

# The FUSE–System: an Integrated User Interface Design Environment

*Frank Lonczewski and Siegfried Schreiber*

Institut für Informatik, Technische Universität München, Arcisstraße 21,  
D-80290 München, Germany

Phone: +49-89-289-22035 – Fax: +49-89-289-28180

E-mail: {lonczews, schreibs} @informatik.tu-muenchen.de

WWW: [http://www2.informatik.tu-muenchen.de/persons/  
{lonczews/fralo.html, schreibs/schreibs.html}](http://www2.informatik.tu-muenchen.de/persons/{lonczews/fralo.html, schreibs/schreibs.html})

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 modelling 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 generates task–based help– and user guidance components.

## **Keywords**

Automatic generation of user interfaces, model-based interface design, specification of styleguides, user guidance, user interface design, generated on-line help.

## **Introduction**

Even with the most advanced layout oriented UI construction tools (UI toolkits, UI builders, UIMS) the task–based and user–oriented development of GUIs remains a time–consuming and difficult process. Therefore tools for the formal specification and automatic generation of UIs (MB-IDEs) 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 UI 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 of the UI development process.
- Generation of working prototypes in early phases of the development process.
- Standardization of UIs by formal specification of UI styleguides.
- Generation of powerful help– and user guidance components.

The paper is organized as follows: in section 1 we describe the overall methodology and architecture of the FUSE–System ; in section 2 we demonstrate the capabilities of FUSE for the generation of UIs in different layout styles and of help– and user guidance components by using the example of an ISDN phone simulation. Section 3 describes the stages in the development process of the ISDN UI using the FUSE–System. In section 4 we discuss related work in the field of model-based UI construction. Finally we give an overview about practical experience with FUSE and directions of further research.

## 1 The FUSE-Methodology: an Overview

The overall architecture of the FUSE–System is shown in figure 1. FUSE consists of the four components BOSS (BedienOberflächen-SpezifikationsSystem, the german translation of “user interface specification system” [Schreiber94a, Schreiber94b]), FLUID (Formal User Interface Development, [Bauer96]), PLUG–IN (Plan–based User Guidance for Intelligent Navigation [Lonczewski95a, Lonczewski95b]) and FIRE (Formal Interface Requirements Engineering [Schwab95]). Each of these tools may also be used independently of the FUSE–System.

The UI development process with FUSE consists of the phases requirements analysis, design and evaluation. For the UI part of an interactive application no implementation phase is needed, as FUSE generates running UIs from design–level specifications. Some activities in the development process have to be carried out only once for a whole class of UIs. 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 UI 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).

In the analysis phase of the development process, an application analyst defines the requirements for the UI by setting up the formal specification of three models. The specification of the task model describes the task hierarchy of the 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.

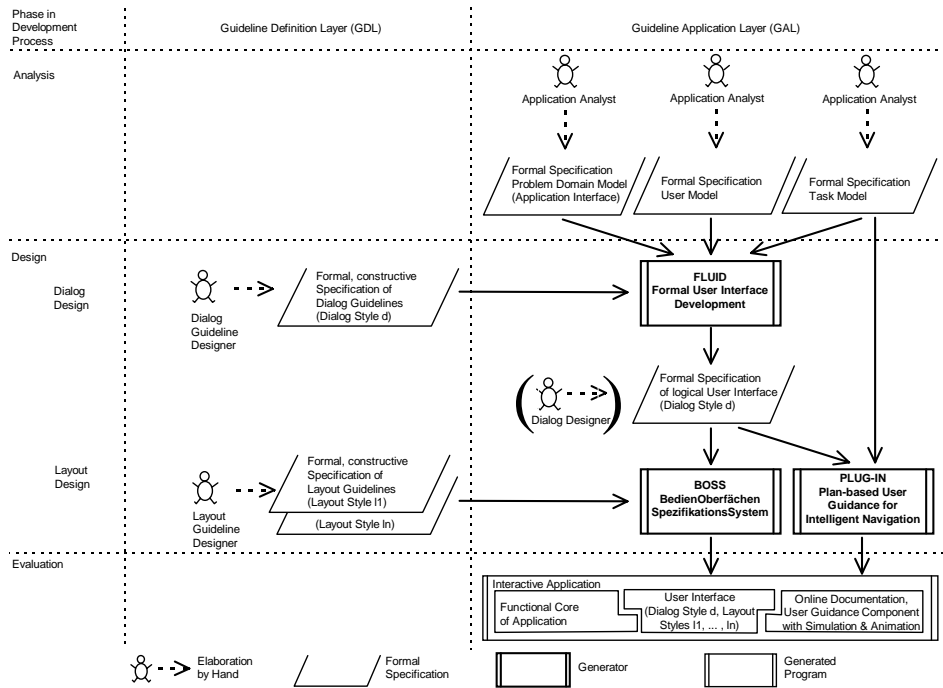


Figure 1. Overall Architecture of the FUSE-System

The user model is a description of static and dynamic properties of user groups and individual users which influences both the UI generation process and the kind and depth of the help offered by the user guidance component. 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 three models and a tool for the generation of an early UI prototype. This prototype represents the task- and problem domain model in terms of menus and dialogue-boxes and provides a good basis for discussing the results of the requirements analysis with end-users.

