System is the main tool for supporting the design–phase in the UI–development process. During this design process, BOSS is used by an automatic (i.e. the FLUID–System, see [3]) or a human (the scenario we assume in this paper) dialogue designer for the specification of user interfaces in its logical structures and by a human layout guideline designer for the specification of layout guidelines.

Important properties of BOSS include:

- BOSS uses an encompassing specification technique (HIT, Hierarchic Interaction Graph Templates) for the specification of user interfaces in its logical structures, user interfaces in a particular layout style and layout guidelines. The HIT specification technique is based on two well–known software construction methods: Dynamic Attribute Grammars (DAG) and Dataflow Diagrams (DFD).
- The HIT specification technique allows to create very modular specifications: The specification of the logical user interface can be composed of reusable building blocks representing single tasks or groups of related tasks ("views") of the task model. Moreover, these building blocks can be stored in libraries for reuse in different projects.
- BOSS offers an integrated, graphical Development Environment (IDE) for working out HIT–specifications in a visual-programming-like manner. HIT–specifications can be transformed into efficient C++ – programs using standard techniques from compiler generation.

In the following we give a brief introduction into the HIT specification technique (section 5.2.1). In sections 5.2.2 and 5.2.3 we show how HIT is used for modeling logical user interfaces and layout guidelines.

5.2.1 The HIT specification technique: An Overview

The HIT specification technique extends a well–known technique in compiler construction, Dynamic Attribute Grammars (DAG) [9], by timing– and event– concepts. A HIT specification consists of a set of basic data type and function definitions and a set of templates called HITs (Hierarchic Interaction graph Templates). HITs serve as prototypes for creating objects (HIT–instances) maintaining their own state, reacting in response to external messages and being connected with other objects in an object structure. A definition of a HIT consists of a structural (syntactic) and a semantic part. The structural definition describes how a HIT h is constructed from "simpler" HITs $h_1, ..., h_n$ using operators like construction of tuples (i.e. "parallel" composition, $h = (h_1, ..., h_n)$) or alternatives $h = h_1 \mid ... \mid h_n$. As in attribute grammars the structural description is enriched by semantic information. Associated with a HIT are various kinds of data flow constraints between the following entities:

- slots (in the context of attribute grammars named attributes) storing the state of a HIT instance. Certain slots of a HIT are distinguished: Through its argument–, argument/result– and result– parameter slots a HIT instance shares part of its state with related HIT instances in an object structure. Input slots may be modified by an external entity (e.g. a human user), the values of output slots are relevant to the environment (and have to be visualized by the user interface).

- message ports for receiving events from external entities and for the distribution of messages across a structure of HIT–instances. Like slots message ports may serve as parameters of a HIT or may be used for receiving (input message ports) and sending (output message ports) messages to external entities.

- rules defining either a directed equation in a "spreadsheet–like" manner (i.e. one–way constraints which should hold at every time) or a transaction caused by an external entity (e.g. an application function called by a user).

Input slots, input message ports and transactions rules may have preconditions. Each alternative h_i of an alternative HIT $h = h_1 \mid ... \mid h_n$ is assigned an applicability condition depending on the argument parameter slots of h_i. Creating an instance of an alternative HIT $h = h_1 \mid ... \mid h_n$ with argument parameter values $a_1, ..., a_{n_h}$ results in creating an instance of one of those alternatives with satisfied applicability condition. When a tuple HIT $h = (h_1, ..., h_n)$ is instantiated, instances for each component HIT h_i are created.

