Designing and Using Human–Computer Interfaces and Knowledge Based Systems
edited by G. Salvendy and M. J. Smith
Elsevier Science Publishers B.V., Amsterdam, 1989 – Printed in the Netherlands

USIT: A TOOLKIT FOR USER INTERFACE TOOLKITS

MICHAEL HERCZEG
ANT Telecommunications, D-7150 Backnang, Fed. Rep. of Germany

SUMMARY

This paper describes the USIT user interface toolkit. USIT serves to create specialized user interface toolkits. It enables a user interface programmer to represent user and application dependent interaction methods as user interface building blocks. The basic methods to perform this are specification, specialization, and aggregation of user interface building blocks. A large collection of predefined blocks is available to build standard user interfaces. With the aid of the USIT Metasystem, an already running user interface may be changed interactively without programming skills.

1 INTRODUCTION

Building user interfaces is afflicted with some characteristic problems (cf. [Löwgren 88]):

- User and application requirements for user interfaces vary heavily during and after construction of application systems.
- I/O-hardware often changes faster than the application software. Standards usually are between 5 and 10 years behind technological feasibility.
- Currently, there are no means for formal specification and automatic generation of complete user interfaces for non-trivial applications.
- Existing user interface toolkits are either too low level or too restricted to be really helpful. They have been built as generic tools to be used for any kind of user interface and are therefore lacking support for application dependent interaction styles.
- Most user interfaces cannot be changed interactively at runtime to explore design alternatives. After being completed by the programmer, they loose their software characteristics and are like hardware for the user.

As a result, many user interfaces contructed are inadequate from application programmers' as well as from end users' point of view [Herczeg 86].

End users much too often have to cope with interaction methods that are inadequate for their special application problems and either highly inconsistent or monotonous in use and appearance like many menu- and form-based user interfaces.

Programmers are virtually unable to make substantial changes to user interfaces within acceptable time. This is especially true for user interfaces providing complex interaction methods like direct manipulation.

A break-through in user interface design emerged in the late 70s with systems built at Xerox PARC. Well known examples are the Smalltalk environment [Goldberg 84] and the Xerox Star [Smith et al. 82]. Both demonstrate the direct manipulation paradigm with a rich user interface and both are based on a user interface toolkit.

2 REQUIREMENTS

User interface toolkits are still the most promising approach for advanced user interface development. Unfortunately, even existing prototypical toolkits do not provide all of the expected and necessary support for the programmer.

What is needed, are user interface toolkits with the following properties:

Modularization: User interface building blocks (UIBBs) should be constructed as encapsulated software modules with clearly defined programming interfaces which are adequate for the applications.

Specialization: Existing UIBBs should be specializable in order to alter their appearance and behavior to some extent. This is the first way of tailoring the toolkit for special applications.

Aggregation: Existing UIBBs should be easily aggregateable to more complex ones, which should be usable as high level building blocks themselves without showing all of their complex interior. This is the second way of tailoring the toolkit to special application needs.

Extension: A user interface toolkit should be designed as an open system, providing tools to build new UIBBs that cannot be built out of already existing ones by specialization or aggregation. This is the third way of customizing the initial toolkit for special classes of applications.

Generality: A broad spectrum ranging from traditional menu- and form-oriented to graphic-based, direct manipulation interaction methods has to be supported by the initial toolkit.

3 THE USIT SYSTEM

During the past 6 years at the University of Stuttgart, we developed the user interface toolkit USIT as a part of the user interface management system INFUIMS [Herczeg 87,Herczeg 88], trying to fulfil the requirements mentioned above to a high degree. Like many other user interface toolkits, USIT has been implemented in an object-oriented programming language (cf. [Lipkie et al. 82,Goldberg 84,Sibert et al. 86,Symbolics 86]. Some of the requirements are considerably supported by the object-oriented programming paradigm:

- It is quite natural to describe UIBBs as objects. The parameters of an interaction method are represented as slots and the UIBB functionality is modelled as methods.

- Communication between objects is exclusively performed via message passing. The interface definitions are the method filters, which can be inspected and modified easily.
- Classes describe sets of similar objects by specifying their attributes and methods. They serve as UIBB generators. Class or pool slots allow to describe or change the behavior or appearance of all of the class' instances.
- Classes may be linked into an inheritance lattice or hierarchy. Through this, classes inherit slots and methods from their superclasses. These inherited descriptions may be specialized and enriched by new definitions. This is a way of describing more or less special UIBBs with low redundancy.
- Instances may be connected by relations in order to build object nets. This is done when complex aggregates are built out of single objects or object subnets.

