Ingrid - An Object Oriented Interface Builder

Nuno Guimarães, Luís Carriço, Pedro Antunes
IST/INESC,
R. Alves Redol, 9, 6°., 1000 Lisboa, Portugal
e-mail: nmg@inesc.pt

Abstract

After a maturing process where models and architectures for User Interface Systems have been defined and generally accepted, the current expectations of researchers, developers and users, are centered around user interface (UI) construction tools incorporating rapid prototyping facilities.

This paper describes the approach taken in the design and implementation of a system for interactive UI construction. The *INGRID* system (INteractive GRaphical Interface Designer), is based on a comprehensive UI architecture, a run time support providing interactive programming facilities in *C++*, and a set of dedicated editors incorporating direct manipulation techniques. The object oriented approach is present, both in the design of the tool itself, and in the proposed methodology for user interface development.

1 Introduction

User Interface (UI) construction is currently a clearly identified field in computer science and technology. As with any maturing technology, the concepts and models have been converging to a smaller set of alternatives [Hartson 89]. Standardization steps have been tried in the supporting technology for interactive systems and applications [Scheifler 86], [OSF 89], [Microsystems 88].

The current challenges and expectations seem to be focused on the development of User Interface Development Systems (UIDS) that provide a suitable environment for interactive construction, rapid prototyping and, ultimately, usability by non programmers.

We believe that an efficient and productive environment for the construction of interactive applications has to incorporate three orthogonal kinds of functionality, a flexible and comprehensive user interface architecture, object oriented techniques and methodology, and rapid prototyping facilities.

This paper presents an environment for building user interfaces based on a generic object oriented toolkit, a run-time support with interpretation capabilities and an interactive construction tool, called *INGRID* (INteractive GRaphical Interface Designer) [Carricco 90,Guimaraes 91].

Background and Objectives The design decisions taken in the definition of our environment were inspired by early experience in the development of the *IMAGES* User Interface Management System (UIMS) [Simoes 87], [Marques 88b], [Marques 88a]. The UIMS was based on a proprietary UI toolkit and an interface specification language. The conclusions drawn from that work and the objectives of our development were:

- To be successful in a real user community, the UIDS must integrate widely available UI toolkits like Xt [McCormack 88]. This assures conformance with the "standardization" trends and compatibility across multiple hardware and software platforms.

T. Korson, V. Vaishnavi, and B. Meyer, Eds. Fifth International Conference on the Technology of Object-Oriented Languages and Systems, TOOLS USA '91. Santa Barbara, California: Prentice Hall, Englewood Cliffs, 1991, pp. 291-300. (ISBN: 0-13-923178-1).

- The benefits of using a specification language are a larger adequacy to the models defined for the UI and a possible validation of the interface definition before its execution. However, from the interface programmer's viewpoint this is not a qualitative evolution. Instead, the objective is the development of tools that allow interactive construction of the UI and minimize the duration of the traditional programming cycle. This approach may be questionable for large software systems, where automation is a more important goal, but is certainly adequate for the small and medium sized applications of, for example, office environments, where rapid-prototyping and experimental programming is most useful.

The definition and implementation of the UI construction environment may be summarized in the following steps:

- Choose the models for application and user interface that will be presented to the user. These are outlined in section 2. The object oriented approach is an important initial decision, since it enforces a particular design and construction methodology.

- Create an environment for interactive programming. This implies a thorough evaluation of the linguistic aspects of the implementation environment. If the implementation language already offers support for interpretation (Lisp, Smalltalk) this step is avoided. If however, for the sake of openness and portability, a language like *C++* [Stroustrup 85,Ellis 90] is chosen, additional run-time support has to be built in. Section 3 focuses on the run-time support system that was developed.

- Materialize the above models in a set of interrelated software components. Encapsulation, reusability and extensibility requirements suggest that an object oriented approach is most adequate in this design. The UI components have to be well integrated with the interactive programming environment while supporting the integration of existing UI software. Our high level toolkit is described in section 4.

