Software Life Cycle Automation for Interactive Applications: The AME Design Environment

Christian Märtin

Fachhochschule Augsburg, Fachbereich Informatik
Baumgartnerstraße 16, D-86161 Augsburg, Germany
Phone: +49-821-5586-454 – Fax: +49-821-5586-499
E-mail: maertin@informatik.fh-augsburg.de
WWW: http://www.fh-augsburg.de

Abstract

The model-based design environment AME offers CASE-tool support for all life cycle activities in the development process for interactive applications. The system allows the rapid automatic construction of interactive software from object-oriented analysis models (OOA) and/or OO-modelling information specified at later design stages. AME provides functionality for UI-structure generation, interaction object selection, layout prototype generation, dynamic behaviour generation, adaptation to user-specific requirements, integration of domain-methods and target code generation. Object-oriented and knowledge-based components provide automatic transition from one refinement stage to the next. System decisions can be visualised before code generation and may be revised by the designer.

Keywords

Design automation, life cycle, model-based approaches, object-oriented models, user interface generators, software engineering.

Introduction

In the next decade application system design will confront the software industry with a set of tough requirements with respect to complexity, usability, flexibility, multimedia-management, quality, time-to-market, ease of maintenance and other factors. In order to meet these challenges, the fields of software engineering and human-computer interaction have to join forces. Object-oriented analysis and design methods (OOA/OOD) [Monarchi92] seem to provide a common denominator for integrating software process automation and user interface design:

- Advanced object-oriented CASE-tools support all activities of the software life cycle. Object technology is now widely used and has become a major driving force for productivity and quality enhancements.

- Object-orientation has also been the principal design approach for the construction of interactive software, since the first applications with GUIs appeared [Goldberg84].

Most automated design approaches for interactive systems, however, do not use software life cycle models to define the various user interface development tasks. As no unified life cycle models exist, the majority of existing design environments for interactive systems also fail to achieve a true integration of the development requirements for the domain parts of the applications and for the user interface components.

Life cycle models should define functionality, sequencing and data interface requirements of all the activities in the development process for interactive systems: from analysis (problem definition) to design (solution specification) and implementation. It is also important to include mechanisms for concurrent design or clustering [Meyer95] of development tasks. Figure 1 shows an example of concurrent life cycles: the development process is divided into activities for the user interface and activities for the domain functionality of the system. Life cycle models also have to support incremental development requirements, especially if they aim to be accepted by designers of highly interactive systems.

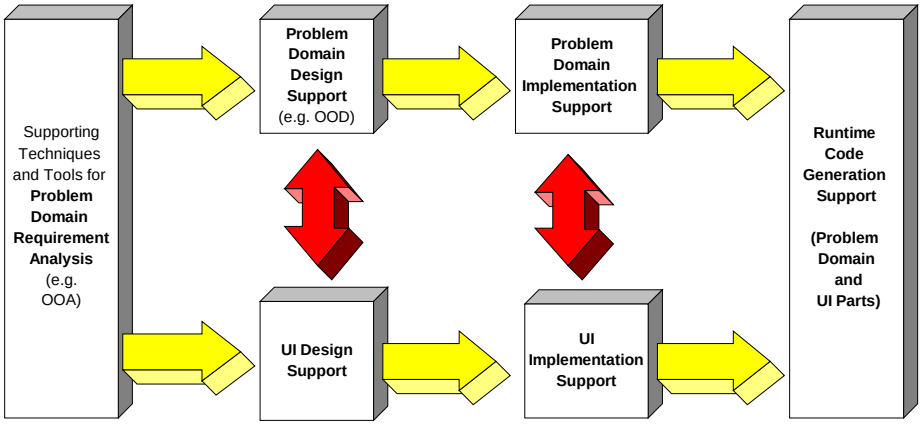

Figure 1. Concurrent life cycle support for user interface and problem domain software development

The Application Modelling Environment AME, which is discussed in this paper, is designed around an object-oriented life cycle model for the concurrent development of user interface and domain parts of interactive systems. AME is a prototypical model-based development environment for interactive business applications [Märtin93, Märtin95, Märtin96a]. AME offers high-level tools for the automatic refinement and generation of the standard parts of interactive systems as well as flexible specification tools for more domain-specific application components. At each life cycle step the designer may either choose to accept the generated solution, or to adapt it to her or his individual requirements.

The following section examines related work and existing model-based approaches for designing interactive systems. Section 2 introduces AME´s architecture and goals. In section 3 the various activities of AME´s software life cycle are demonstrated for an example application.

1 Related Work

The design of interactive applications can be supported by the following categories of tools: implementation- and system-level tools, model-based specification systems, model-based generators [Forbrig96].

1.1 Implementation- and System-Level Tools

Toolkits and *GUI editors* are examples of implementation-level tools. *UIMSs* for user interface definition, generation and runtime support are system-level tools. Approaches based on UIMSs do not integrate the results of the design activities for user interface and domain parts before the final development stages. This is why results from earlier life cycle activities, i.e., analysis and global design, cannot be exploited for user interface construction. *Visual programming environments* that are coupled with extended (object-oriented) programming languages are also system-level tools. They support the easy reuse and modification of existing interaction object classes, but leave it to the developer to couple UI designs and program code interactively.

Implementation- and system-level tools may be used as service-suppliers by high-level-design tools that belong to one of the two research-oriented categories, discussed in the following sections.

1.2 Model-Based Specification Systems

Model-based specification systems allow system developers to specify the structural, functional and dynamic features of the user interface of applications in close co-ordination with the domain parts.

Tools like UIDE [Sukaviriya93] or HUMANOÏD [Szekely93] yield a high-quality and flexible design-level model of the interactive system, which allows to represent both application-independent and application-specific interactive requirements in great detail. The systems support rapid prototyping rather than integrated life cycle models. The modelling process can be complex and time-consuming, however.

