25 ans
Institut d'Informatique
FUNDP • NAMUR

Computer-Aided Design of User Interfaces

Proceedings of CADUI '96

Edited by
Jean VANDERDONCKT

PRESSES UNIVERSITAIRES DE NAMUR

Generating User Interfaces from Formal Specifications of the Application

Bernhard Bauer

Abstract

The generation of the dialogue description from an algebraic specification of the application and its restrictions to different user groups are presented. The idea and motivation for the work is that the development of the application and the UI has to go hand in hand. Moreover, the UI should be generated since the programming of UIs is a time consuming and error-prone task. A formal specification of an application, characterizing the application in an abstract way, allows the automatic analyses and the generation of specifications, describing the dynamic behaviour of the UI. The generated (dynamic) specification can be used as an input for an existing UI Generator (UIG), called BOSS, which is part of a formal UI development environment, called FUSE.

Keywords

Algebraic specifications, user interface generation, model-based approach, user interface, formal methods, application of theorem provers, links between application and UI.

Introduction

Nowadays nearly every software project has to deal with the implementation of UIs, since the end-users of such systems are often computer novices using only the program with little or less knowledge about the computer technology. But the programming of UIs is not a trivial task, especially implementing the dialogue control, since the implementation is a time-consuming, error-prone and complex SE process and therefore expensive.

Moreover the development of a graphical UI is a very critical point in the software engineering process, since the complete interaction between the user and the application is via the UI and according to [Myers88a] 50-88% of the code of an interactive application is the code for the UI. Furthermore the price for individual software should be low to enter into competition with other software developers. Necessary is the generation of UIs from higher specifications, i.e., "I tell you what, you work it out". The software engineer should only describe the "global" information of the UI and define style-guides for the dialogues and presentations. This style-guides have to be defined once and are usable for the generation of a lot of UIs. These style-guides allow to get consistent UI for a family of products with the same look and feel.

Considering a whole application with a UI three layers have to be distinguished:

1. The specification of the **presentation** (layout) the user is interacting with.
2. The specification of the **dialogues** or **tasks** (dynamics) describing all possible dialogues, in a layout-independent way (as presented for document architecture systems in [Eickel90]).
3. The specification of the **application** (functional core) offering an appointed functionality which must be supported by the UI.

Taking this scheme into consideration and looking at the UI development process it is obvious that the UI cannot be constructed without the knowledge of the application, since the application interface, the dynamics of the UI and the user tasks are not independent of the application, since the state of the application controls inherent the performable dialogues. Therefore it is necessary to use the application as a starting point for the UI development.

But which description of the application should be used? An informal specification, a formal specification or the implementation of the application? Using an informal specification does not allow the use of machine supported analyzing of the specification. On the other side, the implementation of the application is too low-level to be considered. Furthermore the implementation of the UI has to be done in parallel to the implementation of the functional core to finish the implementation of both at nearly the same time.

Working with a formal specification technique allows:

- computer supported analyzation of the specifications,
- elucidating the problem and
- consideration of correctness aspects of the obtained software.

Thus the starting point for the UI and the application development is the same, namely a formal specification of the application and the software construction of both can be done hand in hand. In our framework as a starting point for the generation of UIs, algebraic specifications of the applications are used because on the one side this technique allows the abstract specification of the application, describes the input/output behaviour and allows the use of theorem proving techniques for obtaining correct software and on the other side are well-studied (cf. e.g., [Ehrig85, Wirsing90]). The output of the generation process are HIT specifications [Schreiber96] used for the generation of an executable UI with BOSS [Schreiber94a, Schreiber94b] ("**B**edien**O**berflächen**S**pezifikations**S**ystem" the german translation of "UI specification system") and state transition systems. The here presented work is part of the FUSE system (**F**ormal **U**I **S**pecification **E**nvironment) presented in [Lonczewski96] in this volume. The FUSE system consists of the three components BOSS [Schreiber94a, Schreiber94b], FLUID (**F**ormal **UI** **D**evelopment) and PLUG–IN [Lonczewski 95a, Lonczewski95b] (**PL**an-based **U**ser **G**uidance for **I**ntelligent **N**avigation). Within the FUSE architecture, the FLUID system plays the role of a theorem prover (cf. [Bauer95]) and an automatic dialogue designer. This contribution concentrates on the generation of the formal specification of the logical UI - called in the following often *dynamics* of the UI - from the formal specification of the application (i.e., problem domain model and user model).

1 The Problem

As already mentioned in the introduction the programming of UIs is a time-intensive and expensive SE task. Therefore it would be desirable to generate UIs out of a higher specification with the aim "I tell you what, you work it out". One aim is the re-use of the specification of the application for the generation of the UI. Using algebraic specifications (being a well-founded formal specification technique, cf. e.g., [Wirsing90]) for the generation process allows a unifying starting point for the UI and application development. The specification of the application is taken as input of the generation process and the output is a HIT specification or a state transition system describing the possible dialogues with the UI on a logical view. This HIT specification in connection with a given runtime system allows the prototypical development and evaluation of a UI with BOSS.

1.1 The Starting Point

Following [Larson92] the UI design decision framework consists of the following five classes:

- The **structural** and **functional** decision class determine the end users' conceptual model,
- the **dialogue** decision class determines the dialogue style and
- the **presentation** and **pragmatic** decision class determines the refinement of the end users' conceptual model and dialogue style.

In the structural and the functional decision class the structure of the end users' conceptual model is specified including

- the description of conceptual objects (consumed, produced and/or accessed by the end user),
- the application functions and
- the description of constraints and relationships that hold among conceptual objects).