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 so called logical UIs in a particular dialogue style. At the abstraction level of logical UIs, static and dynamic properties of UIs are described without considering presentation issues. Layout guidelines describe the transformation from specifications of logical UIs into specifications of UIs in a particular layout style. The formal specification of dialogue- and layout guidelines has to be carried out once for a whole class of UIs (i.e., belongs to the GDL-layer in the development process) and can be regarded as the specification of an UI styleguide.

Within FUSE, 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 the task-, problem domain- and user model of an interactive application, FLUID generates the specification of static and dynamic properties of a logical UI in this dialogue style  $d$ .

This specification may be modified by a human dialogue designer. For the representation of the design of the logical UI, FUSE employs a specification technique called Hierarchic Interaction graph Templates (HIT), which is based on attribute grammars and dataflow diagrams.

Besides the automatic generation with FLUID, which requires a formal specification of the models in the requirements analysis, FUSE also offers support for designers not skilled in formal techniques like algebraic specification. Based on an informal description of the task-, problem domain- and user model a human dialogue designer can elaborate the HIT specification of the logical UI by hand.

From the specifications of layout guidelines for the layout styles  $l_1, \dots, l_n$  and the specification of a logical UI (generated automatically by FLUID or specified by a human dialogue designer) in a dialogue style  $d$  the BOSS-System generates an implementation of a UI in the dialogue style  $d$ , in which the end-user can switch at run-time between the layout styles  $l_1, \dots, l_n$ .

For the formal specification of the layout guidelines BOSS also uses the HIT specification technique. The separation between logical UIs and UIs in particular layout styles in the design phase (see figure 1), which is typical for model based UI tools, can be also found in related research domains like document architecture (see e.g., [Eickel90, Schreiber93]).

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

## 2 User Support for an ISDN Phone with FUSE

In this section we present examples of different UI for an ISDN phone simulation generated with the BOSS system. The ISDN phone simulation is an interactive application for the simulation of the real ISDN phone described in [Siemens92]. Furthermore we look into the problems that the user can possibly have when using one or more of these UIs. We also show the various kinds of help that PLUG-IN provides for the ISDN phone simulation.

## 2.1 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`.

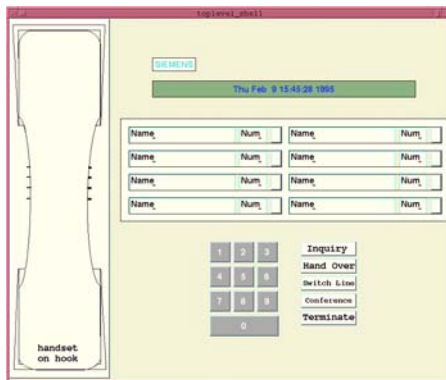


Figure 2. Button Interface

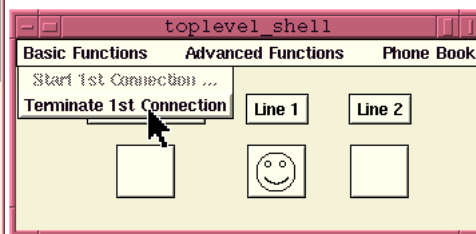


Figure 3. Menu Interface

In figures 2 and 3 we can see two different UIs of the ISDN phone simulation. The left figure displays the button interface. The elements of this UI 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. The right figure displays a functional equivalent menu interface for the phone simulation. Both UIs are generated with the BOSS-System. The user can change the layout between the two styles presented above during runtime.

The alternative UI of the ISDN phone simulation in figure 3 consists of a menu-pane 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 button 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 tasks of the ISDN phone simulation is **StartConference**. In an ISDN phone conference the three participating parties can talk and hear each other simultaneously. Despite the fact that a **Conference** button is available on the button UI (and similarly a **StartConference** menu entry in the menu **AdvancedFunctions** of the menu UI), 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 [Siemens92], it is not as easy as just pressing the conference button on the UI. 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 with dynamic on-line help and task-based user guidance. For this purpose all user interactions are observed.

## **2.2 Task-Based User Support with PLUG-IN**

In order to support 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, the system searches for a method to solve the identified task.

A task can be accomplished if its task-goal can be reached. Typically a unique (sub)state of the UI 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 creates 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 3.1.

A list of ISDN phone tasks is shown in figure 4. If the user selects a task from the list while working with the phone simulation, the necessary interaction sequences are simulated on the UI.

### 2.3 Dynamic On-Line Help with PLUG-IN

The dynamic 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 user-model and can be represented as a set of finite state automations. The information contained in these can be used to:

- generate help pages (see below);
- visualize the set of finite state automations 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.