Both systems demand the explicit specification of user interface functionality and dynamics. The systems provide mechanisms for representing runtime-dependent application dynamics. The result of the modelling process is a detailed design specification of the interactive system, which can be translated into a working prototype of the application. UIDE supports the generation of context-based help as well as layout generation [Kim93]. HUMANOÏD incorporates a co-operative design-goal management system [Luo93]. The MECANO approach, which is discussed in [Pu-

erta96b] includes the explicit modelling of design processes in the form of meta-level models.

The IDA environment [Reiterer94, Reiterer95] provides advanced tools for the construction of graphical user interfaces of high quality. IDA uses an object-oriented approach for designing flexible, reusable interface templates. The construction tool is coupled with a UIMS. A hypertext-based consulting system provides design guidelines and presentational support. A knowledge based quality assurance tool evaluates the modelled prototype and proposes ways for improving the design.

Design critics for co-operative user interface development are also covered in [Fischer93]. EXPOSE [Gorny94, Gorny95] is a consulting expert system for the design of highly-ergonomic user interfaces. The TADEUS system, provides decision support and guidance for user interface design on the basis of a task model, a problem domain model, a dialogue model and a user model [Elwert94]. The system FUSE [Lonczewski96] uses algebraic specification techniques for modelling the static and dynamic parts of interactive systems.

1.3 Model-Based Generators

Model based generators create user interface prototypes from domain data models. ERA models or abstract object-oriented models are exploited by such tools. Some generators need additional state-transition-specifications for dynamic modelling. Generators are supposed to raise software productivity and to help application domain experts with the design of consistent user interfaces. However, the flexibility and the complexity of the generated user interfaces may be restricted.

TRIDENT [Vanderdonckt93] exploits entity-relationship-attributes and attribute-meta-data for a rule-based selection of interaction objects. Activity Chaining Graphs (ACG) specify the data flow during task execution and are used for generating dialogue dynamics. GENIUS [Janssen93] also exploits ERA models, additional meta-information and action-names for generating the static user interface of a window. Petri-net-like dialogue nets are interpreted for generating dialogue dynamics. UIDE [de Baar92] exploits attributes of domain object classes, action names and meta-data for generating application windows and their menus. All of these systems require explicit information about which of the domain data or object attributes will be grouped together in one target window.

The JANUS-approach [Balzert93, Balzert94, Balzert95a] uses Coad/Yourdon object-oriented models [Coad91a] for generating multi-window database applications. Each object class is mapped to a window of the user interface. Rules are used to translate object-attributes and operations to interaction objects or menu-entries. Inheritance, aggregation and association between object classes are exploited to construct the global structure of the user interface and to generate standard functions for navigating between windows. In order to build exploitable OOA models, application designers have to know the system's mapping rules. Specifications can-

not be changed at the OOD-level. No explicit dynamic modelling is supported. The specification systems TADEUS [Schlungbaum96] and FUSE [Bauer96] include components for generating user interfaces from model specifications, which explicitly cover dynamic aspects.

2 The AME Environment

The Application Modelling Environment (AME), which is discussed in the following, is an experimental CASE-environment with full life cycle support for interactive systems development. AME integrates object-oriented and knowledge-based tools and is able to model, prototype and generate flexible business applications with graphical user interfaces. The generator produces a complete user interface: static structure, domain-independent and partly domain-dependent dynamic behaviour and dynamic links to domain object functionality. Adaptation to specific users or target environments is supported by standardised user and environment object classes. The generated prototype can directly be executed as an application under the target environment (e.g. MS-Windows).

2.1 Design Automation for Interactive Systems

It is a goal of AME to combine the ease of use of model-based generators with the design flexibility of model-based specification systems. As soon as an OOA model of the problem domain is available, automatic user interface prototype generation can be started. No explicit information about the user interface has to be given in the model. Structural, behavioural and presentational user interface design knowledge is available to the generator in the form of design methods and forward-chained rules.

It is not always possible for AME´s generator tools to find optimal mappings from a given domain object pattern (e.g., [Coad92]) to a group of implementation-level objects with associated behaviour. Therefore, the designer may introduce additional specifications at a later stage of the supported software life cycle. Such specifications may concern the mapping of domain object groups to interaction objects, object behaviour, user interface presentation and style, as well as user- or environment-specific features of the user interface. Additional information of this kind may lead to improved system performance or enhanced usability.

The user interface, however, is only one side of an interactive application. It was also a goal to support the concurrent evolution of the non-interactive problem domain components during all life cycle activities. When using an object-oriented modelling approach, a model of an application can be seen as a set of object clusters or groups with clear data-interface definitions between the groups. A group may contain classes for the interactive parts of the system and/or domain problem classes (e.g. for application functionality, database access, distributed object environments etc.). Communication between groups has to be managed by specific ob-

jects. AME´s OOA tools support standard object modelling methodologies [Coad91a, Rumbaugh91] and their grouping concepts.

An OOA model may be passed to AME´s generator tools. OOA classes, whose features and inter-class relations are exploitable for user interface generation, will then first be mapped to generated design patterns (OOD) and later to implementation classes (OO-language). Appropriate abstract interaction objects are selected and assigned to these classes.

Before code generation, existing method source code is embedded into the generated class structures. Dummy calls are generated, if no method code for a domain or user interface class operation is available or can be generated. Although it was no explicit goal of AME to support method code generation for other than user interface functionality, external CASE-tools for these purposes can be embedded into the environment.