I.e., more or less an abstract datatype with a special observable interface is defined in the structural and functional decision class. Such an abstract datatype can easily be specified by an algebraic specification.

We assume the reader to be familiar with the basic notions of algebraic specifications such as signature $\Sigma = (S, C, F)$, Σ-terms $T_\Sigma(X)$, ground terms T_Σ, (ground) substitutions σ, set of partial Σ–algebras $Alg^{partial}(\Sigma)$ (for more details see [Ehrig85, Wirsing90]).

Let $\Sigma = (S, C, F)$ be a signature consisting of a set of sort symbols S, constructor symbols C and function symbols F and Ax a set of equations of the form $t = r$ with $t, r \in T_\Sigma(X)$, whereby the function symbols in $f \in F$ with functionality $f : s1, s2,..., sn \rightarrow s$ may be partial (with $s1, s2,..., sn, s \in S$), i.e. there are some restrictions on the parameters denoted in the following way:

$$fct(f) = x_{f, s1} : s1, x_{f, s2} : s2,..., x_{f, sn} : sn \,.\, Eq_f(x_{f, s1}, x_{f, s2},..., x_{f, sn}) \text{ -> } s$$

such that f is only defined if $Eq_f(x_{f, s1}, x_{f, s2},..., x_{f, sn})$ is valid, whereby $Eq_f(x_{f, s1}, x_{f, s2},..., x_{f, sn})$ is an equation with the only identifiers in $\{ x_{f, s1}, x_{f, s2},..., x_{f, sn} \}$.

A subset Obs of the sorts S is distinguished being the observable sorts.

A partial algebraic specification is a tuple $Sp = <\Sigma, Obs, Ax>$.
The semantics is defined by its signature Σ and the behavioural class

$$Beh(Sp) = \{ A \in Alg^{partial}(\Sigma) \mid A \mid=_{beh} ax \text{ for all axioms } ax \in Ax \}.$$

The behavioural satisfaction $\mid=_{beh}$ is defined by

$$A \mid=_{beh} t = r \text{ iff for all context } c[z_s] \text{ of observable sort holds } A \mid= c[t] = c[r]$$

whereby a Σ-context $c[z_s]$ is a term over the signature Σ with a distinguished identifier z_s occurring exactly once in c. The application of a context $c[z_s]$ to a term t (denoted by c[t]) is done by substituting the identifier z_s by t if t is of sort s. $\mid=$ denotes the usual satisfaction relation.

The model class of an algebraic specification is defined by:

$$Mod(Sp) = \{ A \in Alg^{partial}(\Sigma) \mid A \mid= ax \text{ for all axioms } ax \in Ax \}.$$

The sorts and constructor symbols define the conceptual objects, the function symbols the application functions, the observable sorts characterize those objects which are observable by the end-user and the parameter restrictions with the axioms describe the constraints and relationships between the conceptual objects.

The notion of algebraic specifications has to be extended by a set of distinguished function symbols applicable to the conceptual objects (called in the following *interface functions*) which should be supported by the UI and the sort of the application state, i.e., the sort of the terms representing the state of the functional core. The use of interface functions cannot be neglected by identifying the function symbols with observable result sort as the interface function, since it would be desirable to use application functions only changing the internal state of the application. Furthermore the initial state of an application may be defined.

Note, that the meaning of the functions (by defining the semantics of the functions by axioms and parameter restrictions) is specified, but not their format or sequencing of invocation is defined.

The three important types of decisions made in the dialogue decision class are

- what are the units of information exchanged between the user and the application (defined by the observable sorts and the interface functions),
- how this units of information are structured into messages between the user and the application (not considered here) and
- what the appropriate sequences of message exchange are (main issue of this contribution).

The aim of the new approach is to generate the sequence of information exchanged between the user and the application, namely to automate part of the dialogue decision class.

1.2 Specification of the Application: an Example

We start with the algebraic specification *ISDN-Application* of the application. A similiar specification can be found in [Bauer95]. The specification of the ISDN telephone is a syntactical enrichment of the natural numbers (*NAT*). The sorts describe the connection with a participant (*Connection*), the internal state of the telephone (*State*) and the state of a connection (*Cstate*).

The internal state is viewed in an abstract way, i.e., at most two connections can be achieved with the telephone (*mkState*). *mtCon* states an empty connection. A (non-empty) connection consists of a telephone number (represented by a natural number being the only observable sort) and the status of the line (*mkCon*). A line can either be *waiting* or *telephoning*.

The function *call* describes the telephone call with a single participant, *secondCall* starts a telephone call with a second participant and the conference function enables a conference session between the user of the telephone and the two participants on the other lines. *call*, *secondCall* and *conference* have parameter restrictions denoted by a first order formulae after **pre**.

All telephone calls are ended with *endCalls*. *emptyConnections*, *singleConnections* and *doubleConnections* are predicates stating none, one and two connections. The interface functions, i.e., the set of functions which should be supported by the UI are *call*, *secondCall*, *conference* and *endCalls*.