5.2.2 Specification of the Logical ISDN UI

Designing the logical UI consists of designing views which contain user interactions, system interactions and problem domain objects for a single task or a group of logically related tasks. In our example we follow the goal of designing a logical UI which is similar to the real ISDN phone. Therefore we introduce four views. By the view $BasicFunctions$ users gain access to the basic functionality of the ISDN phone for starting and terminating phone calls on line 1 (i.e. interface functions $start\_1st\_connection$, $terminate\_1st\_connection$, see figure 8). The view $AdvancedFunctions$ provides access to the advanced functions of the ISDN phone dealing with inquiries and conferences (i.e. application functions $start\_inquiry, ..., switch\_connections$, see figure 8). The view

$DialPhoneNumberTask$ corresponds to the task $DialPhoneNumber$ (see figure 9) allowing users to dial phone numbers directly digit by digit or to select phone numbers from the phone book. Finally the view $LogicalISDNUI$ describes the entire logical ISDN UI, i.e. $LogicalISDNUI$ contains the views $BasicFunctions$, $AdvancedFunctions$ and $DialPhoneNumberTask$. Moreover the view $LogicalISDNUI$ should present the state of the ISDN phone and allow users to add and remove entries from the phone book.

The views of the logical ISDN UI can be easily represented in the HIT specification technique. Figure 11 shows how the logical ISDN UI is represented as a tuple–HIT

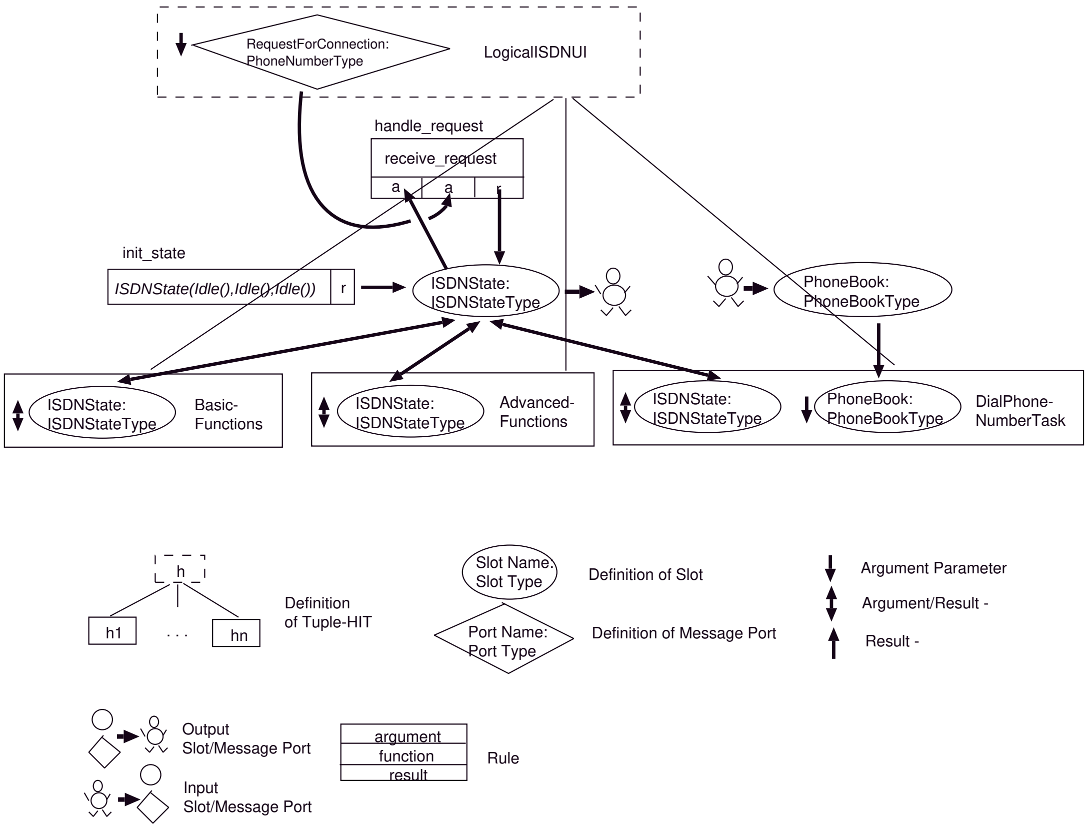

Figure 11: View $LogicalISDNUI$ of logical ISDN UI

$LogicalISDNUI$. The component HITs $BasicFunctions$, $AdvancedFunctions$ and $DialPhoneNumberTask$ represent the views in the logical ISDN UI described above. Through its argument message port $RequestForConnection$ the HIT $LogicalISDNUI$ receives phone calls on the external line. The slot $ISDNState$ stores the current state of the ISDN phone. As the user should be permanently informed about the state of the phone, $ISDNState$ is declared as an output slot. The slot $PhoneBook$ stores the phone book. As the user should be able to add and remove entries from the phone book, $PhoneBook$ is declared as an input slot. To indicate that the state $ISDNState$ can be alterated by user interactions in the views $BasicFunctions$, $AdvancedFunctions$