2.2 AME Architecture Levels

AME is organised in three levels, as shown in figure 2 : *modelling level, construction level* and *implementation level.*

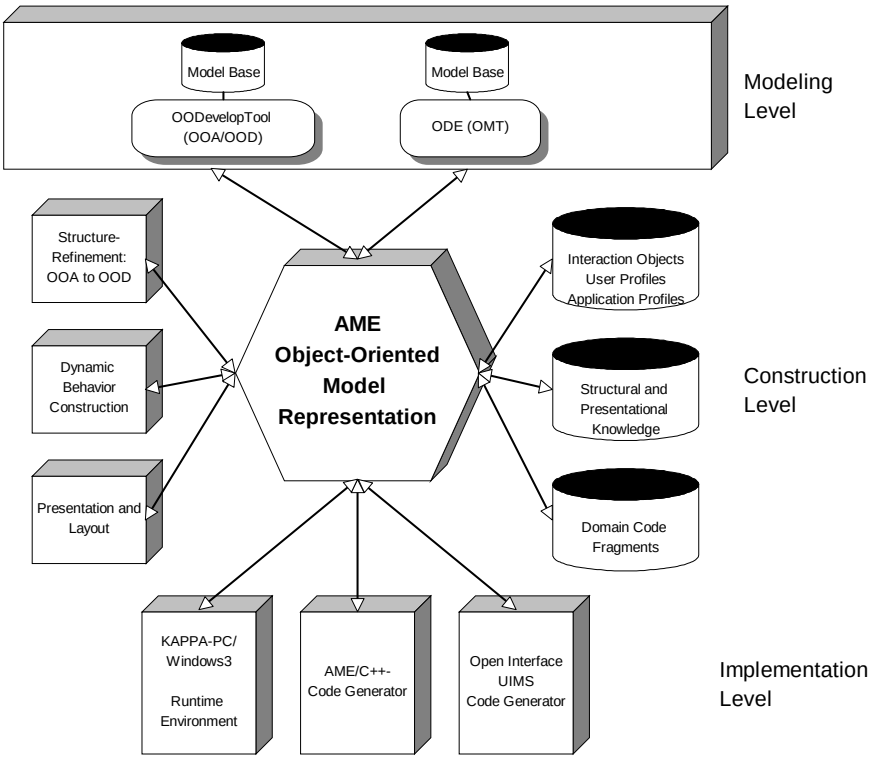

Figure 2. AME architecture

AME was developed under MS-Windows. The modelling tools and generator components were implemented using KAPPA-PC by Intellicorp, Borland C++ and Microsoft Visual C++. At the *modelling level* AME offers tools for object-oriented analysis and design. OODEVELOPTOOL is a comprehensive CASE tool that supports the modelling elements of the OOA/OOD-method by Coad and Yourdon [Coad91a, Coad91b].

It provides OOA/OOD class-, attribute- and method-editors, OOD-support for user interface design, pre- and postconditioning [Meyer88], a design versioning system and software documentation mechanisms. ODE is a compact tool for creating OOA object models using OMT [Rumbaugh91] plus a message-based dynamic link notation for specifying OOA-level dynamics. Both tools can also be used as stand-alone components. Models can be translated into AME´s internal object representation and passed to the construction level.

At the *construction level,* models are refined into detailed design specifications by a series of automatic design steps for OOD-structure generation, interaction object selection, dynamic behaviour construction, presentation and layout design. This level also provides the functionality for adapting application models to specific target environment (e.g., MS-Windows 3.x) and individual users. Section 3 discusses the construction level in more detail. Model features, generated by any of the construction level components, may be modified interactively by the designer. The final design specification is passed to the implementation level.

AME´s *implementation level* offers different ways for generating runtime applications. A C++-code generator translates static and dynamic parts of the user interface specification into C++-source code. It also embeds domain methods into the generated C++ implementation classes. The code is compiled into a Windows application by the Borland C++-compiler. Other AME-tools generate UIMS-code for Open Interface, KAPPA-PC runtime applications from the specification model.

2.3 Representing Application Models and Knowledge

AME supports the following knowledge types: application models, user interface design knowledge and adaptation knowledge.

2.3.1 Application Model

The scheme for representing the application model in all its development states has to meet the following requirements:

- Representation of classes and all typical intra- and inter-class modelling elements used in OOA- and OOD-methods.
- Representation of all structural, functional and dynamic features of the model during its transition from a very abstract analysis model to a rather concrete design specification.

- Representation of all generated or designer-specified components of the UI with their structural, presentational and layout properties.

To provide the required expressiveness and to keep the formalism simple, an object-oriented representation scheme built on top of the frame-like weak-typing approach of the KAPPA-PC environment was defined. Frame-based representation schemes were already used in earlier user interface generators [Wiecha89, Märtin90].

AME introduces the class Application System Object (ASO) for representing any OOA or OOD class during the development process. An ASO-object offers about 50 different attributes (slots) for representing the structural and semantic properties of the application model objects during their lifetime (figure 3).