As an example a STD for the ISDN phone simulation 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 dynamic on-line help pages and animation sequences. The dynamic on-line help pages can be inspected with a WWW browser like Netscape or NCSA Mosaic.

One dynamic on-line help page is displayed in figure 6. Each dynamic on-line help page is typically divided into four regions:

- information about the current state of the application from the user's point of view;
- 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 hyper-text version of the user manual of the application.

Since all operations the user can perform on the original UI can also be triggered through the WWW browser, it can be regarded as an alternative UI. In contrast to the original UI, the goal of the WWW-based UI 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 UI through simulations provided by PLUG-IN.

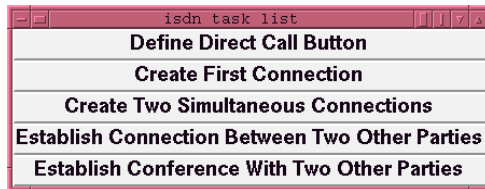


Figure 4. Task List for ISDN Phone Simulation

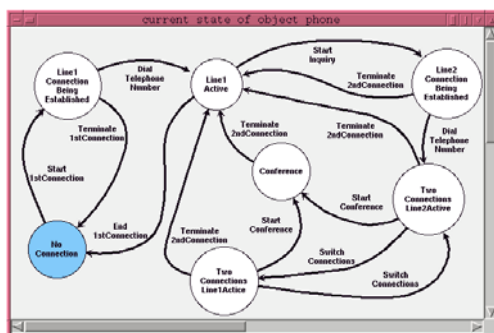


Figure 5. STD for ISDN Phone Simulation

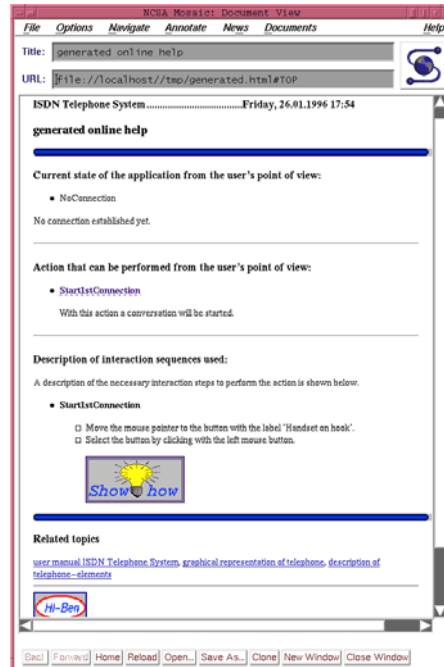


Figure 6. One Dynamic On-Line Help Page generated by PLUG-IN

Depending on the current state of the application (in figure 5 this is the highlighted state *NoConnection* of the STD) and the chosen layout style for the UI, 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 UI shown in figure 2 is displayed in figure 6. If the user selects the light bulb icon on the page, the described user interactions are animated on the UI. In this example PLUG-IN would take control of the mouse pointer, then change 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 UI 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 tries to find a path to the selected node. If a path can be found, the corresponding user interactions are animated on the displayed UI.

PLUG-IN has the capability to deal with different UI layouts with regard to the generated on-line help pages and animation sequences. If the layout style of the UI changes during runtime, the description of the necessary interaction steps on the dynamic 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 with PLUG-IN is very flexible, because the help offered adapts itself automatically to the runtime context.

It is worth mentioning 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 [Bauer96] will provide the automatic generation of these STDs by using the information from the task-, problem domain- and user model.

### 3 Modelling the ISDN User Interface with Fuse

In the following we describe the systematic development of the ISDN UI described in section 2. In section 3.1 we demonstrate how the task- and problem domain models are represented during the requirements analysis. In section 3.2 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 [Bauer96].

#### 3.1 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  $\text{Spec}_{AI} = \langle \Sigma_{AI}, \text{Ax}_{AI} \rangle$ . The signature part  $\Sigma_{AI}$  consists of the definitions of sorts (data types) with associated constructor- and selector functions for the description of the conceptual problem-domain objects. Furthermore  $\Sigma_{AI}$  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 domain 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 `Phone-`