spec ISDN-Application =
 enrich NAT **by**
  sorts Connection, CState, State
  obs-sorts Nat
  cons
   mkState: Connection, Connection -> State,
   mtCon: -> Connection,
   mkCon: Nat, CState -> Connection,

waiting, telephoning: -> CState
opns
call: Nat, $x_{call, State}$: State. **pre** emptyConnections($x_{call, State}$) = true -> State,
secondCall: Nat, $x_{secondCall, State}$: State. **pre** singleConnections($x_{secondCall, State}$) = true -> State,
conference: $x_{conference, State}$: State. **pre** doubleConnections($x_{conference, State}$) = true -> State,
endCalls: State -> State,
emptyConnections: State -> Bool,
singleConnections: State -> Bool,
doubleConnections: State -> Bool
interface functions call, secondCall, conference, endCalls
axioms forall nr, nr2: Nat, s: State.
emptyConnections(mkState(mtCon, mtCon)) = true,
emptyConnections(mkState(mkCon(nr, cs), c)) = false,
singleConnections(mkState(mkCon(nr, cs), mtCon)) = true,
singleConnections(mkState(mtCon, c)) = false,
singleConnections(mkState(mkCon(nr, cs), mkCon(nr, cs))) = false,
doubleConnections(mkState(mkCon(nr, cs), mkCon(nr, cs))) = true,
doubleConnections(mkState(c, mtCon)) = false,
call(nr, s) = mkState(mkCon(nr, telephoning), mtCon),
secondCall(nr, call(nr2, s)) = mkState(mkCon(nr2, waiting), mkCon(nr, telephoning)),
conference(secondCall(nr, call(nr2, s))) =
mkState(mkCon(nr, telephoning), mkCon(nr2, telephoning)),
endCalls(s) = mkState(mtCon, mtCon)
endspec

Because of lack of space (large figures are obtained) and in order to keep the specification small, not the whole functionality presented in [Lonczewski96] in this volume is given, especially with endCalls a conference session is ended and the switching between two participants is omitted.

These features can easily be added to the specification and the generation would be analogous. In this paper mainly the generation idea should be described to get a feeling how the generation is performed.

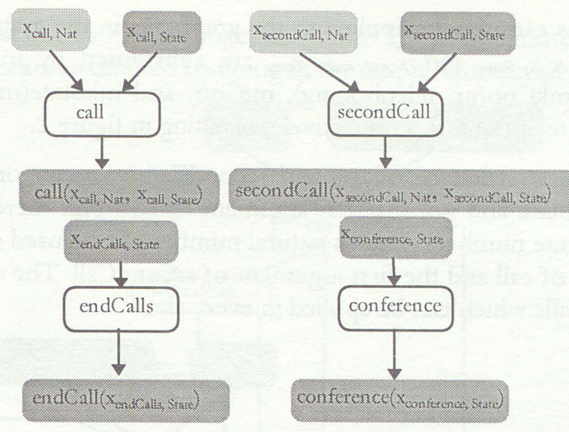
2 The Generation Idea of the Dialogue Specification

In this section the idea for the generation of the dialogue specifications (HITs and state transition systems) and their restrictions to different user groups are informally described.

2.1 Generation of the Dialogue Specifications

The generation process consists of several steps:

As a first step a graph is constructed with nodes marked with function symbols, identifiers for the arguments and the resulting term for each interface function. The only non-observable sort is the sort of the state of the functional core, namely State, marked with ▩ and observable arguments are marked with ▢.



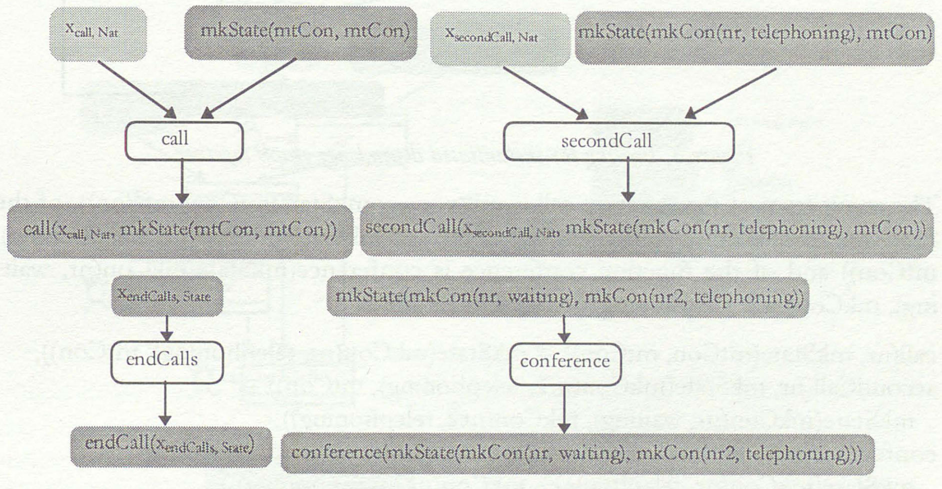
$x_{call, Nat}$
$x_{call, State}$
$x_{secondCall, Nat}$
$x_{secondCall, State}$
call
secondCall
call($x_{call, Nat}$, $x_{call, State}$)
secondCall($x_{secondCall, Nat}$, $x_{secondCall, State}$)
$x_{endCalls, State}$
$x_{conference, State}$
endCalls
conference
endCall($x_{endCalls, State}$)
conference($x_{conference, State}$)

Figure 1. First dependency graph

Now all the parameter restrictions for the functions can be solved by a system solving existential quantified equations by narrowing like RAP [Hußmann89]. Therefore the solutions for the identifiers in the parameter restrictions must be calculated, i.e., the solutions of the existential quantified formulae:

$\exists\, x_{call, State}$: State. emptyConnections($x_{call, State}$) = true,
$\exists\, x_{secondCall, State}$: State. singleConnections($x_{secondCall, State}$) = true and
$\exists\, x_{conference, State}$: State. doubleConnections($x_{conference, State}$) = true