and *DialPhoneNumberTask*, the slot *ISDNState* is connected to the corresponding argument/result–parameter slots of the component HITs. The slot *ISDNState* is initialized by the rule *init_state* to the initial state *ISDNState*(*Idle*(), *Idle*(), *Idle*()) of the phone. A message in the argument message port *RequestForConnection* triggers the rule *handle_request*, which causes an update on the state of the ISDN phone (value of the slot *ISDNState*) according to the semantics of the *receive_request* function.

Figure 12 shows the HIT–representation of the view *BasicFunctions*. It groups user

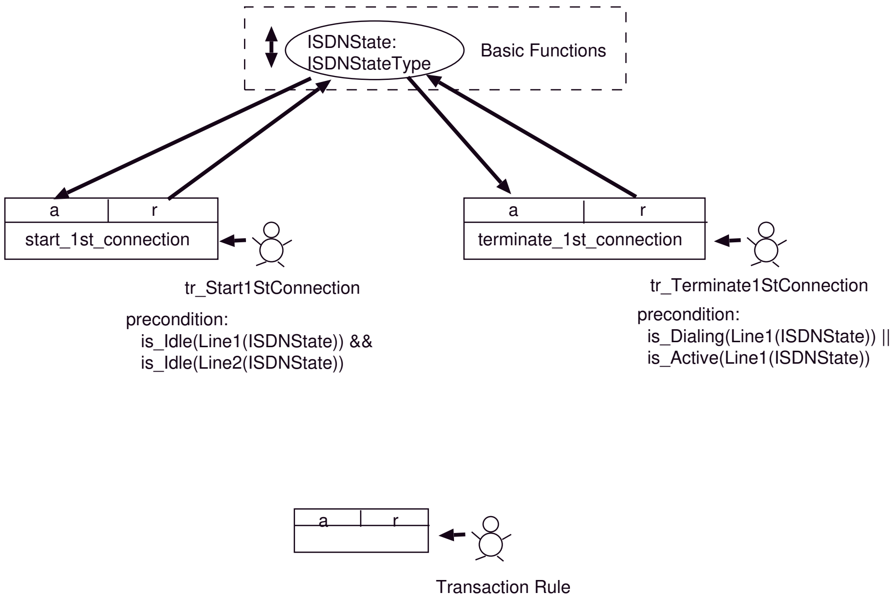

Figure 12: View *BasicFunctions* of logical ISDN UI

interactions related to start and terminate connections on line 1. As these interactions alter the state of the phone, the HIT *BasicFunctions* has an argument/result–parameter slot *ISDNState*. Through the transaction rule *tr_Start1stConnection* the user starts a phone call on line 1 by applying the interface function *start_1st_connection* (see figure 8) to the current state *ISDNState* of the phone. As the UI should prevent users from calling applications functions with parameters violating the function's precondition, the transaction rule *tr_Start1stConnection* is guarded by the precondition of the function *start_1st_connection*, which is taken directly from the algebraic specification of the problem domain model. By triggering the transaction rule *tr_Terminate1stConnection* the user can terminate a connection on line 1.