- Build tools for interactive construction of interfaces. Once the right UI components are available and support for their interactive manipulation is available, dedicated tools or editors may be developed. The major concern here is to find the best direct manipulation techniques and interface design that make UI construction an "easy" task. Section 5 gives a brief picture of the existing tool.

2 The models

The first decision in the design of the UI construction environment was the adoption of a set of models to present to the users. These models include:

- A global model for the application.

- A model for the user interface component.

The notion of *dialogue independence* as defined in [Hartson 89] is currently accepted by users and designers of both UIs and UIDSs. An interactive application is divided in a UI component and a computational component.

The internal model for the UI is however less stable. This model defines the different components that should be considered within a UI and their functionality. We considered that both a clear functional separation between the interface elements and an adequate design methodology were essential.

The functional model defines the UI components according to their role in the several levels of the interaction between user and application (presentation, dialogue and semantic support). This separation has been addressed by several models like Foley [Foley 82,Foley 90], Seeheim [Green 85], or the reference model [Lantz 87].

An object oriented approach, as a structuring mechanism for the design of the UI, provides the correct methodology. Current UI toolkits and UIMSs are unanimous in this issue, either by enforcing object oriented design through programming

conventions (Xt), by using extensions to procedural languages (Andrew [Neuwirth 88,Spector 88]), or through the adoption object oriented languages from the beginning (InterViews [Linton 87], ET++ [Gamma 88]).

We consider that this two-dimensional approach (functional and architectural) To the modeling of the UI is a good answer to the questions concerning design, implementation and execution of the user interface. Examples are given by MVC [Goldberg 83a] or PAC [Coutaz 87].

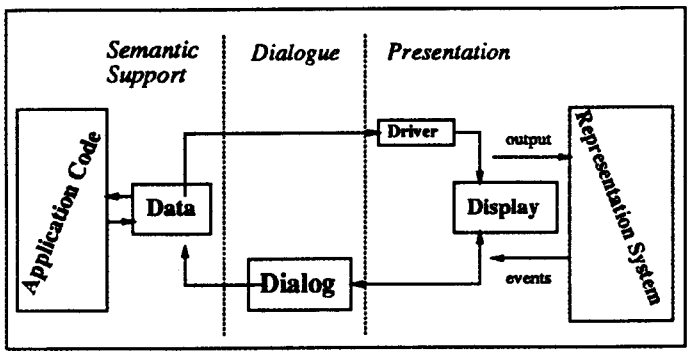

Figure 1: The 4D Model

The 4D model We called our model the *4D model*. The UI is composed of objects which class (type) belongs to one of four specific categories:

- **Display – the presentation component (input and output).**
  This category includes graphical objects such as button, menu, scrollbar, and composite *Display* objects. A composite *Display* encapsulates a collection of objects organized according to the four concepts of the model. This composition originates a recursive model [Coutaz 89] and introduces modularity in the UI. In general, *Display* objects encapsulate the representation system which can be a graphical system but also a set of multimedia devices (speech input, audio output).

- **Data – groups application abstract information.**
  It provides general purpose data structures (integer,file,list,...) with active values [Szekely 88], i.e. they automatically propagate their changes. *Data* objects provide a clear interface between UI and computational parts and introduce semantic information on the UI side, supporting semantic feedback [Dance 87]. Moreover, they may encapsulate additional facilities, like persistence or distribution, that extend the range of applications and systems that can be built.

- **Dialogue – maintains the syntactic structure of the interaction.**
  The *Dialogue* category groups dialogue control objects. The *4D* model is intentionally unexplicit about the dialogue models to allow multiple implementations of dialogue control: dialogue languages, dialogue cells, etc.

- **Driver – drives information between *Data* and *Display* components.**
  Driver objects perform data conversion (such as integer to string). Drivers introduce an extra degree of flexibility by minimizing the needs for new *Data* and *Display* objects.

Communication between objects is modeled through the **link** concept. A link is a permanent communication channel that carries messages between objects. A difference between links and method invocations is that an object, when sending a message, does not require explicit knowledge about the receivers (like their type or protocol, for example) but uses only the outgoing links. This increases the degree of flexibility and reusability by providing a well defined and autonomous behavior for a type of objects. The establishment of links between objects follows the discipline illustrated by the directions of the arrows in Figure 1.