The solutions - denoted here as substitutions - can be easily calculated as
$\sigma 1$ = { mkState(mtCon, mtCon) / $x_{call, State}$ },
$\sigma 2$ = { mkState(mkCon(nr, telephoning), mtCon) / $x_{secondCall, State}$ } and
$\sigma 3$ = { mkState(mkCon(nr, waiting), mkCon(nr2, telephoning)) / $x_{conference, State}$ }



$x_{call, Nat}$
mkState(mtCon, mtCon)
$x_{secondCall, Nat}$
mkState(mkCon(nr, telephoning), mtCon)
call
secondCall
call($x_{call, Nat}$, mkState(mtCon, mtCon))
secondCall($x_{secondCall, Nat}$, mkState(mkCon(nr, telephoning), mtCon))
$x_{endCalls, State}$
mkState(mkCon(nr, waiting), mkCon(nr2, telephoning))
endCalls
conference
endCall($x_{endCalls, State}$)
conference(mkState(mkCon(nr, waiting), mkCon(nr2, telephoning)))

Figure 2. Instantiated dependency graph

These substitutions can now be applied to the graph, i.e. in the graph the identifiers $x_{\text{call, State}}$, $x_{\text{secondCall, State}}$ and $x_{\text{conference, State}}$ are substituted by mkState(mtCon, mtCon), mkState(mkCon(nr, telephoning), mtCon) and mkState(mkCon(nr, waiting), mkCon(nr2, telephoning)), respectively, resulting in figure 2.

Since the parameter restrictions of call and secondCall influence only the second argument of sort State and not the first argument of sort Nat there is no restriction on the telephone numbers. Thus a natural number can be used as an input for the first argument of call and the first argument of secondCall. The same holds for the function endCalls which can be applied in every state.

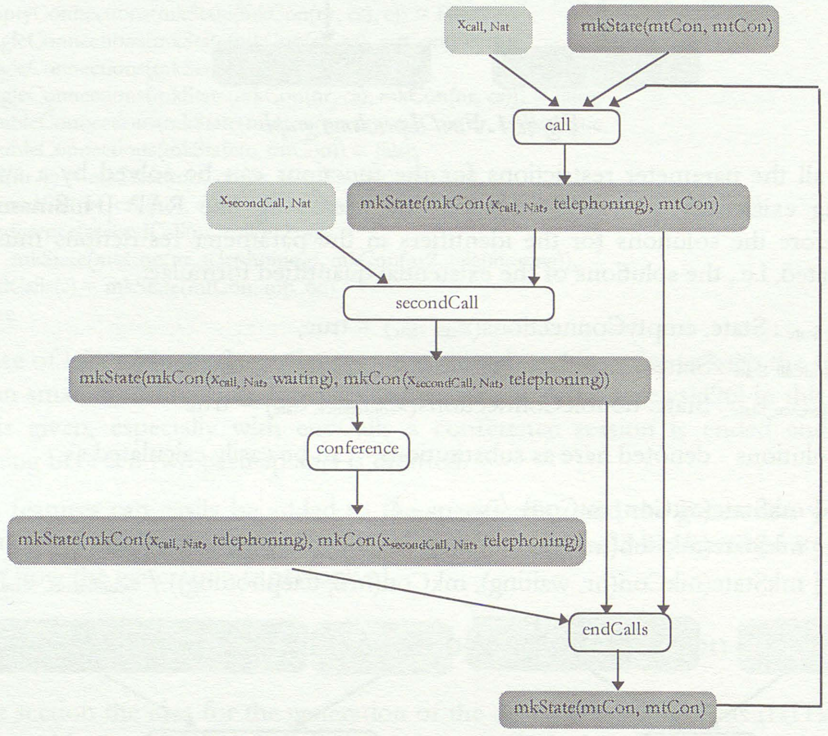

x call, Nat
mkState(mtCon, mtCon)
call
x secondCall, Nat
mkState(mkCon(x call, Nat, telephoning), mtCon)
secondCall
mkState(mkCon(x call, Nat, waiting), mkCon(x secondCall, Nat, telephoning))
conference
mkState(mkCon(x call, Nat, telephoning), mkCon(x secondCall, Nat, telephoning))
endCalls
mkState(mtCon, mtCon)

Figure 3. Putting the instantiated dependency graph together

The result term of the function call is call($x_{\text{call, Nat}}$, mkState(mtCon, mtCon)), of the function secondCall is secondCall($x_{\text{secondCall, Nat}}$, mkState(mkCon(nr, telephoning), mtCon)) and of the function conference is conference(mkState(mkCon(nr, waiting), mkCon(nr2, telephoning))). Moreover it holds

call(nr, mkState(mtCon, mtCon)) = mkState(mkCon(nr, telephoning), mtCon)),
secondCall(nr, mkState(mkCon(nr2, telephoning), mtCon)) =
  mkState(mkCon(nr, waiting), mkCon(nr2, telephoning)),
conference(mkState(mkCon(nr, waiting), mkCon(nr2, telephoning)) =
  mkState(mkCon(nr, telephoning), mkCon(nr2, telephoning)))
and endCalls(s) = mkState(mtCon, mtCon) for all States s.

Now the graphs can be merged together (figure 3) and the non-observable state of the application can be omitted resulting in the graph reproduced in figure 4.

