Support for Interactive Tools and Systems

Luís Carriço, Nuno Guimarães, Pedro Antunes

INESC

R.Alves Redol, 9, 6o., 1000 Lisboa
Portugal

Tel: +351-1-3155150
e-mail: lmc@inesc.pt

1 Introduction

In this paper, we describe what we consider to be the essential support for interactive tools, in an open environment. The next section proposes an extensible architecture that covers most aspects that tools and applications are concerned with. The following section describes a runtime system for C++ that supports our interactive programming needs. Then a section is dedicated to tools that were built using these two main components. The first tool is the INGRID application builder [3,8,9]. The second, called HypIngrid, is a hypertext system [4] that mimics part of the HyperCard [11] functionality in the Unix/X environment. This tool has evolved from INGRID with a minimal effort, thus proving the concept that the underlying support for interactive programming and application construction is generic and reusable. We finish with an overview of future directions, as well as a set of conclusions.

2 Architecture for Interaction

The definition of an architecture for interactive applications should provide the structural guidelines for its construction. As premisses it should define a clear separation between the computational and interactive parts of an application (*dialogue independence* as defined in [10]), a well defined functional partition within the interactive component and a methodological approach to the composition of those functional parts.

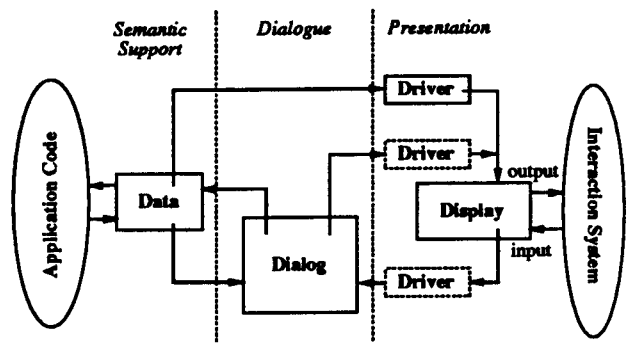

Figure 1: The 4D functional architecture

Functional partition According to the above principles, we defined the 4D architecture. It classifies the objects composing an interactive application into four possible categories as shown in fig. 1:

Display: This category corresponds to the presentation component of the UI [12], usually interfacing with a graphical system or a set of multimedia devices. Display objects integrate existing UI components, such as *Xt* widgets [15], adapting them to the 4D architecture.

Data: These objects provide the interface between the interactive and computational parts of the

Proceedings of the ERCIM Workshop on Distributed Systems, User Interfaces and Multimedia, Decision Support Methods and Applications. Lisboa, Portugal, November, 1991.

application. In general they represent abstract data types that range from simple ones (like Integers, Floats or Lists), to full database access. Data objects are another locus of integration for existing class libraries (like libc++, libg++ or NIHCL).

Dialogue: Dialogue objects define the control of the interaction. Although the architecture promotes the definition of event driven dialogues, it easily allows other dialogue models.

Driver: This category manages the transfer of information between objects. Its role is to perform data conversion, optimising the integration of the other components.

Architecture's dynamics The 4D architecture introduces a fifth concept addressing the dynamics of the application:

Link: A selective communication channel that carries messages between objects. Each *link* is characterised by a *sender* object, an *event identification* on that sender, a *receiver* and an *action* to be performed on the receiver.

When an event is triggered in an object, its links (for which he is the sender) are scanned and, for those corresponding to that event, the associated actions are executed. That execution corresponds to a message sent to the receiver. On the reception of a message, the new object may trigger its own events, repeating the process. On Display objects, events are usually triggered as a consequence of a user action.

The link is mainly a programmable entity that allows each component to work autonomously without explicit knowledge of the other objects. In fact, as actions may specify any message on the receiver, they can adapt any object to its environment. For structural organisation, however, Data and Display objects must not be linked together (see fig. 1) and the action specification of their links is disabled. This rules will definitely concentrate the definition of the specific behaviour of each interface in the Dialogue and Driver components.

Object composition Apart from the principles that guide object classification into 4D categories, object composition must obey the following rules:

1. Links defined on a component, to objects outside its encapsulator (the composed object), follow the rules applied to both objects (component, and encapsulator).
2. Incoming links can never specify actions in components.
3. Display objects can only be composed into an encapsulator Display object.