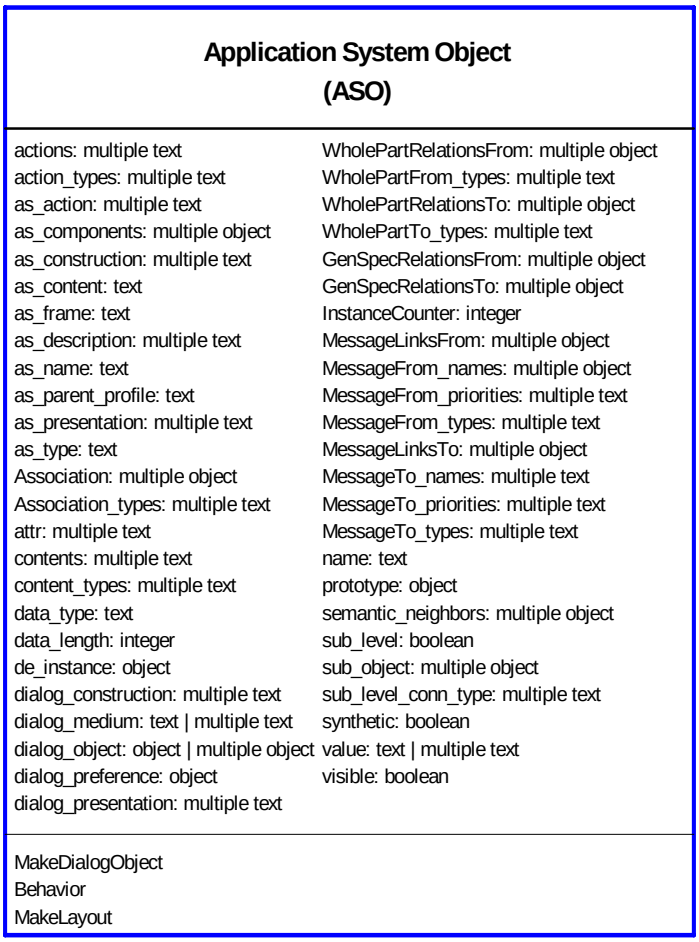

Figure 3. ASO class structure. Some attribute values (e.g. content_types, action_types*) are internally specified in more detail to be exploitable by generator components*

After OOA-modelling, every class is mapped to an ASO-object. As each OOA-class may have its own number of attributes, methods and relations to other model classes, OOA attribute-, method- and relation-specifications are mapped to the entries of specific list-valued ASO-slots *(contents, content_types, actions, action_types, WholePartRelationsTo, WholePartTo_types* etc.).

Most ASO-slots, however, are not specified by OOA tools. They are filled by AME components during the construction process (e.g., the slot *dialog_object* is only filled, if an appropriate abstract interaction object for the OOA class can be assigned). During the construction process all slot values may be modified dynamically.

2.3.2 User Interface Design Knowledge

AME´s design knowledge for selecting abstract interaction objects and generating the structural, dynamic, presentational and layout properties of the interactive target application is provided by methods of the construction level components and in the form of selection rules.

AME offers separate class hierarchies of abstract interaction objects, interaction media and domain-specific interface templates. These hierarchies can be exploited for abstract interaction object selection. Selected abstract interaction objects are linked to ASO objects via the *dialog_object* slots.

The presentational settings as specified by the abstract interaction objects are visualised for prototype simulation. They can be modified by the designer or by applied presentation rules.

The ASO structure may also serve as a runtime data model of the application. Communication between model objects is specified by the designer during OOA and by the system or the designer after OOD structure generation. After abstract interaction object selection, the required user interface behaviour specification is automatically extracted from the OOD model and mapped to the target system.

2.3.3 Adaptation Knowledge

Special classes are provided for representing user and environment profiles. Objects of these classes specify usability items. They may also include designer-defined rule groups, which support specific adaptation requirements for structure, presentation and layout.

3 AME Software Life Cycle

The AME software life cycle is shown in figure 4. In the following sections, the typical steps of the life cycle are illustrated for a small example application. The purpose of the application, named TRANTOOL, is to provide language translation assistance for an existing text processor.

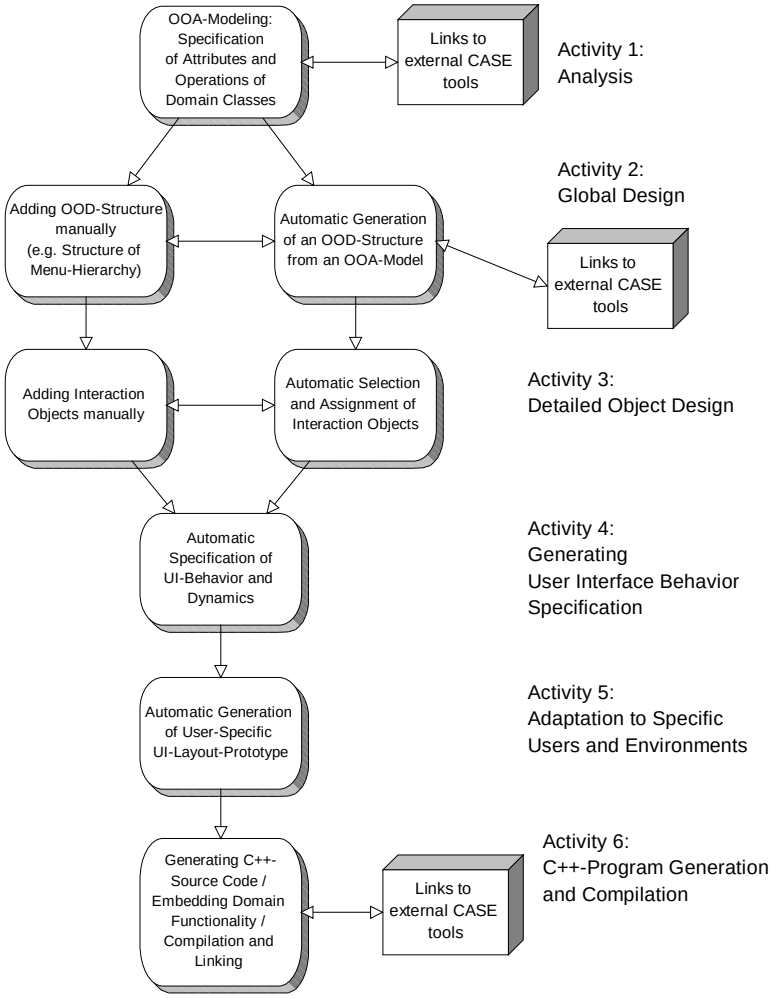

Figure 4. AME´s automated life cycle for building interactive systems

3.1 Analysis

This is the first activity of the development process supported by AME. An OOA model is created by the designer. For each domain object class the following intra-object class modelling data can be specified:

- attributes (name, data type and starting value);
- methods (name, calling parameters, data types of calling parameters, return value, return type).