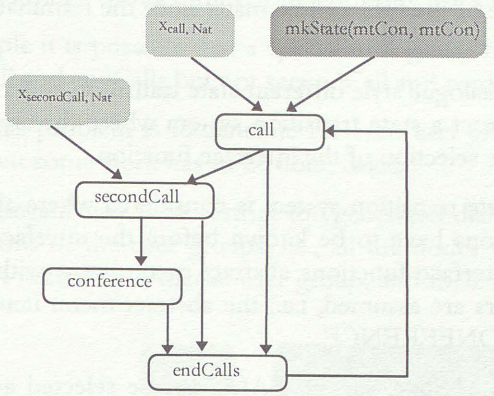


Figure 4. Composed instantiated dependency graph

The obtained graph can now be translated on the one side into a state transition system and on the other side into a BOSS specification. In this generation process special dialogue style guides (specifiable in a formal way by defining transformation rules for the obtained graphs) can be used, e.g., for a user or system driven dialogue style. We assume here a hard-coded transformation into the dialogue specifications.

A transaction-rule in BOSS (for more details, see [Lonczewski96] in this volume and [Schreiber96]) is fired by the user, e.g., by selecting a menu-item, or by a push-button., i.e., each interface function is viewed as a non-repeatable transaction rule and the observable arguments as input slots, i.e., the user has to enter some information for it. The corresponding BOSS-specification looks like figure 5.

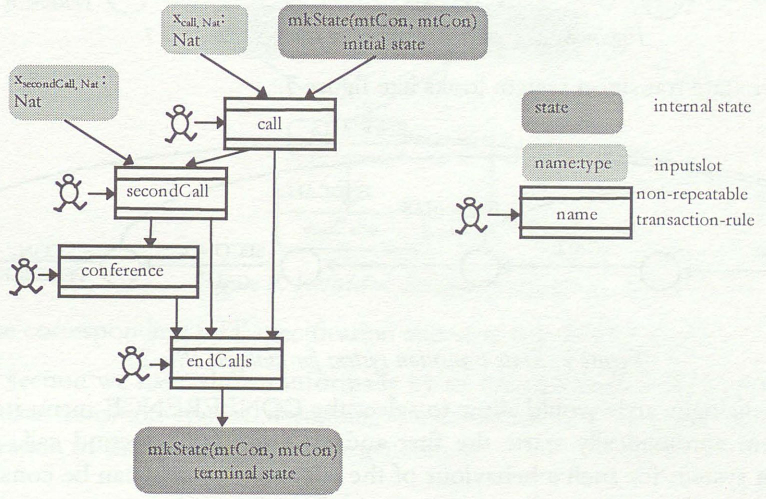


Figure 5. HIT specification

Using non-repeatable transaction rules states, that the whole HIT has to be worked through starting with the initial state until the termination state is reached. Now a new instance of the HIT can be made since the termination state is equal to the initial state.

Depending on the dialogue style different state transition systems are obtained. Let us first of all construct a state transition system where the arguments are entered after performing the selection of the interface function.

As a next step a state transition system is considered where all the parameters of the interface functions have to be known before the interface function is determined. With the interface functions abstract menu items with the function symbols in capital letters are assumed, i.e., the abstract menu items are CALL, SECONDCALL and CONFERENCE.

Starting with an initial state, say s0, CALL can be selected according to the dependency graph in figure 4. Afterwards the telephone number (a natural number) has to be entered. After performing a call either a second call can be started (beginning with SECONDCALL and entering the telephone number afterwards) or the telephone call can be ended (ENDCALLS).

After performing a second call either all telephone calls can be ended (ENDCALLS) or a conference sessions can be started (CONFERENCE) and then all telephone calls can be ended (ENDCALLS). The obtained state transition system looks like figure 6.

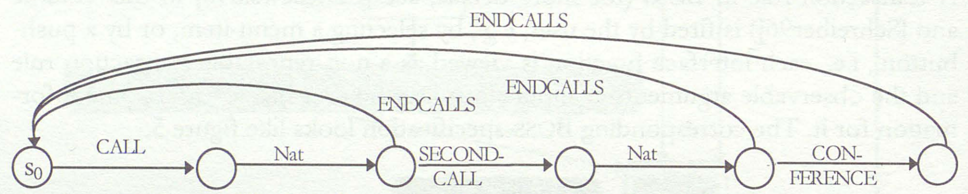


Figure 6. State transition system for dialogue style 1

The other state transition system looks like figure 7.

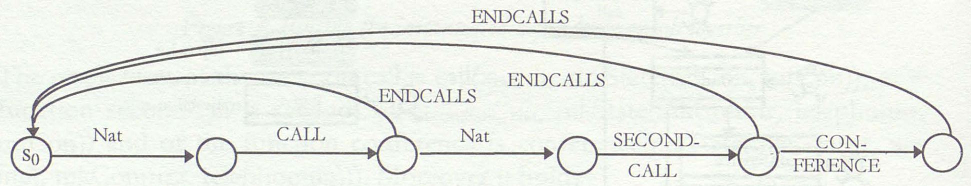


Figure 7. State transition system for dialogue style 2

Another dialogue style would allow to select the CONFERENCE menu-item and the system automatically starts the first and afterwards the second call. A state transition system for such a behaviour of the telephone system can be constructed analogous.

2.2 Restricting the Dialogue Specification to Different User Groups

Usual different user groups with a different functionality use a software product.

In the ISDN-example it is possible that a special user group may only use the interface functions call and endCalls but not secondCall and conference.

One solution for this problem is to generate for each user group a different dialogue description, but some work has to be done twice.