In Figure 13 we show how the view *DialPhoneNumberTask* (which corresponds to the task *DialPhoneNumber* in the task model, see figure 9) is represented by a corresponding HIT. Like the HITs *BasicFunctions* and *AdvancedFunctions* *DialPhoneNumberTask* has an argument/result–parameter slot *ISDNState* to indicate that the state of the ISDN phone is accessed and altered by user interactions. Through the argument parameter slot *PhoneBook* the phone book is passed. As the user

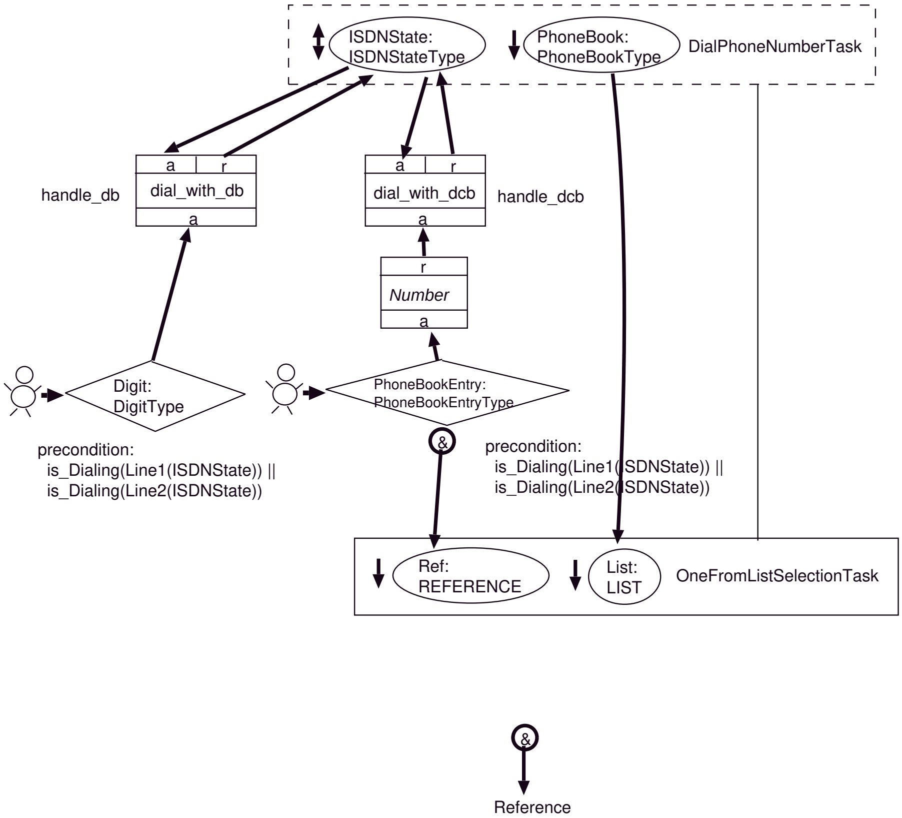

Figure 13: View *DialPhoneNumberTask* of logical ISDN UI

is allowed to enter the phone number digit by digit, the HIT *DialPhoneNumberTask* contains an input message port *Digit*. A message (i.e. a digit of the phone number) in the *Digit* message port triggers the rule *handle_db*, which alters the state of the ISDN phone according to the semantics of the *dial_with_db* function. To allow users to select a phone number directly from the phone book, we introduce an input message port *PhoneBookEntry*. A message (i.e. a selected entry of the phone book) in the *PhoneBookEntry* message port triggers the rule *handle_dcb*, which updates the state of the phone according to the semantics of the *dial_with_dcb* interface function. The preconditions of *Digit* and *PhoneBookEntry* ensure that user interactions with these message ports are enabled only in appropriate states of the phone. The selection from the phone book is modeled through a HIT *OneFromListSelectionTask*, whose argument parameter slots are supplied with a reference to the *PhoneBookEntry* message port (the selection should cause a message in *PhoneBookEntry*) and with the value of the *PhoneBook* slot (the list from which the item should be selected). As the user interaction "selection from a list" appears in many logical user interfaces, the BOSS–System provides a standard library containing HITs like *OneFromListSelectionTask* for the representation of standard interaction tasks.

The BOSS–System provides a very comfortable, integrated development enviroment

(IDE) which allows to draw specifications in the graphical notation shown in figures 11–13. Assuming a given set of layout guidelines, all the steps from the specification of the logical user interface to a "running" UI implementation are performed automatically by the BOSS–System.

