The Design for the Amulet User Interface Toolkit

Brad A. Myers, Rich McDaniel,
Andrew Mickish, Alex Klimovitski

January 1995

Human Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

(412) 268-5150
bam@cs.cmu.edu
http://www.cs.cmu.edu:8001/Web/Groups/amulet/amulet-home.html

Abstract

Amulet is a new user interface software environment for C++ to support future user interface software research. This environment, which will be portable across X/11, Microsoft Windows, and the Macintosh, is designed to be very flexible: parts can be replaced and new technologies and widgets can be easily created and evaluated. Built-in support will be provided for direct manipulation, multi-font text editing, gesture recognition, speech recognition, 2-D and 3-D animations, visualizations including maps and large data sets, world-wide-web browsing and editing, and multiple people interacting with the system at the same time (CSCW). Another goal is to be useful for students, which means that Amulet must be easy to learn. Finally, the system will provide sufficient performance, robustness and documentation so it will be useful for general user interface developers.

Keywords: User Interface Software, User Interface Management Systems, Toolkits.

This research was sponsored by NCCOSC under Contract No. N66001-94-C-6037, Arpa Order No. B326. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NCCOSC or the U.S. Government.

1. Introduction

The Amulet user interface development environment will include a number of design and implementation innovations including new constraint, object, input, and output models, and new forms of interactive tools. In addition, it will support the creation of innovative user interfaces that incorporate the features that user interface researchers are now investigating, such as speech and gesture recognition, 3–D, animations, visualizations, world-wide web (WWW) access and editing, and multiple people operating at the same time (called computer-supported cooperative work—CSCW).

Amulet, which stands for Automatic Manufacture of Usable and Learnable Editors and Toolkits, is being implemented in C++. Amulet is based on our group's substantial experience with the Garnet user interface development environment [Myers 90a], which is implemented in Common Lisp. Amulet will bring to C++ the dynamic and rapid user interface design and implementation capabilities that Garnet provided in Lisp, while adding many new capabilities.

This paper provides an overview of the goals and the initial design of Amulet. At the time of this writing, most of the toolkit level of Amulet is complete. We expect to release Amulet for general use, probably in early 1995, by putting it in the public domain.[1]. If you are interested in using Amulet when it is available, please watch the WWW address given above, or send mail to amulet@cs.cmu.edu to be put on the mailing list.

2. Goals

An important research objective of the Amulet project is to provide high-level support for the *insides* of application programs. Conventional toolkits like the Macintosh Toolbox and Unix Motif provide a collection of widgets like menus, scroll-bars, buttons and text input fields. However, for graphical applications like drawing editors, CAD programs, visual language editors, visualizations, and charting programs, most of the programming for the user interface deals with the contents of the graphic windows which do not contain any widgets. For these, programmers are left to program directly at the

[1] The Garnet user interface development environment is already available for free by anonymous FTP on a.gp.cs.cmu.edu. Change to directory /usr/garnet/garnet and retrieve the file README, and follow the directions. Garnet runs on any Common Lisp system on Unix or the Macintosh. More information on Garnet is available at http://www.cs.cmu.edu:8001/Web/Groups/garnet/garnet-home.html.

window manager level, without much support. In contrast, Amulet will provide high-level support for these kinds of graphical applications including automatic refresh of the graphics, widgets such as selection handles to make interactive behaviors easy to handle, and built-in editing commands, such as cut, copy, paste, print, save, open and undo, that can often be used directly from the library without modification.

Another objective for Amulet is to explore the extensive use of *interactive tools* that will allow most of the user interface to be specified without conventional programming. Our ultimate goal is to allow the user interface designer to simply *draw* examples of the graphics of the interface, and then *demonstrate* the interactive behaviors to show how the interface should react to the user. The motivation for this is that whereas today's programming frameworks such as MacApp [Wilson 90], have demonstrated productivity gains of factors of 2 to 5, interactive tools like HyperCard and interface builders like the NeXT Interface Builder have demonstrated productivity gains of factors of 10 to 50. The various interactive tools in Garnet have demonstrated that many parts of interfaces can be created interactively. Amulet will contain practical and useful tools that cover even more of the interface than Garnet's.