3 Run-time support

The use of 4D objects as a base for the development of interactive programming tools, and the implementation of links, both require a run-time foundation which supports interpretation, as a base for experimental programming [13], and a mechanism which allows the preservation of programming sessions.

The language level support Having chosed a language like C++ for the sake of the openness and portability it provides, a run-time support, named ICE, has been developed which offers the following services:

Object identification: enables the association of user readable names to objects.

Object creation: provides a primitive for object instantiation, independently of the class it belongs to.

Message invocation: allows the invocation of member functions through a message like mechanism, using a common generic primitive which maintains the characteristics of the host language (e.g. member overloading).

Object storage/retrieval: allows object passivation and activation, supporting multiple external representations, including C++ itself.

Except for the first service (*name-service*), implemented as a set of hash tables, all the other are

based in the existence of *type-objects*. These objects describe classes and C++ primitive types, providing an uniform run-time type information. Class specific type-objects implement the algorithms for method lookup offering the services for object creation and message invocation. The storage/retrieval service is supported by dedicated objects, named *io-objects*. Each of them may define its own syntax and storage source/sink, but all of them base the semantic contents of their storage result on the instance description obtained from type-objects. Finally, ICE provides a common interface to most of the above services, that can be inherited by deriving from the *IObject* class, or simply using typed-references, represented by *IOID*.

ICE also includes a parser for C++ definitions, that generates code for the static instantiation of type-objects.

Architecture specific support The basic aspects of 4D architecture are defined on several classes that classify objects into each category and provide mechanisms to establish links, verify and maintain lists of links, trigger events, select them and executing the associated actions. This execution rely on the ICE message invocation mechanism provided by the IObject base class. Finally, more specialised classes were implemented in this foundations, integrating existing toolkits, simply recurring to parsers and other code generation tools.

4 Tools for Application Construction

The 4D architecture and the ICE run-time provided support for the construction of two tools. The first tool, **INGRID**[1], is an interactive tool for user interface construction, allowing rapid prototyping and incremental development. The second tool, **HypIngrid**, is an open hypertext authoring system that allows the creation of hypertext applications, with a functionality very similar to HyperCard.

[1] INteractive GRaphical Interface Designer

4.1 INGRID

The INGRID tool is composed by a set of subtools. The top-level one is the **Interface Organizer**. It allows access to the interface objects, manages the creation of links, and provides the interface to functions like save/retrieve, C++ code generation, online help, etc. Display objects can be created and parameterised with the **Display Editor**. General object parameterisation can be performed through class specific *inspectors*.

INGRID and the 4D architecture The adoption of the 4D model for user interface construction offers several advantages:

- The definition of four categories allows a structured guided construction of the user interface.
- The behaviour of 4D objects allows the use of direct manipulation through the programming process.
- The flexibility of the link concept allows modification and test of the interface under construction, without having to compile it.
- Composition allows the definition of new classes that organise simple components and can be easily inserted in the INGRID tool.
- The Store and Retrieve functionality allows to save and recover the UI without explicit knowledge of the objects it handles.

INGRID and ICE Using the facilities described above for the ICE support, INGRID is able to:

- Allow a new class to be easily added to the tool. As an example, the integration of the Motif widget set was done simply by (automatically) generating a new Display class for each widget, that afterwards could be used transparently by INGRID.
- Use the type run-time information to build class *inspectors* for object parameterisation.
- Use the method information to guide the user through the establishment of links between UI objects.

4.2 HyperCard on Unix

Once INGRID was available, we observed that it could be easily modified to produce an hypertext authoring system, similar to HyperCard. In fact, the visual characteristics of HyperCard were provided by the INGRID Display editor, the Data objects of the 4D architecture are the components that encapsulate the introduction of persistence, and the HyperCard scripting language could be seen as a dialogue mechanism.

The development of this system explored the functionality of existing components: The SOHO[2] storage system, the 4D Toolkit, and the INGRID interface builder.

HypIngrid and SOHO The SOHO system is based on the HAM model [2] and its object oriented interface offers a set of hypertext objects: graph, context, node, and link. All of these objects may have attributes attached to them. Internally, SOHO is implemented using the *sdbm* library. HypIngrid defined the *Stack, Background, Card, Field and Button* abstractions using these storage facilities.