5.2.3 Specification of Layout Guidelines

Layout guidelines describe the mapping from logical UIs to interfaces in particular layout styles by defining the representation of the states and state transitions of the logical UI in terms of interaction objects and events of an abstract or concrete UI–toolkit.

In BOSS the HIT specification technique is used both for the representation of the logical UI and the UI in a particular layout style. Consequently, as shown in figure 14, layout guidelines in BOSS model the transformation from the HIT–specification of a

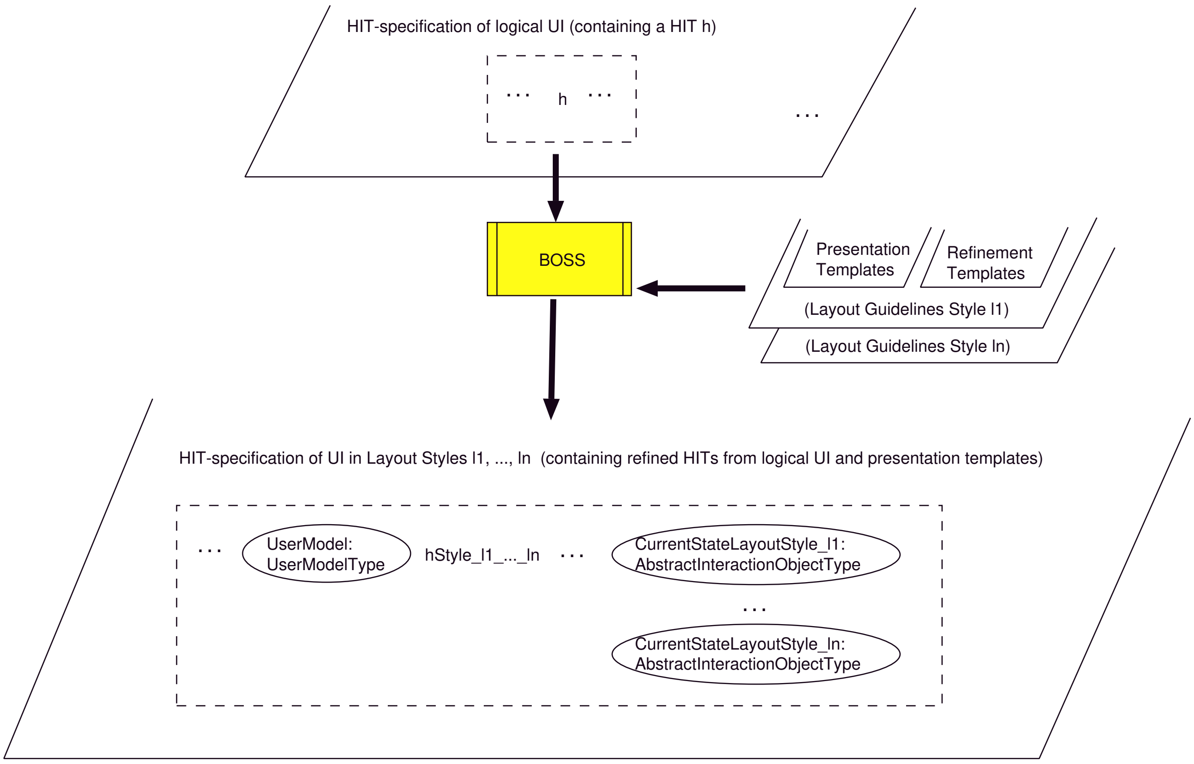

Figure 14: Layout Guidelines in the BOSS–System

logical UI into the HIT–specification of a UI in the layout styles described by the guidelines. Given layout guidelines for the styles $l_1, ..., l_n$ a HIT h in the specification of the logical UI is refined into a HIT $hStyle\_l1\_...\_ln$. $hStyle\_l1\_...\_ln$ contains an additional argument parameter slot $UserModel$ and additional result parameter slots $CurrentStateLayoutStyle\_l1, ..., CurrentStateLayoutStyle\_ln$. The result parameter slot $CurrentStateLayoutStyle\_li$ contains the layout of the current UI state (represented in terms of abstract interactions objects) for the layout style l_i depending on the properties of the user model passed in the argument parameter slot $UserModel$. This architecture results in a high flexibility of the generated UI: As $hStyle\_l1..ln$ contains layout information for each style, it's possible to switch the layout style at runtime.