The construction of an interface is the process of creation and composition of objects belonging to the different categories and the definition of links between them. Figure 2 shows an example of an application built according to this model. The interaction of the user with the scrollbar generates messages that are sent to the *Dialogue* object. The *Dialogue* object invokes the *Data* object (*Integer*) to update its value. Since *Data* objects have active

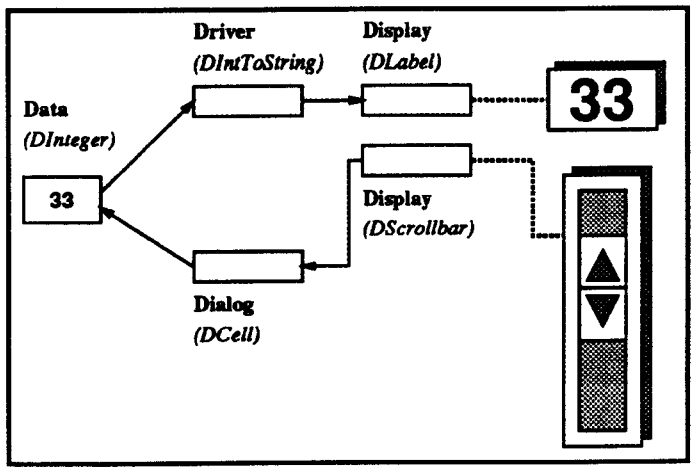

Figure 2: An Example

values, the value change propagates to the linked *Drivers*. Finally, *Drivers* convert the new value and update the *Label*.

3 The run-time support system

The interactive creation, customization, and inspection of the objects that compose a UI, requires the availability of a complete run-time description of those objects, together with interpretation capabilities.

The *ICE* (Interactive C++ Environment) library has been developed as a run-time system to support interpretation in *C++*. Its main features are:

- **Type information** with type identification, conformance and conversion testing, and knowledge about instance data and member functions.
- **Communication by message** based on the available type information, provides a standardized paradigm of communication between objects. Dynamic binding is an immediate consequence of this communication paradigm.
- **Object identification** enables the assignment of human readable names to objects, enforcing a coherent mechanism of interaction between the user and the programming entities.
- **Object storage and retrieval** again supported by the type interface description, enables transparent storage/retrieval facilities.
- **Interface uniformity** offers an abstract handling of objects, essential to ease the development of flexible and programming environments, i.e. if all objects are accessible through a common interface other types can be easily integrated without code modification.

The basic concept of *ICE* is the notion of *type-object*. It is an object that fully describes a *C++* type. A type-object is an extension of the Smalltalk's *class-object* concept [Goldberg 83b] to the *C++* basic types (char, int, float, ...), pointer types and function types. The introduction of this broader notion, enables an uniform, coherent, access to the heterogeneity of *C++* types. Type-related mechanisms like storage or retrieval are available both to user defined classes, and *C++* basic types.

To have a complete functionality, an user-defined class must have at run-time its type-object. Such an object must contain very exhaustive information for which even the availability of suitable macros (as in ET++) require a large effort from the class programmer. *ICE* includes a parser for *C++* definitions, that automatically generates code for the corresponding type-object.

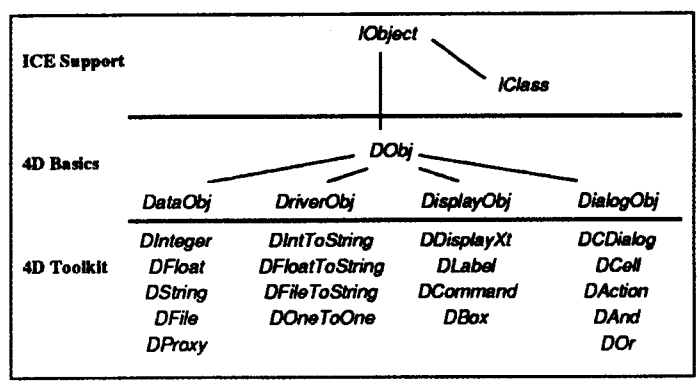

Figure 3: The 4D Classes

4 The 4D toolkit

The *4D* toolkit provides a set of *C++* classes (see Figure 3) that are used to create the UI. The first levels of the inheritance hierarchy relate to the run time support and to the generic functionality defined by the *4D* model. More specialized classes provide the components available for user interface construction.