The following relation types are available for connecting OOA classes:

- generalisation/specialisation (including multiple inheritance);

- aggregation (including the specification of aggregation multiplicity);
- association (including the specification of association multiplicity);
- dynamic link (including the specification of a name, message contents, a type, a priority).

This modelling information is exploited for automatic user interface construction by later life cycle activities. During OOA the designer does not explicitly specify any user interface properties. Functional specifications for OOA class methods may be provided. They can be exploited for domain method code generation by external CASE tools. The designer may choose one of the available graphical editor tools OODEVELOPTOOL [Märtin93] or ODE to specify the domain object model of the application. Figure 5 shows an ODE screen during OOA modelling of the TRANTOOL application.

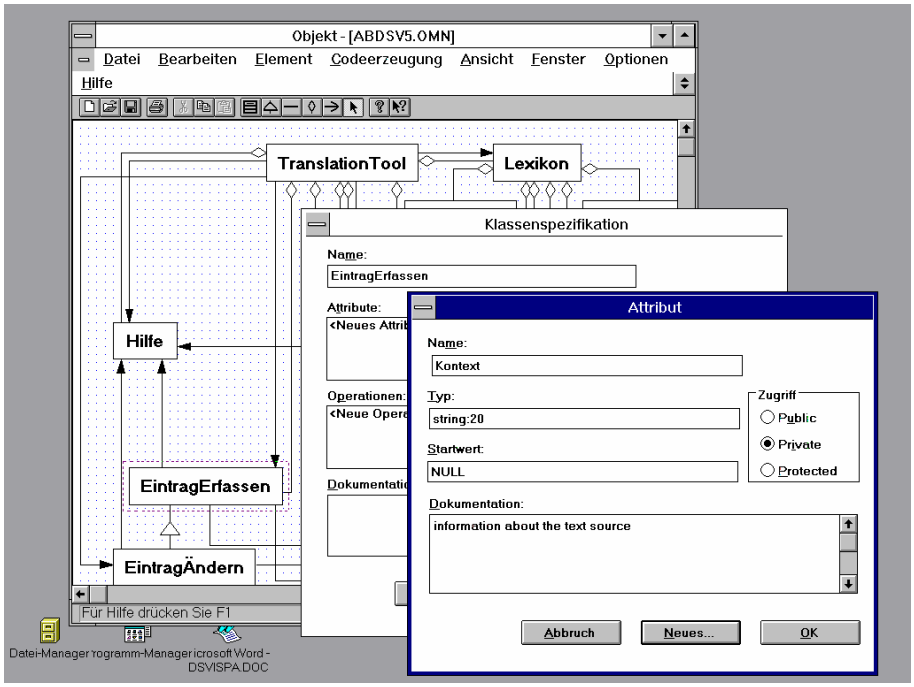

Figure 5. OOA model for Trantool designed with the ODE-editor

An OOA model can be translated into an internal AME representation by selecting the item *Kappa* from the menu *Codeerzeugung (code generation)*. Thus, for each OOA class, an ASO object is instanciated. Attributes are mapped to *contents*-slot of the ASO object as a value list. Attribute types are mapped to a list of values for the *content_types*-slot. *Content*- and *content_types*-entries with the same index belong together. Methods and their types are mapped to the slots *actions* and *action_types* in a similar way. Inter-class relations are translated into directed pointers between ASO objects.

3.2 Global Design

This activity defines the object-oriented design structure of the application. The representation of the OOA model is expanded to an OOD model, which includes the window structure and the menu and command hierarchy of the application. OOA classes with multiple exploitable attributes are mapped to patterns of related OOD classes. This task is accomplished automatically by construction level components. Additional manual design makes sense, whenever domain-dependent decisions concerning the user interface structure, which are based on information not available to the system, have to be taken. Such decisions may include the assignment of one or a group of specific interaction objects to a particular domain object or the command or menu representation of a domain method in the user interface. If the generated OOD-structure needs some modifications for efficiency or usability reasons, the designer may also modify the OOD model.

To provide global design automatically the system needs some basic information about the target runtime environment at this early stage. The AME prototype uses structural knowledge about the MS-Windows 3.11 environment (e.g., standard menus *File, Edit, Help, View* etc., which appear in typical applications, standard menu items and their synonyms). Textual pattern matching techniques are used to map the method names of OOA classes to the synonymous items of standard- or application-specific pull down-menus in the Windows environment. It is not easy to automate this task, because standard Windows applications provide pull down-menus only for the main window of an application. Therefore, the matching algorithm has to know which OOA class will be mapped to which window type *(main window, window* or *dialogue box).*

For this and other structural purposes an object parser is provided. It examines and exploits the generalisation/specialisation, association and aggregation structure of the OOA domain model and the internal features of each OOA class. Each attribute or method can only be inherited once. The parser automatically assigns a window type to each complex object, i.e., each OOA class with aggregated classes or multiple exploitable attributes.

The class at the top of the aggregation hierarchy or the topmost non-generic class in the generalisation/specialisation hierarchy is mapped to the application main window. If more candidate windows exist, the designer has to choose the main window. Other complex objects are mapped to dialogue boxes, if they contain *Cancel* and *OK* methods (name synonyms are accepted) or to ordinary windows, if not. The multiplicity-value of aggregations is used to specify whether one or more instances of this window class may be created at the same time.

Methods that belong to *main window* or *window* objects are mapped to pull down menu entries of the main window. Window methods for which no synonyms can be found are mapped to an application-specific pull down menu. If the OOA classes contain methods for standard services (e.g., *Print, Find, Replace, Open, Close),*

these methods are mapped to the corresponding menu items and linked with standardised ASO objects, which represent the related dialogue box or default action.