The specifications of layout guidelines for a style l_i consists of a set of presentation templates and a set of refinement templates. For each element in the HIT specification language dealing with user interaction (input–,output slots, input–, output message ports, transaction rules) a specialized presentation template is defined (e.g. a template $PresentOutputSlotStateStyle\_li$ for the presentation of the value of output slots). The refinement templates describe the refinement from HITs without layout information (e.g. h) to HITs with layout information (e.g. $hStyle\_l1\_...\_ln$). This refinement is done by attaching the appropriate presentation templates to the interactive parts of a HIT. E.g. in the HIT $LogicalISDNUI$, which describes the main view on the logical ISDN UI, a $PresentOutputSlotStateStyle\_li$ presentation template is attached to the output slot $ISDNState$. In the BOSS–System, presentation– and refinement templates are defined in the HIT specification technique itsself. E.g. the presentation template $PresentOutputSlotStateStyle\_li$ is defined as a HIT with argument parameter slots $UserModel$ (the user model) and $OutputSlotState$ (i.e. the value of the output slot) and a result parameter slot $OutputSlotStateLayout$, which delivers a layout in terms of abstract interaction objects. The HIT specification technique is well–suited for representing such presentation templates, as the typical decision–tree–like structure can be expressed easily through nested alternative HITs.

As the HIT specification technique allows modular specifications, the specifications of guidelines for different layout styles differ only in a few presentation and refinement templates. Furthermore, it is possible to combine general–purpose guidelines with guidelines for a specific problem domain. For the generation of the ISDN interfaces shown in figures 3 and 2 we combined a general–purpose styleguide with a specialized ISDN styleguide containing a few specialized presentation templates e.g. for presenting objects of the sort $ConnectionStateType$ as "smilies".

6 Conclusion and further research

The BOSS–System has been implemented in C++ on top of UNIX/X11R6. It currently supports the Athena and the OSF/Motif toolkits. The animation component of PLUG–IN is based on Tcl/Tk. The context–sensitive help–component of PLUG–IN is based on the WWW browser Mosaic. The FLUID–System, whose theoretical foundations are presented in [3], is currently under development. The FUSE methodology and tools have been applied successfully to a number of examples (ISDN phone simulation, user interface for a literature retrieval system, user interface for a home banking system, formula editor for LaTeX). Important parts of the FUSE development environment (e.g. the subsystem FIRE, see [23]) have been specified with BOSS. In the future we plan to increase the level of compatibility of the FUSE development environment to other model based methodologies and tools. E.g. for setting up the problem domain model, we want to support OOA and ERA data models in addition to the currently supported algebraic specification technique. In order to gain more practical experience with the FUSE–methodology and the related tools, we plan to organize a course in user interface specification at the Munich University of Technology.

7 Acknowledgements

This work has been partially supported by Siemens Corporate Research and Development, Department of System Ergonomics and Interaction (ZFE ST SN 51). The authors would like to thank Werner Schreiber for his useful comments and suggestions on draft versions of this report.

References

[1] Siemens AG. Telefon Bedienungsanleitung Hicom Standard 300, 1992.

[2] H. Balzert. From OOA To GUI – The Janus System . In *Proceedings Interact 95* . IFIP , 7 1995.

[3] B. Bauer. Generating User Interfaces from Formal Specifications of the Application. Submitted to DSV-IS'96 Workshop, Namur, Belgium , 1996.

[4] F. Bodart, A.M. Hennebert, J.M. Leheureux, I. Provot, and J. Vanderdonckt. A Model–Based Approach to Presentation: A Continuum from Task Analysis to Prototype. In F. Paterno, editor, *Proceedings Eurographics Workshop Design,Specification, Verification of Interactive Systems, Carrara, Italy Juni 8-10, 1994* . Springer Focus on Computer Graphics Series , 1995.