In addition to incorporating research innovations into Amulet itself, we want to make Amulet useful for other user interface researchers and developers. This will help to demonstrate that the innovations in Amulet are sound and useful, and will hopefully facilitate technology transfer. In order of priority, we want to make Amulet useful to:

- **User Interface Researchers**. Amulet will contain facilities to make it easy to create new kinds of interactive tools for user interface software construction. In addition, the constraint system is designed to support different constraint-solving algorithms. The "intrinsics layer" of the toolkit makes it easy to create new kinds of widgets and modify the existing widgets. Finally, hooks are in place to allow integration of new input mechanisms such as speech, gestures and CSCW. The MASTERMIND project at USC/ISI and Georgia Tech [Neches 93], has already started using Amulet as the foundation for their user interface software research, and we hope many others will use Amulet after it is released.
- **Students.** By aiming for Amulet to be useful to students, we will perform the required iterative user testing and refinement to simplify the programming interface of Amulet itself. The success of SUIT [Pausch 91] and Microsoft's Visual Basic

show that it is possible to provide useful functionality in a way that is easy to learn, but unlike those other systems, Amulet will provide a natural growth path to the complete fully-functional system.

- **General Developers.** We also want Amulet to be useful for general user interface software construction. For this reason, we will make Amulet run on X/11, Microsoft Windows and the Macintosh. We will also provide a high-level of robustness and documentation. A number of users of our previous Garnet toolkit reported that it was more stable and better documented than competing commercial products, so we are confident that Amulet will also be useful.

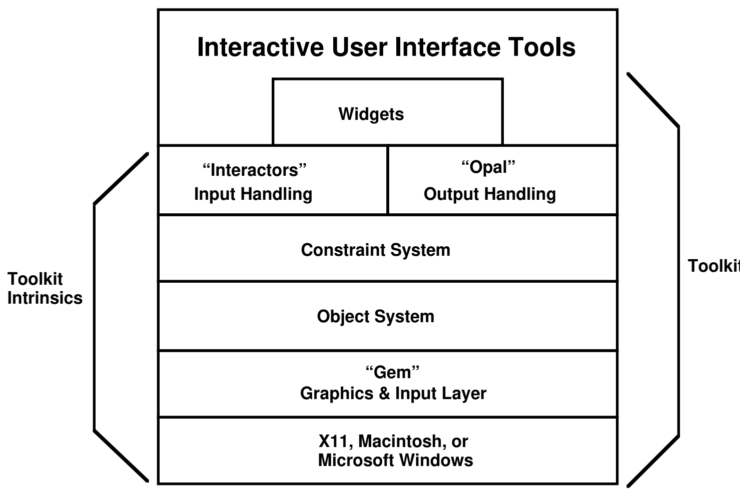

Figure 1: The overall structure of the Amulet system.

3. Design

The overall design of Amulet is much like Garnet. There is a toolkit and there will be a set of interactive tools (see Figure 1). The toolkit contains a set of widgets and the "intrinsics layer" which is how the widgets are implemented. This layer contains an abstract interface to the window managers, and custom models for objects, constraints, input, and output. The following sections describe the overall design of each of these.

3.1. Gem: Abstract Interface to the Window Managers

Since we want to have a portable interface to various window managers, it is important that the window manager dependent parts be isolated into a small layer. Gem, which stands for the Graphics and Events Manager, provides this interface. Gem uses standard

C++ objects and mechanisms to provide a graphics and input interface to the rest of Amulet. Any code written using Gem will port to different windowing systems without change. We expect that the normal Amulet user will never see the Gem interface, however, since the higher-level parts of the Amulet toolkit provide access to the same functionality. Unlike with Garnet where using the higher-level interface was required, in Amulet we will export the Gem interface for advanced Amulet users. In particular, if the programmer wants to make something very efficient, or if access to the low-level window manager interface is required, calling Gem directly may be appropriate.