To resolve name collisions between methods a menu entry is only generated for the method that belongs to one window: the one which is itself the main window or the nearest one to the main window. A button is assigned to the other method(s). Dialogue box methods are always mapped to buttons or button groups. At this life cycle step, neither real menus nor buttons, but only ASO representation objects with the appropriate slot settings are created. For each OOA method, which could be mapped to a menu action or a button, a dynamic link to the OOA object is generated. At a later stage, the code generator exploits these links and creates code for calling the method, whenever the menu entry or button is selected.

During global design each *complex* OOA class (e.g., a class representing a data entry form for a multilingual dictionary) has to be expanded to an aggregation of many (typically dozens or hundreds) simple OOD-classes. Each *exploitable* class attribute is mapped to one or more OOD-classes, representing one interaction object for some simple component of the entry form (e.g., a list box for selecting the target language). Each generated OOD class is linked to its origin class by an aggregation relation. For grouping attributes special *content_types* settings are available for the designer. *Simple* OOA classes (with only one content value or attribute) are directly adopted as OOD classes.

To be *exploitable* an attribute needs a data type, known to the system and some qualifying meta information. These data are used for mapping each attribute and its contents to an abstract interaction object. The attribute *Language* of an OOA object *Language Environment* with a content_type *string:20,* for example, is mapped to two ASO objects, which represent a label with the value *Language* and an edit field of *length 20*. The values of the *dialog_object* slots, which specify the AIO, are set to *Static* and *Edit*. Two aggregations from the ASO representing *Language Environment* to the new ASO objects are generated. To facilitate layout generation the new ASO objects are connected by associations.

3.3 Detailed Object Design

During this life cycle activity the system selects abstract interaction objects for all OOD classes. AME uses similar selection techniques as the systems in [de Baar92, Vanderdonckt93]. Attribute data types, cardinalities and some meta information are evaluated for this purpose. To find interaction objects for method activations the calling parameters and return types of the methods are evaluated.

Specific abstract interaction object types *(Function, Code, Event)* support the integration of domain functionality, code fragments or event based user interface dynamics. OOD classes, whose *dialog_object* was already specified during global design, are revisited during detailed design. In some cases abstract interaction objects are refined to more specific types (e.g., from a group of single *buttons* to a *button group).* The knowledge for selecting abstract interaction objects can be expressed in the

form of rules. For efficiency reasons, these rules are coded as *if-then-else* cascades in a global resource method. The method *MakeDialogObject,* which was inherited by each ASO, calls the resource method for choosing the interaction object type. To make the selection process more flexible, a great number of data type synonyms is known to the system. A designer can easily change the generated interaction object assignments.

3.4 Automatic Specification of Dialogue Behaviour and Dynamics

The remaining construction level components map OOD object features to interaction object features *(behaviour mapping)* and build the specification for the dynamic properties of the entire interactive system.

Each ASO object owns the same common *Behaviour* method. For each abstract interaction object a *specific* Behaviour method (e.g., *ComboBoxBehavior*) is provided. After an abstract interaction object was assigned to an ASO, the specific method is activated by *Behaviour*. It specifies how the contents of the relevant ASO slots should be mapped to the features of this specific interaction object type. The specification information is written to reserved ASO slots.

In the target environment, the C++-code generator uses this information for creating concrete interaction object classes with correct interactive properties. The behaviour mapping process also provides information needed to generate menu activations, external application calls and code for embedding domain objects, which encapsulate event handlers or application code fragments. For generating these control specifications ASO-*actions* with specific *action_types (e.g., Create, Delete, Activate)* have to be evaluated together with the dynamic embedding structure of their ASO objects (see below).

In pure object-oriented systems inter-object communication is specified by messages (dynamic links) between classes or objects. AME allows the specification of dynamic links between OOA classes. During global design, additional dynamic links are generated between each OOA class, representing a window, and all OOD classes, whose interaction objects (including menus) can dynamically be referenced by this window at runtime. The configuration of these dynamic links guides the C++-code generation for window activation and deactivation. Inter-class method calls are also modelled by typed dynamic links between OOA or OOD classes. Message based specification and generation of interactive dynamics in AME is discussed in more detail in [Märtin95].

An additional dynamics tool [Schmalzbauer95] is currently being integrated into the AME environment. This detailed design level tool allows the specification and generation of message based domain-dependent dynamics for MS-Windows platforms (e.g. the time- and situation dependent change of the appearance of graphical application objects or the availability of menu-entries, if a condition evaluates to *True)*. The tool allows the modelling of state-dependent conditions that control the

dynamic behaviour, the specification of activation messages between OOD objects and the source code specification of the message handling methods.

OOD attributes may be used as state variables. At runtime state changes (e.g. *below value, above value, changed, exact value* or more complex conditions involving multiple attributes) are watched by generated daemons. An extension to the C++ code generator exploits these specifications to generate the method implementations.