Base Classes A base class, *DObj*, is defined to support the *4D* model, its components and links. It defines a common behavior to establish, verify and maintain lists of links, using links to send and receive messages. Links rely on the *ICE* message passing mechanism provided by the IObject base class. Four classes are derived from *DObj*: *DataObj*, *DriverObj*, *DisplayObj* and *DialogueObj*. An object belongs to a specific component if its class inherits from one of these.

***Display* classes** Display objects have access to the Window System. The consensus around the X Window System made available several graphical toolkits (Xt, Andrew, InterViews). The current implementation of the *4D* toolkit uses Xt (and several widget sets: Athena, Motif). This provides *4D* with features like system-independence, portability, network transparency, multiple options of functionality and "look and feel". Furthermore, and since these toolkits are often difficult to use, an independent high-level access to its entities is already an advantage.

One class, *DisplayXt*, defines the interface to the Xtoolkit intrinsics. Another class, *DisplayRoot*, is the root class in the display hierarchy. The other *Display* classes bind to the specified widget sets (Label, Command, ...) and are derived from *DisplayXt*. *Display* classes are created automatically with the help of parsing tools, and therefore no effort is required when, for example, a new widget set needs to be integrated in the toolkit.

***Data* classes** *Data* classes are currently simple data types. However, following the same approach adopted for the automatic creation of *Display* classes, they may be complex data structures like *Tree*, *Matrix* or *Graph*. A specialized *Data* class, *Proxy*, interfaces with the computational component of the application, acting as its representative in the UI [Shapiro 86]. This object uses the *ICE* run-time support to dynamically create its interface.

We believe that, in the future, the *4D Data* classes should be created automatically from available *C++* class libraries, just as we currently do with *Display* classes. This would enrich the toolkit and *INGRID* significantly.

***Driver* and *Dialogue* classes** *Drivers* are expected to be dedicated objects for specific conversions. However, functions like the *X Converters* may be used to implement them. *Dialogue* objects are currently message handlers that can be composed to perform the parsing of the message stream.

Save and Retrieve The capability of doing a snapshot of the user interface run-time structure is essential for interactive construction of the UI. The Save/Retrieve facility has two concerns. The first is related with the run-time structure within the *4D* model, mainly the created objects and their links; The second is concerns the graphical parameterization of *Display* objects, and is related with the widgets used internally by *Display* objects. In the first case, the *ICE* run-time information is sufficient to produce a snapshot; in the second, a database with graphical information must be generated, using the facilities provided by the X Window System (*X Resource Management*).

5 The INGRID Tool

The final step in the implementation of our UI construction environment is the development of the interactive construction tool. We considered the following issues as fundamental in the design and development of such a tool:

- **Direct manipulation**
  Direct manipulation is well adapted to the interaction with a UI construction tool, namely