Therefore a more elegant way is to restrict the generated dialogue description to the interface functions of the user groups, i.e., all the nodes with interface functions, which are not usable by a special user group, and their argument nodes are "deleted":

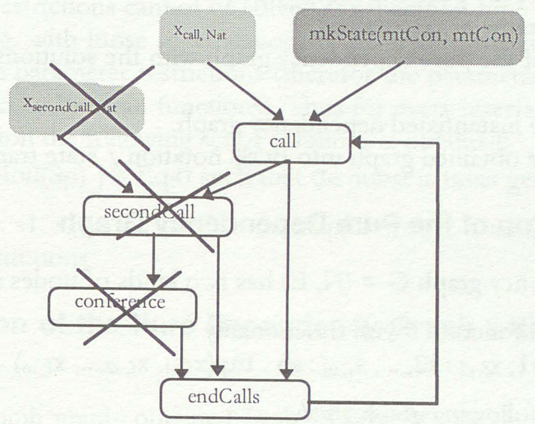


Figure 8. Restricting the dialogue specification to different user groups

resulting in:

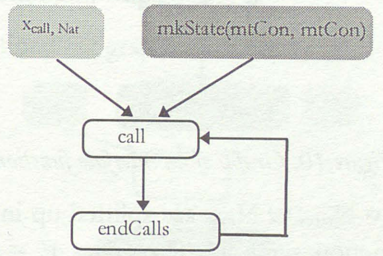


Figure 9. Restricted dialogue specification

with the corresponding HIT specification and state transition system.

In this section we have shown informally by an example how a HIT specification and a state transition system, describing the dynamics of a UI out of an algebraic specification of the application can be generated.

3 Generating a Specification of the Performable Dialogues

In the previous section we have seen by an example what the idea of generating the dialogue specification from an algebraic specification is. The starting point is a given algebraic specification $Sp = <(\Sigma, C, F), Obs, Ax>$.

The sorts are splitted up into observable and non-observable sorts and the state sort, i.e. the observable sorts describe those objects visualizable to the end-user and the non-observable objects not visible by the end-user and the objects of the state sort describe the internal state of the application also not visible by the user.

The generation process consists of five phases:

1. Construction of the pure dependency graph.
2. Solving the parameter restrictions.
3. Instantiation of the pure dependency graph with the solutions of the parameter restrictions.
4. Merging of the instantiated dependency graph.
5. Converting the obtained graph into BOSS notation / state transition system.

3.1 Construction of the Pure Dependency Graph

The pure dependency graph $G = (N, E)$ has two kinds of nodes and edges.

For each interface function f with functionality

$$fct(f) = x_{f,s1} : s1, x_{f,s2} : s2, \ldots, x_{f,sn} : sn \,.\, Eq_f(x_{f,s1}, x_{f,s2}, \ldots, x_{f,sn}) \text{ -> } s$$

we construct the following graph *graph$_f$*:

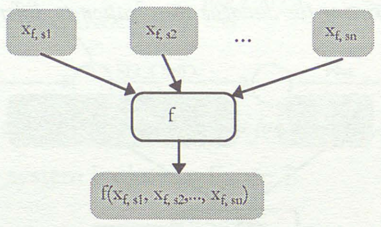


Figure 10. Graph of an interface function f

Therefore the nodes $N = N_{term} \cup N_{func}$ are splitted up into N_{term} the set of terms and N_{func} the set of function symbols. The edges $E = E_{termtofunc} \cup E_{functoterm} \cup E_{functofunc}$ are splitted into edges from $n_{term} \in N_{term}$ to nodes $n_{func} \in N_{func}$ in the set $E_{termtofunc}$, edges from $n_{func} \in N_{func}$ to nodes $n_{term} \in N_{term}$ in the set $E_{functoterm}$ and edges from $n_{func} \in N_{func}$ to $n_{func} \in N_{func}$ in the set $E_{functofunc}$. $E_{functofunc}$ are used later.

The pure dependency graph is the set of graphs of each interface function f.

3.2 Solving the Parameter Restrictions

In this phase it is tried to solve the parameter restrictions of the interface functions, i.e. the solution of the parameter restriction for an interface function f with functionality

$$fct(f) = x_{f,s1} : s1, x_{f,s2} : s2, \ldots, x_{f,sn} : sn \,.\, Eq_f(x_{f,s1}, x_{f,s2}, \ldots, x_{f,sn}) \text{ -> } s$$

are the solutions of the existential formulae:

$$\exists\, x_{f,s1} : s1, x_{f,s2} : s2, \ldots, x_{f,sn} : sn \,.\, Eq_f(x_{f,s1}, x_{f,s2}, \ldots, x_{f,sn})$$

To solve existential quantified formulae theorem provers can be applied, namely the solutions can be found by narrowing (e.g., by the RAP system [Hußmann89]), whereby the most general solutions are obtained.

If the parameter restrictions cannot be solved at generation time (because information is missing, e.g. with loose specifications is dealt with) the run-time system of BOSS controls the parameter restrictions (therefor the parameter restrictions have to be implemented by Boolean functions). Thus for every interface function f with parameter restriction the following set of solutions is obtained:

$$\sigma(f) = \{ \sigma \mid Mod(Sp) \models Eq_f\sigma \text{ such that } \sigma \in Subst \text{ is most general solution } \}$$

with $fct(f) = x_{f,s1} : s1, x_{f,s2} : s2, \ldots, x_{f,sn} : sn \,.\, Eq_f(x_{f,s1}, x_{f,s2}, \ldots, x_{f,sn}) \text{ -> } s$ and Subst is the set of all substitutions.