[5] J. Eickel. Logical and layout structures of documents. *Computer Physics Communication*, 61:201–208, 1990.

[6] T. Elwert and E. Schlungbaum. Modelling and Generation of Graphical User Interfaces in the TADEUS Approach . In *DSV-IS'95 Workshop Proceedings* . Eurographics Association, 1995.

[7] T. Fehrle, K. Klöckner, V. Schölles, F. Berger, M. Thies, and W. Wahlster. PLUS - Plan-based User Support. Technical report, DFKI-Report RR-93-15, 1993.

[8] J.D. Foley. History, Resums and Bibliography of the User Interface Design Environment (UIDE), an Early Model-based System for User Interface Design and Implementation . In *DSV-IS'94 Workshop Proceedings* . Springer, 1995.

[9] H. Ganzinger. *Optimierende Erzeugung von Übersetzerteilen aus implementierungsorientierten Sprachbeschreibungen.* PhD thesis, Technische Universität München, 1978.

[10] S. Hudson and R. King. A generator of direct manipulation office systems. *ACM Transactions on Information Systems*, 4(2):132–163, 1986.

[11] C. Janssen, A. Weisbecker, and J. Ziegler. Generating User Interfaces from Data Models and Dialogue Net Specifications . In *ACM Interchi 93 Proceedings* . ACM, 1993.

[12] P. Johnsen, P. Markopoulos, and H. Johnsen. Task Knowledge Structures: A specification of user task models and interaction dialogues . In *Proceedings of 11th Interdiciplinary workshop on informatics and psychology* , 6 1992.

[13] F. Lonczewski. PLUG–IN: Using Tcl/Tk for Plan Based User Guidance. In *Proceedings of the Tcl/Tk Workshop, July 6-8 1995, Toronto.* USENIX Association, ISBN 1-880446-72-3, 1995.

[14] F. Lonczewski. Using a WWW–Browser as an alternative user interface for interactive applications. In R. Holzapfel, editor, *Poster Proceedings of the 3rd World Wide Web Conference, Darmstadt, Germany.* Fraunhofer Institute for Computer Graphics , 1995.

[15] F. Lonczewski and S. Schreiber. The FUSE–System: An Integrated User Interface Design Environment. Submitted to DSV-IS'96 Workshop, Namur, Belgium , 1996.

[16] P. Luo, P. Szekely, and R. Neches. Management of Interface Design in HUMANOID . In *ACM Interchi 93 Proceedings* . ACM, 1993.

[17] R. Moriyon. Automatic Generation of Help from Interface Design Models. In *CHI' 94 Proceedings* . ACM , 1994.

[18] D. R. Olsen. MIKE: The Menu Interaction Kontrol Environment. *ACM Transactions on Graphics*, 5(4):318 – 344, 1986.

[19] D. R. Olsen. A programming language basis for user interface managment. In *ACM CHI 89 Proceedings.* ACM, 1989.

[20] S. Schreiber. Specification and Generation of User Interfaces with the BOSS-System . In A. Cypher and J. Gornostaev, editors, *Proceedings East-West International Conference on Human-Computer Interaction EWHCI'94* . ICSTI Moscow, 8 1994. Also in: Human Computer Interaction, Selected Papers EWHCI'94 Conference, Springer LNCS 876 .

[21] S. Schreiber. The BOSS–System: Coupling Visual Programming with Model–Based Interface Design . In F. Paterno, editor, *Proceedings Eurographics Workshop Design, Specification, Verification of Interactive Systems 1994, Carrara, Italy Juni 8-10, 1994* . Springer Focus on Computer Graphics Series, ISBN 3-540-59480-9, 1995.

[22] W. Schreiber. Prosaische Logik für Dichter und Denker – Textverarbeitung maßgeschneidert . *Forschung für Bayern*, (6), 1993.

[23] R. Schwab. Generierung von Standardbedienoberflächen aus Applikationsbeschreibungen . Master's thesis, Technische Universität München, 1995.

[24] C. Wiecha. Generating highly interactive user interfaces. In *CHI'89 Proceedings*, 1989.