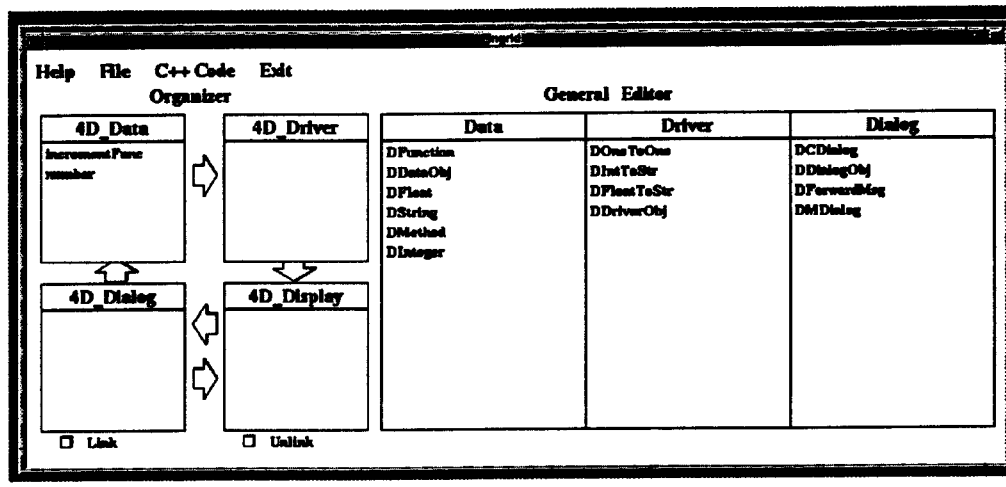

Figure 4: The Organizer and the Data,Driver and Dialogue Editors

when the graphical part of the UI is being handled. It is however difficult to apply to general purpose object customization e.g. parameterization of the attributes of a data object. The solutions to adopt in this case should not limit the interactive nature of the tool.

- **Model enforcement**
  The importance of a flexible model for a UI has been stressed in the beginning. The UI construction tool enforce it during the process of UI construction, providing directions to the user building the interactive application.

- **Multi-threaded interaction**
  The ability for the user to interrupt an action, execute another and return to the conclusion of the first, ought to be supported by the tools.

- **Binding to the computational component**
  The result of a working session is a set of files that describe the UI. However, mechanisms must be provided to define the binding to the computational part of that application. Those mechanisms offer a flexible front-end that supports connections to multiple programming languages without requiring language specific knowledge from the UI programmer. Besides *C++* and *C*, binding to the computational part should also be available through languages like Pascal or Lisp.

- **Help and guidance**
  The flexibility provided by an interactive tool, with direct manipulation of multiple UI entities, may present to the designer a very large set of options. Help and guidance have to be provided throughout the UI construction process, either by pointing the most adequate solution, defining a structured approach to the interface organization coherent with the model, or even by limiting the choices to the ones valid in the current context.

According to these requirements, the interactive programming goals and the *4D* model and toolkit, we considered that the *INGRID* tool should incorporate five components:

- **Interface organizer** The interface organizer allows global access to the interface objects (see fig. 4). It is the front-end of *INGRID*. The organizer manages the creation of links between UI objects, and provides the interface to global functions like UI store/retrieve, *C++* code generation, on-line help, etc.

- **Display editor** This editor handles objects with display properties, like position, size, text and bitmaps. The editor is divided in three areas: palette, canvas, and attribute programmer. These areas are the interface to the instantiation, placement and parameterization oper-