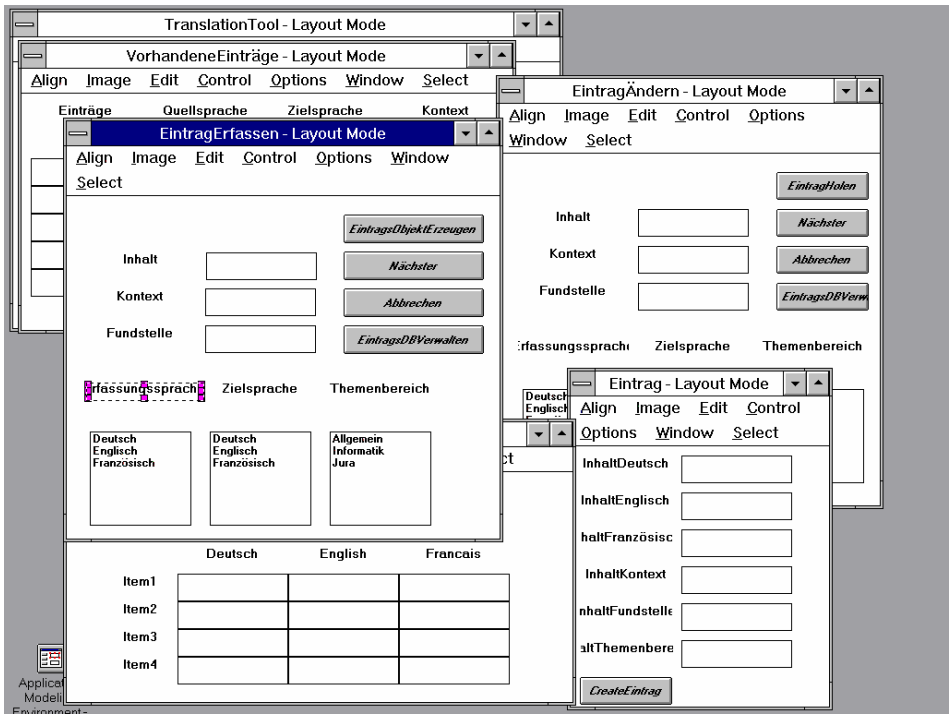

Figure 6. Simulated Trantool user interface in interactive layout mode

3.5 Layout and Style Generation

To provide a realistic simulation of the application before code generation, prototypical instances of all specified interaction objects are created. Instances inherit the look-and-feel characteristics from AME´s interaction object class-hierarchy.

A specific layout-method *(MakeLayout)* is assigned and adapted to each OOD object that represents a window or a dialogue box. The window types and their layout methods are selected by counting the number of each interaction object type in a window. A set of window and layout classes and their layout routines were chosen as AME standard resources by evaluating and comparing the window and dialogue box types of existing commercial MS-Windows applications. A simplified example for selecting a layout type for a dialogue box *x* is shown in the following:

```
If (x:number(ComplexInteractionObjects) > 0) /*e.g. spreadsheets*/
Then x:layout_type := linear
Else If (x:number(Edit) > 4) Or (x:number(Editor) > 4)
      Then x:layout_type := entry_mask
      Else If (x:number(Edit) > 0)
           Then x:layout_type := entry_dialog
           Else If (x:number(Listbox) > 0) Or (x:number(Combobox) > 0)
                Then x:layout_type := listbox_dialog
                Else x:layout_type := message_box.
```

The layout generator also evaluates the association relations specified between OOD classes to find semantically linked interaction objects. To facilitate layout generation, each window or dialogue box is divided into rectangular areas. Each resource layout type (e.g., *entry_dialog)* defines in which rectangle instances of a certain interaction object type typically appear. The detailed design (spaces between elements, row and column ordering, width and height of the window) depends on the actual number of each element of a given type.

A preview of the layout of all windows is created by activating the layout methods. Presentational settings like colours, fonts or sizes are inherited from AME´s interaction object resource hierarchy and can be changed by the designer. The user interface specifications in the layout prototype are still independent of a specific GUI platform. A designer can also add application- or user-specific presentation and layout rules to the environment- and user-profile. Such rules are activated in a forward-chained mode. Figure 6 shows the first simulation of the TRANTOOL user interface. The designer may change the generated layout and presentation. Any changes will be stored and passed to the target code generator.

3.6 Target Source Code Generation

Finally, the detailed design model that includes the specification of structure, dynamics, layout and presentation of the interactive system can be passed to the *implementation level*. A code generator at this level exploits the design model to create C++-source code. The source code can be translated by a Borland C++ compiler and automatically linked with domain method code. At the generator level, the designer still may modify the specification, before it is parsed and translated into source code. To support different target platforms, OOD specifications of applications can be translated into *Open Interface* UIMS code.

Conclusion

To compare the AME design process with established conventional approaches several application prototypes were developed, including a spreadsheet application and a simple accounting system. As the system is still developing new application projects typically require some new interaction object classes and additional construction knowledge. The integration of new resources and design knowledge is a

relatively easy task. Once integrated the new functionality can be used by the system like any other standard resources. This learning process turned out to be quite efficient, as most parts of the design knowledge are implemented as method-code. Many consistency problems of earlier rule-based generators could be avoided.

Naturally, our approach does not offer solutions to all possible productivity problems. However, design time can be drastically reduced for those application design situations, where the existing design knowledge can be applied for generating the standard parts of the application. AME does not take the domain modelling task from the designer. If AME´s design resources fit the application domain, however, an OOA model will be automatically expanded into an OOD model with possibly several hundreds of ASO objects and their interaction objects.

Without programming, the resulting application provides correct interaction object mappings, a raw layout, presentation and style attributes, links to all domain code methods, the menu hierarchy, application-independent interactive dynamics and part of the application-dependent dynamics.

Acknowledgements

The author would like to thank Johann S. Kempfle, Michael Schmalzbauer, Axel Struwe, Christian Winterhalder and all others, who did their diploma thesis work with the AME project, for their contributions.

References

[Balzert93] Balzert, H.*, Der JANUS-Dialogexperte: Vom Fachkonzept zur Dialogstruktur*, in Softwaretechnik Trends, Band 13, Heft 3, Proceedings der GI-Fachtagung Softwaretechnik, Dortmund (8-10 November 1993), pp. 62-72.

[Balzert94] Balzert, H., *Das JANUS-System: Automatisierte, wissensbasierte Generierung von Mensch-Computer-Schnittstellen,* in Informatik-Forschung Entwicklung, Vol. 9, Springer-Verlag, Heidelberg, 1994, pp. 22-35.

[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 this volume, pp. 141-158.

[Coad91a] Coad, P., Yourdon, E., *Object-Oriented Analysis*, Prentice-Hall, 1991.