USIT supports the creation of user interfaces in several ways. It provides three basic mechanisms: the *USIT-Toolkit*, the *USIT-Workbench*, and the *USIT-Metasystem*.

Figure 1 shows an example of a direct manipulation interface built with USIT.

3.1 USIT as a Toolkit

There is a large set of standard UIBB classes, which may be directly used to build the user interface for an application. This is done by instantiating predefined UIBB classes and connecting them to the application system. Subclasses of the UIBB classes can easily be defined to introduce a more specialized behavior or appearance for an interaction method.

Predefined UIBBs in the initial toolkit are textareas, icons, buttons, switches, menus, forms, property-sheets and tables, each with a variety of adjustable parameters concerning contents, size, position, layout and interaction behavior.

3.2 USIT as a Workbench

We provided a basic toolkit, called the *USIT Workbench*, with elementary user interface building blocks (EUIBBs) to create UIBBs. This is a toolkit to build application and user dependent user interface toolkits. Such EUIBBs are:

Presentation-Objects: representations of visible or audible user interface particles like texts, bitmaps, rectangles, lines, and sounds

Layouters: controllers for the screen positions of sets of display-objects (visible presentation-objects)

Syntax-Descriptions: rule-oriented event-response descriptions to specify the behavior of UIBBs

Display-Areas: light-weight windows serving as output areas for display-objects

Readers: objects reading from the keyboard, building high level tokens from low level character streams, and performing arbitrary actions after having created such tokens

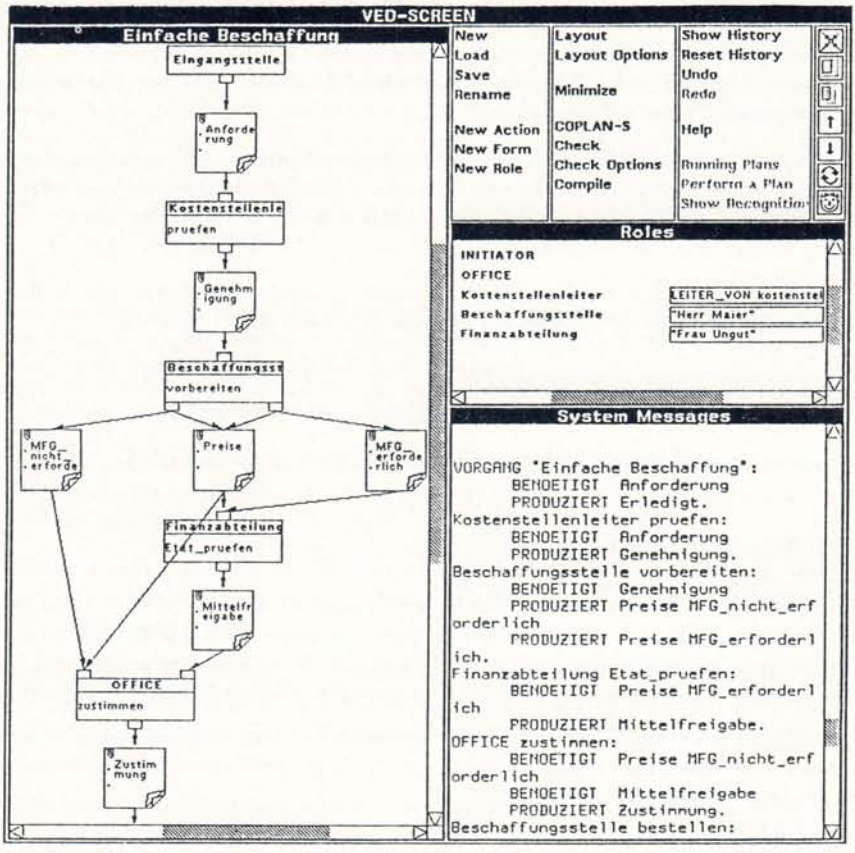

Figure 1: A user interface for an office procedure editor built with USIT

Fonts, Bitmaps, Sounds: functional objects producing screen images or acoustic output for presentation-objects

Interaction-Objects: shells around aggregation nets of UIBBs representing special interaction methods

There are generic classes (UIBB prototype classes) that may be subclassed to build specialized UIBBs. They serve as *templates* to define new interaction methods. Metaclasses define convenient class definition schemas and therefore a kind of user interface specification language.