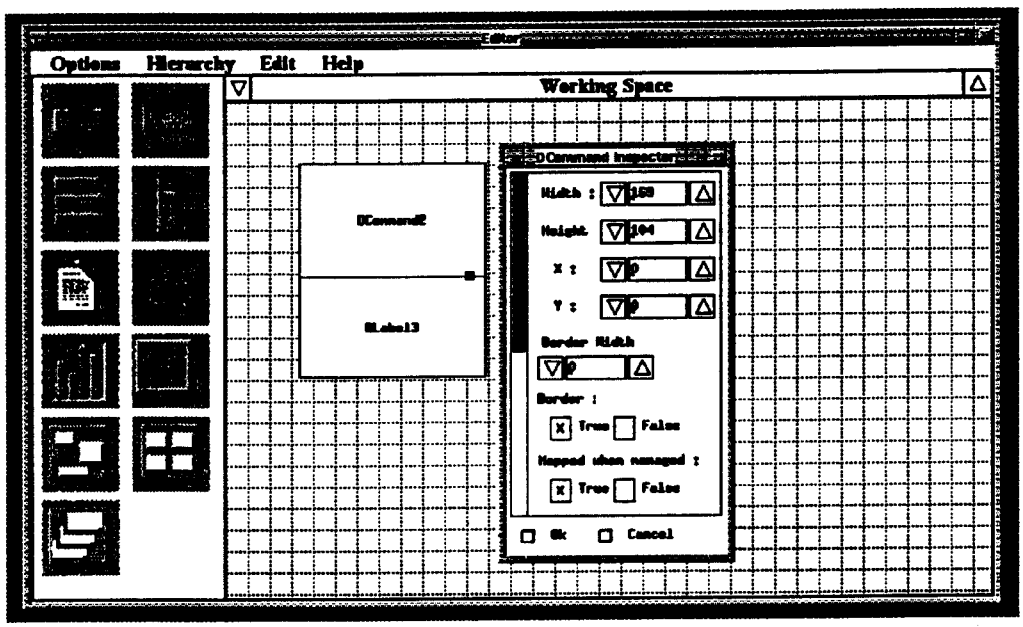

Figure 5: The Display Editor and the Inspector of a *DCommand*

ations (see fig. 5). Object customization is performed through class specific *inspectors*.

As mentioned before, the *Display* classes are generated automatically generated from *Xt* widgets with auxiliary tools. The same happens with the *inspectors*, which are generated automatically from the available type information. This functionality makes it very easy to integrate a new widget set. Currently, we support the Athena and the OSF/Motif 1.1 widget sets.

- **Data editor** *Data* objects represent abstract structures used by both the interactive and the computational components of the application. The *Data* editor provides an identification mechanism that associates a user-readable name to each object. Later, these objects can be selected from a menu of created instances. Instantiation and parameterization can also be performed from menus that refer to the *4D* toolkit *Data* classes.

- **Driver editor** The nature of *Driver* objects difficults the use of a direct manipulation paradigm adapted to their parameterization. In this case, menus allow instantiation, and parameterization. The identification mechanism is similar to the one used for *Data* objects.

- **Dialogue editor** The dialogue editor depends on the dialogue control model that is provided by the underlying toolkit. Our first approach defines an object oriented event driven dialogue for which an event or a sequence of events arriving to an object may trigger a pre-defined action. The dialogue can be represented as a directed graph, suitable for interactive programming using direct manipulation techniques.

6 Related and Future work

The architecture and components of our interface construction environment are closely related with existing systems. Smalltalk-80 [Goldberg 83b] has been an inspiration for the object oriented approach and much of the run-time support system.

We believe that object oriented design methodologies and interactiveness have to be provided in the current Unix workstation environments based on "standard" and widely available UI technologies (namely X and Xt). ADEW [Neuendorfer 91] and the NeXT Interface Builder [Webster 89] are two

good examples of object oriented tools designed for the workstation environment. Their adoption implies however a definite choice of a programming environment and UI style.

Several UI development tools specifically designed to build interfaces with *Xt* widget sets have been reported [Sall 91], [Foody 91]. Several design and architectural decisions presented by these tools are significantly close to our own decisions (the editor approach, the interpretation capabilities, the external representation mechanisms). We stress however that, although our current implementation is based on *Xt*, some degree of independence from the toolkit, or at least, an indirection or encapsulation level, is important. Moreover, the construction of the UI has to consider the other components, namely dialogue and binding to the computational component of the the application.

Future Directions The user interface architecture that supports *INGRID* can be extended in several ways. One of them is the introduction of global, persistent and shared *Data* objects, supported in the *Comandos* platform [Marques 89,Guedes 89], or of replicated *Data* objects, bases in a platform such as Isis [Birman 90]. This evolution may provide facilities for building applications that manipulate persistent information without explicit programming of the access to storage systems, and for construction of multiuser or CSCW (Computer Supported Collaborative Work) [Greif 88] applications.

On the other hand, the architecture and tool provide a comprehensive framework for the development of specialized tools, that do not manipulate general purpose user interface objects but rather domain specific components. In fact, domain specific toolkits may be derived from the base *4D* classes and inherit all the interactive manipulation facilities. One example of a set of components is a hypermedia toolkit developed according to the same principles [Guimaraes 90], [Puttress 90]. The evolution of the system in this direction will provide tools for construction of domain specific applications, which we can refer to as CASE tools.