3.3 Instantiation of the Pure Dependency Graph with the Obtained Solutions

Now for every graph graph$_f$ obtained from an interface function f the set of instantiated graphs instgraph$_f$ is defined by:

$$instgraph_f = graph_f, \text{ if no solution exists,}$$
$$instgraph_f = \cup_{\sigma \in \sigma(f)} \sigma(graph_f) \text{ otherwise}$$

such that $\sigma(graph_f)$ is defined for graph$_f$ of figure 10 by:

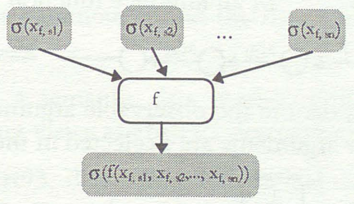


Figure 11. Applying a substitution to a graph

3.4 Merging of the Instantiated Dependency Graphs

After calculating the instantiated set of graphs

$$InstGraphs = \cup_{f \in interface(Sp)} instgraph_f$$

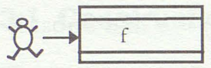
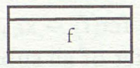
whereby interface(Sp) yields the interface functions of the application. The set of instantiated graphs InstGraphs is examined whether nodes of sort N_{term} can be connected. An edge between two nodes t1, t2 $\in N_{term}$ is drawn if Mod(Sp) $\models$ t1=t2 holds and there exists an edge (t1, f1) $\in E_{termtofunc}$ and an edge (f2, t2) $\in E_{functoterm}$ for some function symbols f1 and f2 and terms t1 and t2. If an edge from t2 to another term t of N_{term} exists then instgraph$_f$ is duplicated. The new obtained graph is merged together in the following way:

If edges (f1, t1) $\in$ E and (t1, f2) $\in$ E exist

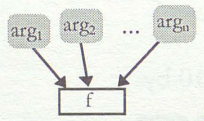
- and there is no edge (t1, f3) $\in$ E (with f3 $\neq$ f2) then (f1, t1) and (t1, f2) are deleted in E and (f1, f2) is added to E.
- and there is an edge (t1, f3) $\in$ E (with f3 $\neq$ f2) then (f1, f2) is added to E.

3.5 Obtaining a Boss Specification / State Transition System

The obtained graph of the merging phase is converted into a HIT-specification as follows:

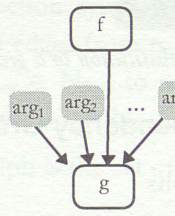
Each node $\boxed{f}$ of an interface function f is converted into a transaction rule  if f is interface functions and an equational rule  otherwise.

The obtained graph of the merging phase is converted into a state transition system as follows. Depending on the dialogue style different state transition systems can be constructed. The transformation presented here is performed by first selecting the abstract menu item of the corresponding interface function and then entering the arguments.

Each subgraph  of an interface function f is converted into

$$\begin{tikzcd} \bigcirc \arrow[r, "f"] & \bigcirc \arrow[r, "arg_1"] & \bigcirc \arrow[r, "arg_2"] & \bigcirc \arrow[r] & \dots \arrow[r, "arg_n"] & \bigcirc \end{tikzcd}$$

such that $arg_1, arg_2, \dots, arg_n$ denote the observable arguments which have to be entered. The non-observable arguments are neglected in the state transition system.

Subgraphs of the form  of interface functions f and g are converted into

$$\begin{tikzcd} \boxed{f} \arrow[r, "g"] & \bigcirc \arrow[r, "arg_1"] & \bigcirc \arrow[r, "arg_2"] & \bigcirc \arrow[r] & \dots \arrow[r, "arg_n"] & \bigcirc \end{tikzcd}$$

such that $\boxed{f}$ denotes the state transition system obtained from the interface functions f and its arguments.

Cycles in the merged instantiated dependency graph are expressed analogous in the state transition system.

The restriction of the dialogue description for special user groups is done by deleting the non-usable interface functions from the obtained HIT specification or state transitition system.

Parallelism can also be taken into consideration in the generation process. Things can be done in parallel without synchronisation if the output of the fourth phase are non-dependent graphs. Then each of these graphs can be worked through in parallel.

Converting the graph of the fourth phase into BOSS is the same as described. The obtained state transition systems have to be put into the construct for expressing parallelism.

Using the structuring mechanisms well-known from algebraic specifications allows to use these technique also for larger projects. The experience shows that the generation of the dialogue description for subspecifications can often be put together without considering the context in which the subspecifications are used. Otherwise normalization techniques exists for the structured algebraic specifications and the normalized specification can be used as the starting point for the generation process.

4 Related Work

MIKE [Olsen86] (Menu Interaction Kontroll Environment) und MIKEY [Olsen89] generate UIs with menus and dialogue boxes based on a description of the functions (argument and result parameters) and the data structures in the application interface.

In HIGGENS [Hudson86] a semantic data model of the application interface is used as the base for deriving views as abstract descriptions of the UI layout.

The JANUS–System [Balzert93, Balzert94a, Balzert95a] uses OOA (Object–Oriented Analysis) for describing the problem domain model (i.e., application interface) of an data base–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.

In the UIDE system (UI Design Environment) [Foley91, Foley93, 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 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 an 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.

Recently the research on UIDE and HUMANOID were joint in the MASTERMIND project.

The GENIUS–System (GENerator for UIs Using Software Ergonomic Rules) [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 interface, 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 (TAsk based DEvelopment of UI Software) [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, an extension of dialogue–nets). In this system the dynamics of the application is not taken into consideration or the specification of the application is not used for dynamics considerations of the application.

ITS (Interactive Transaction System) [Wiecha89] offers a frame based language for the specification of UIs in its logical structures ("dialogue content"). Moreover, ITS allows the specification of style rules, which describe the mapping between logical UIs and UIs in a particular style.

In the ADEPT system [Johnson92b, Wilson96], a process–algebra–like specification technique called Task Knowledge Structures (TKS) 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 UI". 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 TRIDENT (Tools foR an Interactive Development ENvironmenT) system [Bodart94a, 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 AIO, rules for mapping AIO to CIO and rules for the placement of CIO.