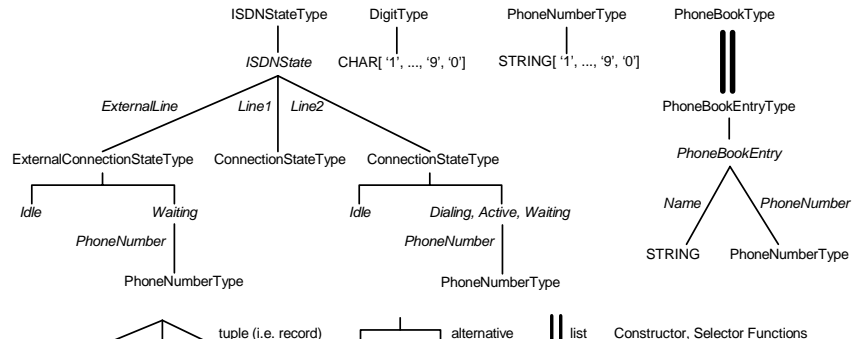


Figure 7. Sorts, Constructor- and Selector Functions for the ISDN Phone

NumberType defines phone numbers as strings out of the ordered character set  $\{ '1', \dots, '9', '0' \}$ . 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.

```

// ... 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

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  $is\_idle(Line1(sb)) \wedge is\_idle(Line2(sb))$ , i.e. the phone is not in use. After

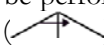

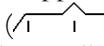
calling `start_1st_connection`, line 1 of the phone should be in the state `Active`, if there was a phone call request on the external line. If there was no such request, line 1 should be in the state `Dialing`. This behaviour is expressed by the axioms

$\forall n \in \text{PhoneNumberType}$ :

`start_1st_connection(ISDNState(Waiting(n),Idle(),Idle())) = ISDNState(Idle(),Active(n),Idle())`  
`start_1st_connection(ISDNState(Idle(),Idle(),Idle())) = ISDNState(Idle(),Dialing(<->),Idle())`

In a similar way, the semantics of the other interface functions are described. The information in the algebraic specification  $\text{Spec}_{AI}$  of the ISDN problem domain can be used for different purposes in the UI development process. Using techniques from theorem proving, the STD for PLUG-IN (e.g., figure 5) can be generated automatically (see [Bauer96]). Moreover, the information in  $\text{Spec}_{AI}$  is used for the specification of the dynamics of the logical UI (see section 3.2.2).

While the problem domain model defines the requirements for the UI 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 problem domain model. An example is shown in figure 9.

The task `EstablishConference` can be decomposed into the subtasks `Create1st-Connection`, `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 problem domain model are denoted by the symbol . Besides the **sequence**, other constructs define temporal relations in the task space. Each nodes pre- and postcondition referring 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.

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 modelled. Examples of 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. Besides the static properties, we also plan to incorporate dynamic properties into the user model. Dynamic properties are obtained during runtime. With these it will be possible to give help that is adapted to the user’s individual interaction behaviour. One example of a dynamic property is the “frequency of already solved tasks”. With this property it is possible

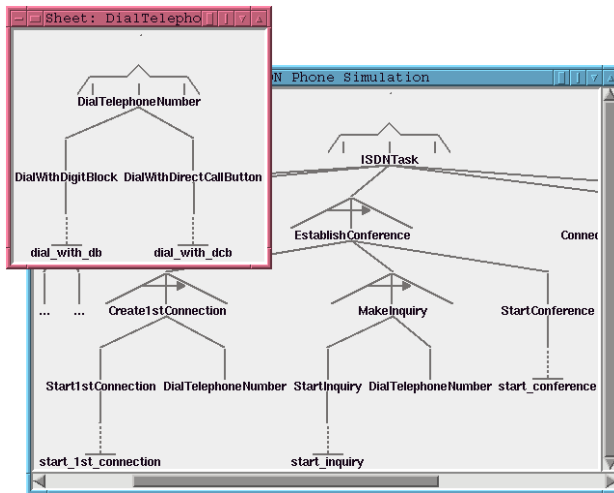


Figure 9. Excerpts from Task Space of Phone Simulation

Figure 10. Property Sheet of Example Task

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.

### 3.2 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 phase, BOSS is used by an automatic (i.e., the FLUID-System, see [Bauer96]) or a human (the scenario we assume in this paper) dialogue designer for the specification of the logical structures of UIs 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 the logical structures of UIs, UIs 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 Data Flow Diagrams (DFD).
- The HIT specification technique allows the creation of very modular specifications: The specification of the logical UI 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. Because of this modularity, this HIT specification technique scales up for modelling complex UIs.

- BOSS offers an Integrated graphical Development Environment (IDE) for working out HIT-specifications in a visual-programming-like manner (a textual specification is also possible). HIT-specifications can be transformed into efficient C++ programs using standard techniques from compiler generation.

In the following we give a brief introduction to the HIT specification technique (section 3.2.1). In sections 3.2.2 and 3.2.3 we show how HIT is used for modelling logical UIs and layout guidelines.

### 3.2.1 The HIT Specification Technique: an Overview

The HIT specification technique extends a well-known technique in compiler construction, DAGs [Ganzinger78], with timing- and event-concepts. A HIT specification consists of a set of basic sort (data type) and function definitions and a set of templates called Hierarchic Interaction graph Templates (HITs). HITs serve as prototypes for creating objects (HIT-instances) that maintain their own state, react in response to external messages and are 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  $h_1, \dots, h_n$  using operators like construction of tuples (i.e., “parallel” composition,  $h = (h_1, \dots, h_n)$ ) or alternatives,  $h = h_1 \mid \dots \mid h_n$ . 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 (known as attributes in the context of attribute grammars) store 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 UI);