7 Conclusions

We believe that interactive tools are a good solution for User Interface construction in the context of applications and programming environments where experimentation is envisaged. Moreover, these tools have to be as open as possible, to cope with technology changes and evolution.

The consideration of three components in the user interface construction environment seems to be adequate, both to satisfy the functionality requirements and to ease the implementation process:

- The user interface model covers the essential aspects of the user interface and its object oriented nature provides a correct design methodology.
- The effort spent in the development of a runtime support system for *C++* proved to be rewarding, given the degree of openness and compatibility with existing software that is achieved.
- The *4D* toolkit, as an high level toolkit, is an interesting solution to encapsulate and integrate discrete components behind a normalized and generic interface. Further exploitation of this encapsulation role is expected to provide a much richer set of tools.
- The adequate exploitation of the object oriented design and implementation guidelines has proven extremely useful to achieve all these objectives.

Acknowledgements

This work was supported partially by the Commission of the European Communities, under the (*Comandos* Esprit Project), and partially by JNICT, the Portuguese National Board for Research.

References

[Birman 90] K. Birman, R. Cooper, T. Joseph, K. Marzullo, M. Makpangou, K. Kane, F. Schmuck, and M. Wood. *The ISIS*

System Manual, Version 2.0. Technical Report, The ISIS Project, Dept. of Computer Science, Cornell University, Ithaca, September 1990.

[Carricco 90] L. Carriço, N. Guimaraes, and P. Antunes. INGRID : A Graphical Tool for User Interface Construction. In *Proceedings of the EUUG Spring Conference, Munich,*, April 1990.

[Coutaz 87] J. Coutaz. The Construction of User Interfaces and the Object Paradigm. In *Proceedings of ECOOP'87*, pages 121–130, June 1987.

[Coutaz 89] J. Coutaz. Architecture Models for Interactive Software. In *ECOOP'89, Proceedings 3rd European Conf. on Object-Oriented Programming, Nottingham*, July 1989.

[Dance 87] J.R. Dance. The Runtime Structure of UIMS-Supported Applications. *Computer Graphics*, 97–101, April 1987.

[Ellis 90] M. Ellis and B. Stroustrup. *The Annotated C++ Reference Manual.* Addison Wesley, 1990.

[Foley 82] J.D. Foley and A. Van Dam. *Fundamentals of Interactive Computer Graphics.* Addison-Wesley, 1982.

[Foley 90] J.D. Foley, A. Van Dam, Steven Feiner, and John Hughes. *Computer Graphics - Principle and Practice.* Addison-Wesley, 1990.

[Foody 91] M. Foody. UIM/X : A GUI Builder for Motif or OPENLOOK. Tutorial presented at the 5th Annual X Technical Conference, Boston, January 1991.

[Gamma 88] E. Gamma, A. Weinand, and R. Marty. ET++ - An Object Oriented Application Framework. In *Proceedings of the Autumn 1988 EUUG Conf., Portugal*, pages 159–173, October 1988.

[Goldberg 83a] A. Goldberg. *Smalltalk-80: The Interactive Programming Environment.* Addison-Wesley, 1983.

[Goldberg 83b] A. Goldberg and D. Robson. *Smalltalk-80: The Language and its implementation.* Addison-Wesley, 1983.

[Green 85] M. Green. Report on Dialogue Specification Tools. In *User Interface Management Systems , G.Pfaff (ed.)*, pages 7–20, Springer Verlag , Berlin, 1985.

[Greif 88] I. Greif. *Computer Supported Collaborative Work :A Book of Readings.* Morgan Kaufman Publishers, 1988.

[Guedes 89] P. Guedes and J.A. Marques. Operating System Support for an Object-Oriented Environment. In *Proceedings of the 2nd IEEE Workshop on Workstation Operating Systems, Asilomar*, September 89.

[Guimaraes 90] N. Guimaraes. HOT: a Generic Hypermedia Toolkit. In *Proceedings of the TOOLS'90 Conference, Paris*, June 1990.