Below is an example of the definition of an interaction-object defining a complex kind of icon. Only slight syntactic changes have been made to enhance readability. Text in bold face denotes other UIBBs or EUIBBs. The appearance of such icons on the screen is shown in Figure 1.

Figure 2 shows the object net for the defined application-specific icon.

```
(user-interface-class ved-action-icon
  (superclass icon)
  (slots
    (in-connections
      (part connector))
    (out-connections
      (part connector))
    (actor-name
      (part text-area
        (size = (96 12))
        (border? = t)
        (font = helvetica-8-bold)
        (adjust-text-vertical = center)
        (adjust-text-horizontal = center)))
    (action-name
      (part text-area
        (size = (96 24))
        (border? = t)
        (reactivity = (((mouse left) . start-editor)
                       ((keyboard end) . tell-view-of))))
        (shading-when-inactive = nil)
        (mouse-feedback-when-active = blinker)
        (active-mouse-blinker = cross)
        (interline-spacing = 1.0)
        (font = helvetica-8)
        (adjust-text-vertical = center)
        (adjust-text-horizontal = left)))
    (layouter
      (layout distance-cluster
        (orientation = down)
        (distance = 1)))
    (layout-sequence
      (default (in-connections actor-name action-name out-connections)))
    (pop-up-part
      (default net-action-menu))
    (reactivity = (((mouse left) . tell-view-of)
                   ((mouse right) . pop-up-part)))
    (view of
      (class coplan-action))))
```

3.3 USIT as a Metasystem

Even after a user interface for an application has been instantiated, there are tools to adapt the UIBBs according to new user or application requirements.

We call these tools the *USIT Metasystem*. It turns a USIT user interface into an adaptable system for the programmer and in a more restricted way even for the end user [Herczeg/Böcker 87]. In the current implementation, the metasystem is basically shaped as a rapid prototyping tool for the programmer to create user interface design alternatives.

The current USIT Metasystem provides tools to redefine the following properties of an existing, running user interface:

- the visibility of UIBBs relative to the visibility of their aggregating UIBBs
- the window-size of UIBBs
- the screen layout of groups of UIBBs
  (one of the property-sheets to change the layout is shown in Figure 3)
- the syntactic behavior of UIBBs in respect to mouse and keyboard interactions
- the semantic behavior of UIBBs in respect to the underlying application

These properties of user interfaces can be changed interactively to adapt to a special user or application. The USIT Metasystem has been built exclusively by means of USIT itself and can therefore be recursively applied to itself to change its own appearance and behavior.

4 CONCLUSIONS

Through the fine-grained architecture of USIT, it is easy to design interaction methods with properties varying smoothly in many details according to special needs of the application or the user, without being forced to use the low level functionality of a programming language and a graphics package. This results in an *interaction continuum* rather than in isolated interaction methods. Nevertheless, it is quite easy to predefine any of the common standard interaction methods like icons, menus, and forms.

For the resulting user interface building blocks a kind of deep consistency is guaranteed by using the elementary building blocks of USIT. Additionally, this toolkit for building toolkits allows for rapid creation of new interface methods. It may either be used to support special application systems or generally for the development and evaluation of new user interface methods.

USIT has been built as the kernel of a *user interface management system* (UIMS) with other components like dialog history mechanisms [Rathke 87], user models and help systems.