- message ports receive events from external entities and distribute 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 define 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  $h = h_1 \mid \dots \mid h_n$  is assigned an applicability condition depending on the argument parameter slots of  $h_i$ . Creating an instance of an alternative HIT  $h = h_1 \mid \dots \mid h_n$  with argument parameter values  $a_1, \dots, a_{nh}$  results in creating an instance of one of those alternatives with satisfied applicability condition. When a tuple HIT  $h = (h_1, \dots, h_n)$  is instantiated, instances for each component HIT  $h_i$  are created.

### 3.2.2 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. With the **BasicFunctions** view users gain access to the basic functionality of the ISDN phone for starting and terminating phone calls on line 1 (i.e., functions `start_1st_connection`, `terminate_1st_connection`, see figure 8). The **AdvancedFunctions** view provides access to the advanced functions of the ISDN phone dealing with inquiries and conferences (i.e., functions `start_inquiry`, ..., `switch_connections`, see figure 8). The **DialPhoneNumberTask** view 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 **LogicalISDNUI** view describes the entire logical ISDN UI, i.e., **LogicalISDNUI** contains the **BasicFunctions**, **AdvancedFunctions** and **DialPhoneNumberTask** views. Moreover the **LogicalISDNUI** view should illustrate 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 named **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.

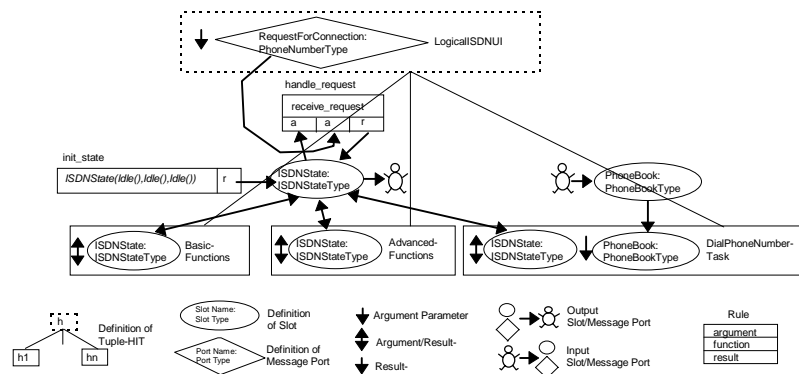


Figure 11. LogicalISDNUI view of logical ISDN UI

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 altered by user interactions in the **BasicFunctions**, **AdvancedFunctions** and **DialPhoneNumberTask** views, the slot **ISDN-**

State 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.

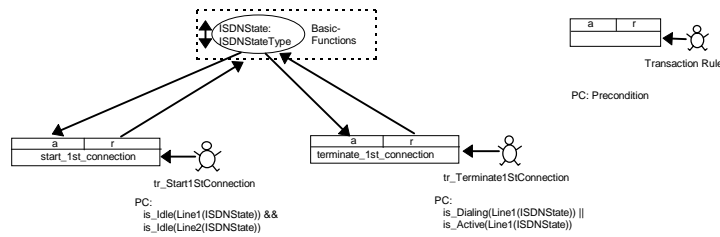


Figure 12. BasicFunctions view of logical ISDN UI

Figure 12 shows the HIT-representation of the BasicFunctions view. It groups user 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 function `start_1st_connection` (see figure 8) to the current state `ISDNState` of the phone. As the UI should prevent users from calling functions with parameters violating the function's precondition, the transaction rule `tr_Start1stConnection` is guarded by the precondition of the function `start_1st_connection` taken directly from the algebraic specification  $\text{Spec}_{AI}$  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 `DialPhoneNumberTask` view (which corresponds to the `DialPhoneNumber` task 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 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 en-

abled only in appropriate states of the phone. The selection from the phone book is modelled 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 UI, the BOSS–System provides a standard library containing HITs like `OneFromListSelectionTask` for the representation of standard interaction tasks.

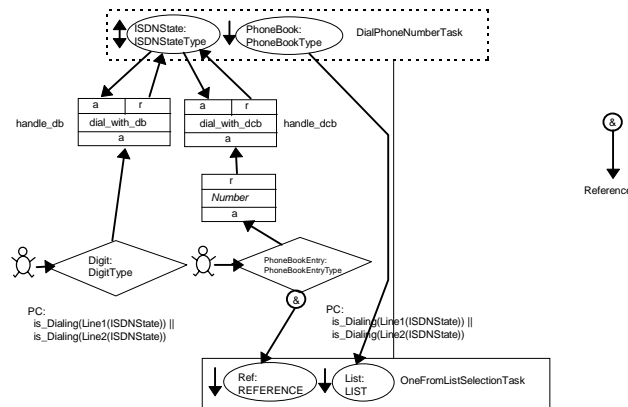


Figure 13. DialPhoneNumberTask view of logical ISDN UI

The BOSS–System provides a very comfortable, integrated development environment (IDE) which allows drawing specifications in the graphical notation shown in figures 11–13. Like the specification of the logical ISDN UI, the specification of more complex logical UI is made up of small, reusable building blocks. In this way the HIT specification technique scales up for more complex examples.

### 3.2.3 Specification of Layout Guidelines

Assuming a given set of layout guidelines, all the steps from a given specification of the logical UI (e.g., figures 11–13) to the production of a “running” UI implementation (e.g., figures 2-3) are performed automatically by the BOSS–System. Similar to the approach followed in the HUMANOID–System [Luo93], layout guidelines within BOSS describe the mapping from logical UIs to UIs in particular layout styles by defining the representation of the states and state transitions of the logical UI in terms of AIOs. As the actual UI layout is computed at runtime, it is possible to specify context sensitive UI layouts depending on values known only at runtime. Due to this approach, systems like BOSS or HUMANOID reach a higher degree of flexibility than systems generating a static UI layout at design time.