These systems start more or less with the specification of the dynamics of the UI which is the output of the FLUID system and can therefore be seen at the same level as the BOSS system in the FUSE system. But they do not take the dynamic semantics of the application into consideration.

Conclusion

The FLUID system, whose theoretical foundations were presented here, is currently under development, whereby prototypes of BOSS and PLUG-IN already exist. The FUSE methodology and tools have been applied successfully to a number of examples (ISDN phone simulation, UI for a literature research system, UI for a home banking system, formula editor for $\LaTeX$).

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, BON and ERA data models in addition to the currently supported algebraic specification technique.

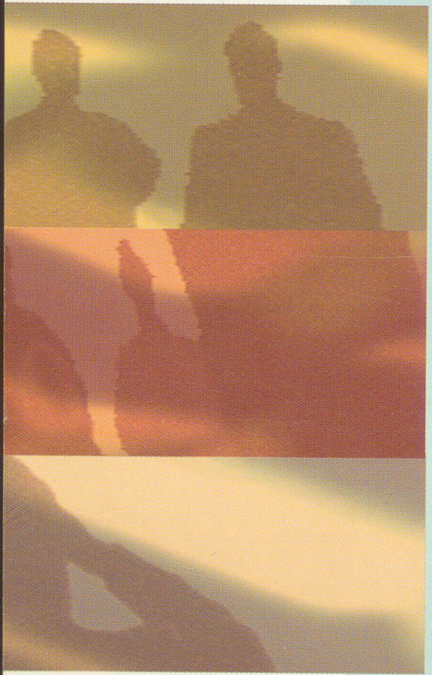
In order to gain more practical experience with the FUSE–methodology and the related tools, we plan to organize a course in UI specification and generation at the Munich University of Technology.

Acknowledgements

This work has been partially supported by Siemens Corporate Research and Development, Department of System Ergonomics and Interaction (ZFE ST SN 51). The author would like to thank Siegfried Schreiber and the anonymous reviewers for their useful comments and suggestions on draft versions of this paper.

COMPUTER-AIDED DESIGN OF USER INTERFACES

For the first time in the history of Human-Computer Interaction, this book gathers the most recent and up-to-date contributions of people, research teams and leading organisations involved in Computer-Aided Design of User Interfaces (CADUI). It provides practical advice on how to use various CADUI techniques to effectively and efficiently develop user interfaces of interactive applications.

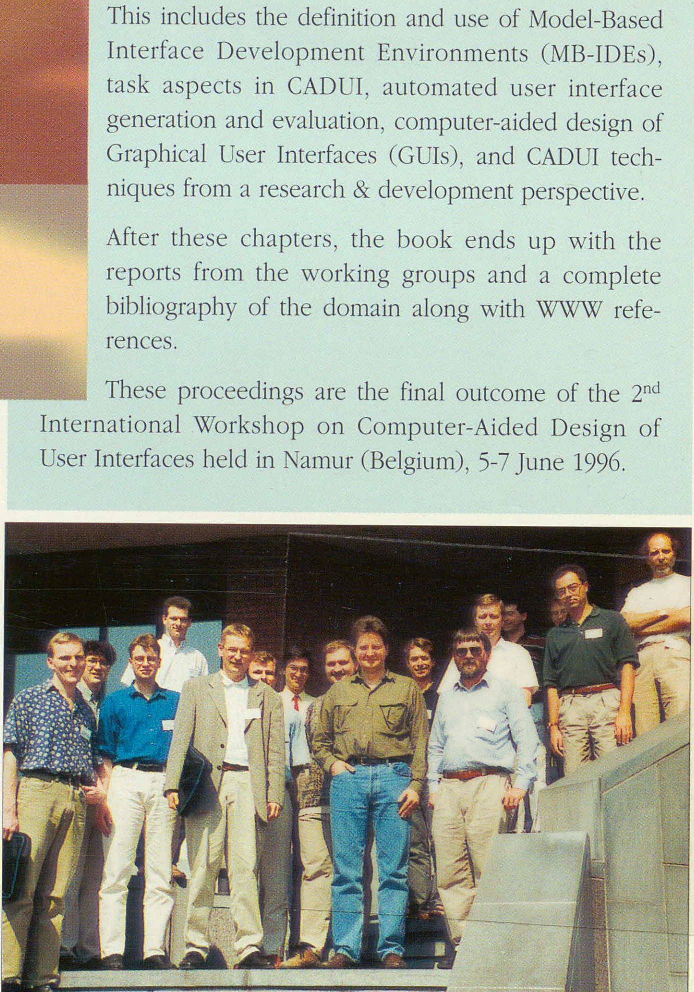


Computer-Aided Design of User Interfaces brings together in one place the invaluable experience the authors have gained during more than the last decade. This extensive experience is now given to a broad range of people who specify, model, design, prototype, generate, implement, evaluate user interfaces with the help of dedicated CADUI tools.

This includes the definition and use of Model-Based Interface Development Environments (MB-IDEs), task aspects in CADUI, automated user interface generation and evaluation, computer-aided design of Graphical User Interfaces (GUIs), and CADUI techniques from a research & development perspective.

After these chapters, the book ends up with the reports from the working groups and a complete bibliography of the domain along with WWW references.

These proceedings are the final outcome of the 2nd International Workshop on Computer-Aided Design of User Interfaces held in Namur (Belgium), 5-7 June 1996.



ISBN 2-87037-232-9

9 782870 372326