[Guimaraes 91] N. Guimaraes. INGRID: Interactive Graphical Interface Designer. Tutorial presented at the 5th Annual X Technical Conference, Boston, January 1991.

[Hartson 89] H.Rex Hartson and Deborah Hix. Human-Computer Interface Development:Concepts and Systems for its Management. *ACM Computing Surveys*, 21(1), March 1989.

[Lantz 87] K. Lantz. Reference Models, Window Systems and Concurrency. *Computer Graphics*, 87–97, April 1987.

[Linton 87] M. Linton, P. Calder, and J. Vlissides. InterViews: A C++ Graphical Interface Toolkit. In *Proceedings of USENIX C++ Workshop , Santa Fe*, November 1987.

[Marques 88a] J.A. Marques, L.P. Simoes, N. Guimaraes, Luis Carriço, and M. Sequeira. Images - An Approach to an Object Oriented UIMS. In *Proceedings of the Autumn 1988 EUUG Conference*, October 1988.

[Marques 88b] J.A. Marques, L.P. Simoes, and N.Guimaraes. A UIMS and Integrated Environment for the Somi Workstation. In *Proceedings of the ESPRIT'88 Conference, Brussels*, pages 1001–1019, North Holland, November 1988.

[Marques 89] J.A. Marques and P. Guedes. Extending the Operating System to Support an Object-Oriented Environment. In *Proceedings of OOPSLA'89, ACM, New Orleans*, October 89.

[McCormack 88] J. McCormack, P. Asente, and R. Swick. *X Toolkit Intrinsics - C Language Interface.* 1988.

[Microsystems 88] Sun Microsystems. *The Open Look GUI Functional Specification, Pre-Release.* Sun Microsystems, July 1988.

[Neuendorfer 91] T. Neuendorfer. ADEW: Building Applications in an Embedded Object Environment. Tutorial presented at the 5th Annual X Technical Conference, Boston, January 1991.

[Neuwirth 88] C. Neuwirth and A. Ogura. *The Andrew System Programmer's Guide to the Andrew Toolkit ITC.* Technical Report, Carnegie-Mellon University, January 1988.

[OSF 89] OSF. *Motif Style Guide, Revision 1.0.* Open Software Foundation, Cambridge, MA, USA, 1989.

[Puttress 90] J.J. Puttress and N. Guimaraes. The Toolkit Approach to Hypermedia. In A. Rizk, N. Streitz, and J. Andre, editors, *Hypertext: Concepts, Systems and Applications, Proceedings of ECHT'90, Paris*, pages 25–37, The Cambridge Series on Electronic Publishing - Cambridge University Press, November 1990.

[Sall 91] K. Sall. TAE Plus, a NASA-developed User Interface Design Tool. Tutorial presented at the 5th Annual X Technical Conference, Boston, January 1991.

[Scheifler 86] R. Scheifler and J. Gettys. The X Window System. *ACM Transactions on Graphics*, 5(2):79–109, April 1986.

[Shapiro 86] M. Shapiro. Structure and Encapsulation in Distributed Systems: the Proxy Principle. In *Proceedings 6th Intl. Conf. on Distributed Computing Systems ,IEEE Cambridge , Mass.(USA)*, pages 198–204, May 1986.

[Simoes 87] L.P. Simoes and J.A. Marques. Images - an Object Oriented UIMS. In *in Proceedings of Interact'87 Human-Computer Interaction, IFIP*, H.Bullinger , B.Shackel (eds) , North-Holland, August 1987.

[Spector 88] A.Z. Spector and J.H. Howard. *Andrew.* Selected Papers from the Usenix Winter Conference 1988, Dallas, December 1988.

[Stroustrup 85] B. Stroustrup. *The C++ Programming Language.* Addison-Wesley, 1985.

[Szekely 88] P.A. Szekely and B.A. Myers. A User Interface Toolkit based on Graphical Objects and Constraints. In *Proceedings of OOPSLA'88 Conference*, pages 36–45, September 1988.

[Webster 89] B.F. Webster. *The NeXT Book.* Addison-Wesley, 1989.