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 logical UI into the HIT-specification of a UI in the layout styles described by the guidelines. Given layout guidelines for the styles  $l_1, \dots, l_n$  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\_11**, ... , **CurrentStateLayoutStyle\_ln**. The result parameter slot **CurrentStateLayoutStyle\_li** contains the layout of the current UI state represented in terms of AIOs 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 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.

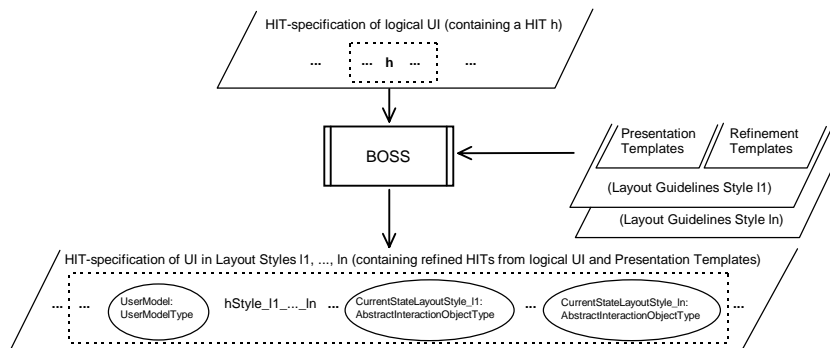


Figure 14. Layout Guidelines in the BOSS-System

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 itself. 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 AIOs. The HIT specification technique is well-suited for representing such presentation tem-

plates, as the typical decision–tree–like structure (see e.g., [Bodart93]) 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 2 and 3 we combined a general–purpose styleguide with a ISDN styleguide containing a few specialized presentation templates e.g., for presenting objects of the sort `ConnectionStateType` as “smilies”.

## 4 Related Work

In the following we give a brief overview of related work in the field of model based UI construction. At the end of this section we summarize the main differences between these approaches and our FUSE–System.

MIKE [Olsen86], MIKEY [Olsen89], HIGGENS [Hudson86] and JANUS [Balzert95a] are examples of tools, which generate UI based alone on a specification of the problem domain model. JANUS differs from the first tools in using a much more powerful technique for data modelling (OOA class diagrams). As none of these tools provides means for the explicit specification of UI dynamics, they are tailored to applications, where the UI dynamics can be derived from data models (i.e. data–base oriented applications).

The ITS–System [Wiecha89] offers a frame–based language for the specification of logical UI (“dialogue content”). Moreover, ITS allows the specification of style rules, which describe the mapping between logical UI and UI in a particular style.

In the UIDE–System [Foley94], the UI development process consists of the description of two models. In the application model, the logical UI is described in terms of application objects and –tasks. The UI–model describes the coupling of the application model to a UI 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 [Luo93] divides the UI development process into the activities of application design, dialogue sequencing, action side effects, presentation design and manipulation design. In the first three design dimensions the logical structure of a UI 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 a logical UI 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 (see [Moriyón94]). Recently the research on UIDE and HUMANOID was joined in the MASTERMIND project.

In the ADEPT-System [Johnson92b], a process-algebra-like specification technique called Task Knowledge Structures (TKSs) 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 Abstract Interface Model (AIM), 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 AIOs in the AIM by the appropriate CIOs in the CIM.

The GENIUS-System [Janssen93] generates UIs for data-base oriented applications. In GENIUS, the problem domain model is represented by an ERA-diagram. Based on this ERA-diagram static aspects of the logical UI are described in terms of so called views, which can be regarded as abstract representations of UI windows. For the representation of the dynamics of the logical UI, GENIUS employs a petri-net-like specification technique (“dialogue-nets”).

For each view in the logical UI, 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). A similar approach is presented in the TADEUS-System [Elwert95]. TADEUS differs from GENIUS in the use of different specification techniques for the representation of the problem domain model (TADEUS uses an object oriented approach) and the dynamics of the UI (dialogue-graphs).

The TRIDENT-System [Bodart94b] 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 called Activity Chaining Graphs (ACGs). Each ACG is structured into presentation units. From these presentation units, the static UI layout can be generated by applying rules for the selection of AIOs, rules for mapping AIOs to CIOs and rules for the placement of CIOs.