The HypIngrid Management Component The HypIngrid abstractions were encapsulated into 4D Data objects, in order to be incorporated in the overall application architecture. This was done with the help of a simple parser.

This component is also responsible for the parsing and execution of HypIngridTalk, the scripting language that is a subset of HyperTalk.

The storage and retrieval of components, like Cards, is rather transparent, and relies upon the ICE and 4D functionality. In fact, Cards are saved using the ICE external representation, together with Xt resource files, in SOHO nodes.

HypIngrid Visual Interface The visual interface of HypIngrid evolved from the INGRID Display Editor. All the direct manipulation facilities offered by the Display Editor apply now to the creation and manipulation of HypIngrid objects. Some facilities where added for stack import/export operations, script editing, and expedite hypertext linking.

The overall result highlighted several advantages of the approach. First, the openness of the system allows the integration of other tools, like the audio device of the Unix workstation, dedicated editors, for text, drawings or images, or other specialised processes. Second, the graphical interface is also based on generalised tools (Xt). This provides *look and feel* compatibility with other applications. On the other hand, it is conceivable to extend the HyperCard concepts with other objects like, for example, a Motif *toggle* to be used as a pin while navigating in the hypertext document.

5 Conclusions and Future Directions

The fundamental conclusions we draw from the work described in this paper are the following:

- Interactive applications and tools require a comprehensive and open architecture to support functional partition and integration of available components. The comprehensiveness of the architecture facilitates the programming process by providing a uniform object model for all the components, and standardised interconnection mechanisms. We believe that the 4D architecture is a good step in this direction.
- The construction of interactive tools and systems requires support for interpretation and rapid prototyping. This can be provided by the language and programming environment (Smalltalk [6], Objective C [5]) or added through a run-time support system like ICE. Although implying an extra effort, it proved successful to design and implement such a run-time, given that we were able to keep openness and easy integration with external components.
- The facility of evolving the INGRID tool to a hypertext/authoring system shows that there is a common denominator in these family of tools, which is implemented by the joint cooperation of the architecture and the run-time.

[2]Storage Of Hypermedia Objects

This fact also suggests that further functionality of the run-time and architecture should be made generic and reusable across different tools and applications.

The future directions can be easily extrapolated from the above conclusions. The promising directions are the extension of the architecture and the extension of the run-time. Our goals are to extend them in the following way:

- The purpose of the run-time is to support interactive programming and rapid-prototyping. Generic support for knowledge acquisition and manipulation is a requirement for adaptive and more intelligent user interfaces [14], which makes it an important extension .
- The architecture can be extended to support distribution. Promising experiences have been made in this direction, [1] opening the way to a smooth transition to CSCW [7] applications.

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

[1] P. Antunes, N. Guimarães, and R. Nunes. Extending the User Interface to the Multiuser Environment. European Conference on Computer Supported Collaborative Work, CSCW Developers Workshop, Amsterdam, September 91.

[2] B. Campbell and J. Goodman. HAM: A General Purpose Hypertext Abstract Machine. *Communications of ACM*, 31(7):856–861, July 1988.

[3] 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.

[4] J. Conklin. Hypertext: An Introduction and Survey. *IEEE Computer*, 17–41, September 1987.

[5] B.J. Cox. *Object-Oriented Programming - An Evolutionary Approach.* Addison-Wesley, 1986.

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

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

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

[9] N. Guimaraes, L. Carriço, and P. Antunes. INGRID : An Object Oriented Interface Builder. In *Proceedings of the TOOLS'91 Conference, Santa Barbara, California*, July 1991.

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

[11] G. Harvey. *Understanding Hypercard.* Alameda, CA : SYBEX Inc., 1988.

[12] Brad Myers. User Interface Tools: Introduction and Survey. *IEEE Software*, 15–23, January 1989.

[13] B. A. Sheil. Power Tools for Programmers. In David R. Barstow, Howard E. Shrobe, and Erik Sandewall, editors, *Interactive Programming Environments*, chapter 2, pages 19–30, McGraw-Hill, 1986.

[14] J.W. Sullivan and S.W. Tyler. *Intelligent User Interfaces.* ACM Press, 1991.

[15] D.A. Young. *X Window Systems Programming and Applications with Xt.* Prentice Hall, 1989.