[Coad91b] Coad, P., Yourdon, E., *Object-Oriented Design*, Prentice-Hall, 1991.

[Coad92] Coad, P., *Object-Oriented Patterns*, Communications of the ACM, Vol. 35, No. 9, September 1992, pp. 152-159.

[de Baar92] de Baar, D.J.M.J., Foley, J., Mullet, K.E., *Coupling Application Design and User Interface Design*, in Proceedings of the Conference on Human Factors in Computing Systems CHI'92 « Striking a balance » (Monterey, 3-7 May 1992), P. Bauersfeld, J. Bennett, G. Lynch (Eds.), ACM Press, New York, 1992, pp. 259-266. ftp://ftp.gvu.gatech.edu/pub/gvu/ tech-reports/91-10.ps.Z.

[Elwert94] Elwert, T., Forbrig, P., Schlungbaum, E., *Meta Models for Task-oriented User Interface Development*, in Proceedings of the 1st Workshop on Cognitive Modelling and Interface Development (Vienna, 15-17 December 1994), pp. 163-172.

[Fischer93] Fischer, G., Nakakoji, K., Ostwald, J., Stahl, G., Sumner, T., *Embedding Computer-Based Critics in the Context of Design*, 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. 157-164.

[Forbrig96] Forbrig, P., Märtin, C., *Automatisierte Entwicklung interaktiver Software: Spezifikation, Generierung*, CASE-Integration, Offene Systeme, Vol. 5, No. 1, 1996, pp. 11-25.

[Goldberg84] Goldberg, A., *Smalltalk 80. The Interactive Programming Environment*, Addison-Wesley, 1984

[Gorny94] Gorny, P. et al., *Projekt EXPOSE, Expertensystem zur phasenorientierten Software-Ergonomie-Beratung bei der Benutzerschnittstellen-Entwicklung*, 2. Zwischenbericht, Universität Oldenburg und Universität Rostock, 1994.

[Gorny95] Gorny, P., *EXPOSE - An HCI-Counseling for User Interface Design*, in [Interact95], pp. 297-304.

[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.

[Kim93] Kim, W.C., Foley, J.D., *Providing High-level Control and Expert Assistance in the User Interface Presentation Design*, 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. 430-437.

[Lonczewski96] Lonczewski, F., Schreiber, S., *The FUSE-System: an Integrated User Interface Design Environment*, in this volume, pp. 37-56. ftp://hpeick7.informatik. tu-muenchen.de/pub/papers/sis/fuse_cadui96.ps.gz

[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

[Märtin90] Märtin, C., *A UIMS for Knowledge Based Interface Template Generation and Interaction*, in Proceedings of the 3rd IFIP TC13 Conference on Human-Computer Interaction INTERACT'90, Cambridge, 27-31 August 1990, D. Diaper, D. Gilmore, G. Cockton and B. Shackel (Eds.), Elsevier Science Publishers, Amsterdam, 1990, pp. 651-657.

[Märtin93] Märtin, C., Winterhalder, C., *Integrating CASE and UIMS for Automatic Software Construction*, in [HCIint93], pp. 291-296.

[Märtin95] Märtin, C., *Generating the Dynamic Behavior of Interactive Applications from High-Level Object-Oriented Models*, in Proceedings of the International Conference on Industry, Engineering and Management Systems IEMS´95 (Cocoa Beach, 1995), G.C. Lee (Ed.), Univ. of Central Florida, 1995, pp. 180-185.

[Märtin96a] Märtin, Ch., *Modellierung, Entwurf und automatische Konstruktion interaktiver Softwaresysteme*, Entwurf der modellbasierten Entwicklungsumgebung Application Modeling Environment (AME), Ph.D. thesis, University of Rostock, 1996.

[Meyer88] Meyer, B., *Object-Oriented Software Construction*, Prentice Hall, Englewood Cliffs, 1988.

[Meyer95] Meyer, B., *Object Success*, Prentice Hall, Englewood Cliffs, 1995.

[Monarchi92] Monarchi, D.E., Puhr, G.I., *A Research Typology for Object-Oriented Analysis and Design*, Communications of the ACM, Vol. 35, No. 9, September 1992, pp. 35-47.

[Puerta96b] Puerta, A., *The MECANO Project: Comprehensive and Integrated Support for-Model-Based Interface Development*, in this volume, pp. 19-35.

[Reiterer94] Reiterer, H., *User Interface Evaluation and Design*, GMD-Report No. 237, Oldenbourg, 1994.

[Reiterer95] Reiterer, H. *IDA – A Design Environment for Ergonomic User Interfaces*, 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. 305-310.

[Rumbaugh91] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W., *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, 1991.

[Schmalzbauer95] Schmalzbauer, M., *Generierung der Dynamik interaktiver Anwendungen aus abstrakten Objektmodellen unter Windows*, Diploma Thesis, Fachhochschule Augsburg, Fachbereich Informatik, October 1995.

[Schlungbaum96] Schlungbaum, E., Elwert, T., *Automatic User Interface Generation from Declarative Models*, in this volume, pp. 3-18. http://www.informatik.uni-rostock.de/~schlung/TADEUS/paper/CADUI96.html

[Sukaviriya93] Sukaviriya, P., Foley, J.D., Griffith, T.*, A Second Generation User Interface Design Environment: The Model and the Runtime Architecture*, 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. 375-382

[Szekely93] Szekely, P., Luo, P., Neches, R., *Beyond Interface Builders: Model-Based Interface Tools*, 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. 383-390. http://www.isi.edu/isd/Interchi-be-yond.ps

[Vanderdonckt93] Vanderdonckt, J., Bodart, F., *Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection*, 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. 424-429. http://www.info.fundp.ac.be/cgi-bin/pub-spec-paper?RP-93-005

[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.