PLUS [Fehrle93] 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 UI specific (e.g., UI layout) properties is combined.

The main difference between FUSE and the approaches presented above is the combination of the following properties: FUSE offers tool-based support across the whole UI development process, whereas e.g., MIKE, MIKEY, ITS, JANUS or ADEPT correspond to subsystems of FUSE (BOSS and FIRE). Like in HUMANOID the UI layout is computed at runtime, which results in a higher flexibility (e.g., the layout depends on values known only at runtime, the layout style can change at runtime) compared to systems generating the static UI layout at design time.

The flexibility is also supported by the powerful on-line help- and user guidance components generated by the PLUG-IN system. In contrast to the approach to on-

line help used in HUMANOID, PLUG-IN generates dynamic online help pages in HTML format that can be inspected with a WWW browser. In comparison with the PLUS-System, where it is necessary to change the database used for the provision of user guidance by hand if the application functionality or layout guidelines are changed, PLUG-IN's generated on-line help adapts itself automatically to the currently used layout style of the UI.

## Conclusion

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 (see [Bauer96]) is currently under development.

The FUSE methodology and tools have been applied successfully to a number of examples (ISDN phone simulation, UI for a literature retrieval system, UI for a home banking system, formula editor for L<sup>A</sup>T<sub>E</sub>X). Important parts of the FUSE development environment (e.g., the subsystem FIRE) have been specified with BOSS.

Up to now, the FUSE system and especially BOSS have been used by developers skilled in related methods from software construction (e.g., attribute grammars). With this background these developers were able to achieve quite soon a high level of productivity using our tools. However, we are aware of the fact that much more practical experience has to be gained with the FUSE-methodology and the related tools. As a first step in this direction we plan to organize a course in UI specification at the Munich University of Technology.

## 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 and the anonymous reviewers for their useful comments and suggestions on draft versions of this paper.

## References

- [Balzert95a] Balzert, H., *From OOA to GUI - The JANUS-System*, in Proceedings of the 5th IFIP TC13 Conference on Human-Computer Interaction INTERACT'95, Lillehammer, 25-29 June 1995, K. Nordbyn, P.H. Helmersen, D.J. Gilmore and S.A. Arnesen (Eds.), Chapman & Hall, London, 1995, pp. 319-324. <http://www.swt.ruhr-uni-bochum.de/forschung/janus/lillehammer.html>
- [Bauer96] Bauer, B., *Generating User Interfaces from Formal Specifications of the Application*, in Proceedings of 2<sup>nd</sup> International Workshop on Computer-Aided Design of

User Interfaces CADUP96, J. Vanderdonckt (Ed.), Presses Univesitaires de Namur, Namur, 1996, pp. 141-158.

[Bodart93] Bodart, F., Hennebert, A.-M., Leheureux, J.-M., Sacré, I., Vanderdonckt, J., *Architecture Elements for Highly-Interactive Business-Oriented Applications*, in Proceedings of the East-West International Conference on Human-Computer Interaction EWHCI'93 (Moscow, 1993), L. Bass, J. Gornostaev and C. Unger (Eds.), Lecture Notes in Computer Science, Vol. 753, Springer-Verlag, Berlin, 1993, pp. 83-104.

[Bodart94b] Bodart, F., Hennebert, A.-M., Leheureux, J.-M., Provot, I., Vanderdonckt, J., *A Model-based Approach to Presentation: A Continuum from Task Analysis to Prototype*, in Proceedings of 1<sup>st</sup> Eurographics Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'94 (Bocca di Magra, 8-10 June 1994), Focus on Computer Graphics Series, Springer-Verlag, Berlin, 1995, pp. 77-94.

[Eickel90] Eickel, J., *Logical and Layout Structures of Documents*, Computer Physics Communication, Vol. 61, 1990, pp. 201-208.

[Elwert95] Elwert, T., Schlungbaum, E., *Modelling and Generation of Graphical User Interfaces in the TADEUS Approach*, in Proceedings of 2<sup>nd</sup> Eurographics Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'95 (Château de Bonas, 7-9 June 1995), R. Bastide and Ph. Palanque (Eds.), Eurographics Series, Springer-Verlag, Vienna, 1995, pp. 193-208. <http://www.informatik.uni-rostock.de/~schlung/TADEUS/paper/DSV-IS95.html>

[Fehrle93] Fehrle, T., Klöckner, K., Schölles, V., Berger, F., Thies, M., Wahlster, W., *PLUS - Plan-based User Support*, Deutsches Forschungszentrum für künstliche Intelligenz, Technical report RR-93-15, 1993.