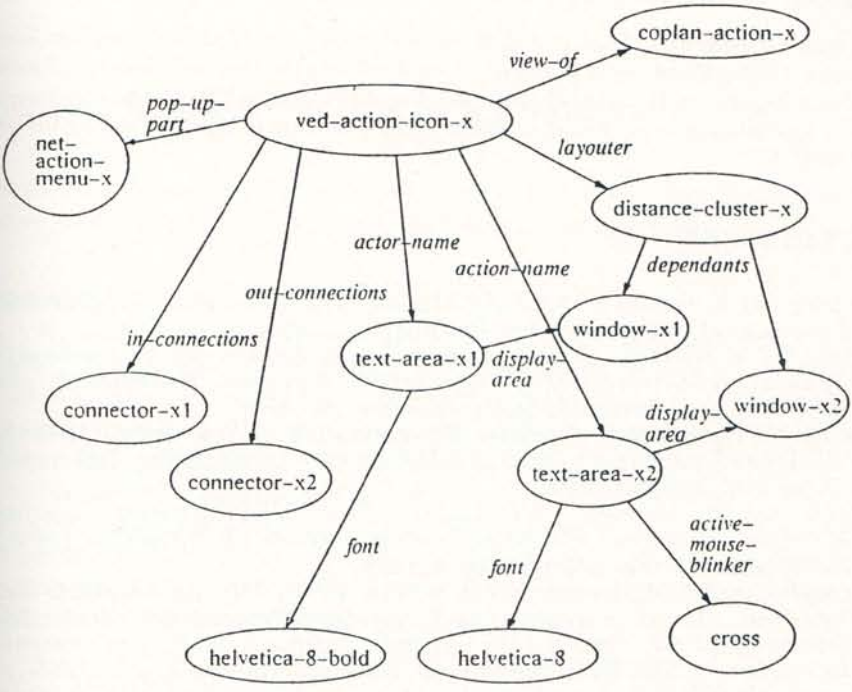

Figure 2: The object net for an iconic interaction-object

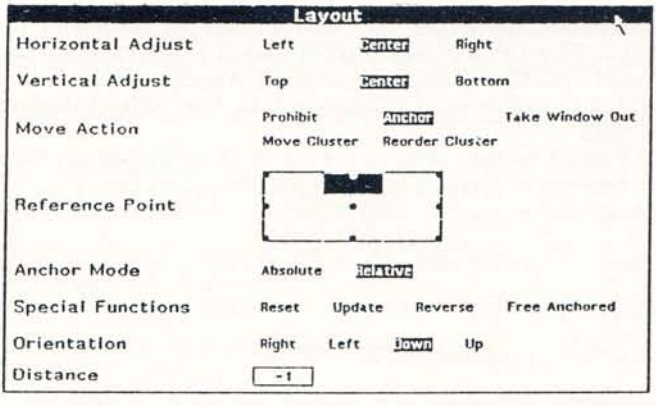

Figure 3: A property-sheet to change the layout of UIBBs

ACKNOWLEDGMENTS

This work has been part of the INFORM and WISDOM projects at the Computer Science Department of the University of Stuttgart. It was funded by the German Ministry of Research and Technology and by the Triumph Adler AG. I would like to thank for this generous support and the contributions of my former colleages at the University of Stuttgart during the years 1982-1988.

REFERENCES

[Goldberg 84] A. Goldberg. *SMALLTALK-80, The Interactive Programming Environment*. Addison-Wesley, Reading, MA, 1984.

[Herczeg 86] M. Herczeg. *Eine objektorientierte Architektur für wissensbasierte Benutzerschnittstellen*. Dissertation, Fakultät Mathematik und Informatik der Universität Stuttgart, Dezember 1986.

[Herczeg 87] M. Herczeg. *Uniform Representation of Interaction Methods*. WISDOM-Forschungsbericht FB-INF-87-30, Institut für Informatik, Universität Stuttgart, 1987.

[Herczeg 88] M. Herczeg. *INFUIMS - The INFORM User Interface Management System*, WISDOM-Forschungsbericht FB-INF-88-25, Institut für Informatik, Universität Stuttgart, 1988.

[Herczeg/Böcker 87] M. Herczeg & H.D. Böcker. DESKTOP: An Adaptable User Interface. In: G. Salvendy (Ed.), *Abridged Proceedings of the HCI International '87. Second International Conference on Human-Computer Interaction*, p. 335, HCI International, Honolulu, Hawaii, August 1987.

[Lipkie et al. 82] D.E. Lipkie et al. Star Graphics: An Object-Oriented Implementation. *Computer Graphics*, 16(3), July 1982.

[Löwgren 88] J. Löwgren. History, State and Future of User Interface Management Systems. *SIGCHI Bulletin*, 20(1):32-44, July 1988.

[Rathke 87] M. Rathke. Dialogue Issues for Interactive Recovery - An Object-Oriented Framework. In: H.-J. Bullinger, B. Shackel, & K. Kornwachs (Eds.), *Proceedings of INTERACT '87. IFIP Conference on Human-Computer Interaction*, pp. 745-750, IFIP, Amsterdam, 1987.

[Sibert et al. 86] J. Sibert et al. An Object-Oriented User Interface Management System. *ACM SIGGRAPH'86*, 20(4):259-268, August 1986.

[Smith et al. 82] D.C. Smith et al. Designing the Star User Interface. *BYTE*, 7(4), April 1982.

[Symbolics 86] Symbolics Inc. *Programming the User Interface, Volume A, B*. Symbolics Reference Manual 999025/999029, Symbolics, Inc., Cambridge, MA, 1986.