[Foley94] Foley, J.D., *History, Results and Bibliography of the User Interface Design Environment (UIDE), an Early Model-based Systems for User Interface Design and Implementation*, in Proceedings of 1<sup>st</sup> Eurographics Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'94 (Bocca di Magra, 8-10 June 1994), Focus on Computer Graphics Series, Springer-Verlag, Berlin, 1995, pp. 3-14.

[Ganzinger78] Ganzinger, H., *Optimierende Erzeugung von Uebersetzerteilen aus implementierungsorientierten Sprachbeschreibungen*, PhD thesis, Technische Universitaet Muenchen, 1978.

[Hudson86] Hudson, S.E., King, R., *A Generator of Direct Manipulation Office Systems*, ACM Transactions on Office Information Systems, Vol. 4, No. 2, April 1986, pp. 132-163.

[Janssen93] Janssen, C., Weisbecker, A., Ziegler, J., *Generating User Interfaces from Data Models and Dialogue Net Specifications*, in Proceedings of the Conference on Human Factors in Computing Systems INTERCHI'93 « Bridges Between Worlds » (Amsterdam, 24-29 April 1993), S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel, T. White (Eds.), ACM Press, New York, 1993, pp. 418-423.

- [Johnson92b] Johnson, P., Markopoulos, P., Johnson, H., *Task Knowledge Structures: A specification of user task models and interaction dialogues*, in Proceedings of 11<sup>th</sup> Interdisciplinary workshop on informatics and psychology, Vol. 6, 1992.
- [Lonczewski95a] Lonczewski, F., *PLUG--IN: Using Tcl/Tk for Plan Based User Guidance*, in Proceedings of the Tcl/Tk Workshop (Toronto, 6-8 July 1995), USENIX Association, 1995, pp. 141-144.
- [Lonczewski95b] Lonczewski F., *Using a WWW-Browser as an alternative user interface for interactive applications*, in Poster Proceedings of the 3<sup>rd</sup> World Wide Web Conference (Darmstadt), R. Holzapfel (Ed.), Fraunhofer Institute for Computer Graphics, 1995, pp. 132-135.
- [Luo93] Luo, P., Szekely, P., Neches, R., *Management of Interface Design in HUMANOID*, in Proceedings of the Conference on Human Factors in Computing Systems INTERCHI'93 «Bridges Between Worlds» (Amsterdam, 24-29 April 1993), S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel, T. White (Eds.), ACM Press, New York, 1993, pp. 107-114. <http://www.isi.edu/isd/CHI93-manager.ps>
- [Moriyón94] Moriyón, R., Szekely, P., Neches, R., *Automatic Generation of Help from Interface Design Models*, in Companion of the Conference on Human Factors in Computing Systems CHI'94 «Celebrating Interdependence» (Boston, 24-28 April 1994), C. Plaisant (Ed.), ACM Press, New York, 1994, pp. 225-231. <http://www.isi.edu/isd/CHI94-Help.ps>
- [Olsen86] Olsen, D.R., *MIKE: The Menu Interaction Kontrol Environment*, ACM Transactions on Information Systems, Vol. 5, No. 4, pp. 318-344.
- [Olsen89] Olsen, D.R., *A programming language basis for user interface management*, in Proceedings of the Conference on Human Factors in Computing Systems CHI'89 «Wings for the mind» (Austin, 30 April-4 May 1989), K. Bice, C. Lewis (Eds.), ACM Press, New York, 1989, pp. 171-176.
- [Schreiber93] Schreiber, W., *Prosaische Logik fuer Dichter und Denker -- Textverarbeitung massgeschneidert*, Forschung fuer Bayern, Vol. 6, Technische Universitaet Muenchen, 1993.
- [Schreiber94a] Schreiber, S., *The BOSS System: Coupling Visual Programming with Model Based Interface Design*, in Proceedings of 1<sup>st</sup> Eurographics Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'94 (Bocca di Magra, 8-10 June 1994), Focus on Computer Graphics Series, Springer-Verlag, Berlin, 1995, pp. 161-179. <ftp://hpeick7.informatik.tu-muenchen.de/pub/papers/sis/eg94.ps.Z>
- [Schreiber94b] Schreiber, S., *Specification and Generation of User Interfaces with the BOSS-System*, in Proceedings of the East-West International Conference on Human-Computer Interaction EWHCI'94 (St. Petersburg, 1994), B. Blumenthal, J. Gornostaev, C. Unger (Eds.), Lecture Notes in Computer Sciences, Vol. 876, Springer-Verlag, Berlin, 1994, pp. 107-120. <ftp://hpeick7.informatik.tu-muenchen.de/pub/papers/sis/ewhci94.ps.Z>

- [Schwab95] Schwab, R., *Generierung von Standardbedienoberflaechen aus Applikationsbeschreibungen*, Master's thesis, Technische Universitaet Muenchen, 1995.
- [Siemens92] *Telefon Bedienungsanleitung Hicom Standard 300*, Siemens AG, 1992.
- [Wiecha89] Wiecha, C., Bennett, W., et al., *Generating Highly Interactive User Interfaces*, in Proceedings of the Conference on Human Factors in Computing Systems CHI'89 « Wings for the mind » (Austin, 30 April-4 May 1989), K. Bice, C. Lewis (Eds.), ACM Press, New York, 1989, pp. 277-282.