The basic classes supplied by Gem include a “Drawonable,” which corresponds to window, a “Style” (which contains color, line-thickness, and halftoning information), a “Font,” and a “Point-List” (for polygons). The Drawonable has member functions to draw all of the primitive graphics, such as rectangles, text, lines and arcs. The Gem layer does not provide a retained-object model, so the Opal layer (see section 3.4) is responsible for redrawing the window if it becomes uncovered. Programming at the Gem layer is fairly straightforward, but quite low-level.

3.2. Object System

Amulet implements a prototype-instance object system on top of C++. In a prototype-instance object system, there is no distinction between classes and instances: every object can be used as a prototype for other objects. An object can have an arbitrary number of “slots” (also called member variables or instance variables), and objects can dynamically add and remove slots at any time. Slots of the prototype can be inherited by instances so that changes to the prototype’s slot’s value will be seen by any instances which do not provide a local value for that slot. The advantage of the prototype-instance object system is that it is very dynamic and flexible. All of the properties of objects can be set and queried at run time, and interactive tools can easily inspect, read and set properties. In fact, most of today’s toolkits implement some form of “attribute-value pairs” to hold the properties of the widgets, but Amulet’s object system provides significantly more flexibility and capabilities.

The object system also implements a part-owner hierarchy. The owner is usually a “group” (or aggregate) object or a window, and the parts are either other groups or primitives like rectangles, lines and text. By supporting the part-owner hierarchy in the object system, we get "structural inheritance," which means that when an instance is made of a group which contains parts, the new instance will have instances of all the previous parts. Thus, programmers can create instances of any type of object without knowing whether it is a primitive or a group, and the system will make sure that the instance has the same structure as the prototype.

3.3. Constraints

Amulet integrates constraint solving with the object system. This means that instead of having a "normal" value like a number or a string, any slot of any object can contain a expression which computes the value. If the expression references slots of other objects, when those objects are changed, the expression is automatically re-evaluated. Constraint expressions can contain arbitrary C++ code, and the only restriction is that accesses to slots of objects must use a special function. This function, in addition to returning the value, also sets up a dependency link so that the constraint will be re-evaluated when the other object changes.

An important research area in user interface software is creating new kinds of constraint solvers (e.g. [Vander Zanden 92][Hill 93][Gleicher 93]). Therefore, in addition to the standard solver described above, Amulet is designed to support other constraint systems at the same time. Since two different constraint solvers were made to work together in Garnet [Sannella 94], we believe that this will work, but how to schedule the various solvers is an open research issue, especially when multiple solvers want to set the same slot.

3.4. Opal Output Model

The graphical object layer of Amulet is called Opal, which stands for Object Programming Aggregate Layer. Opal hides the graphics part of Gem and provides a convenient interface to the programmer by providing a retained object model, also called a structured graphics model or a display list. The programmer creates instances of the built-in graphical object prototypes, like rectangle, line, text, circle, etc., and adds them to a window. Then Amulet automatically redraws the appropriate parts of the window if it becomes uncovered, or if any properties of the objects change. This frees the programmer from having to deal with refresh, and objects can simply be created and deleted and their properties set.

3.5. Interactors

Programming interactive behaviors has always been the hardest part of user interface software, especially since most toolkits and window managers only provide a stream of raw input events for each window which the programmer must interpret and manage. Garnet introduced the "Interactor" model for handling input [Myers 90b] which is refined in Amulet. Each Interactor object type implements a particular kind of interactive behavior, such as moving an object with the mouse, or selecting one of a set of objects. Currently, there are only five types (select, move/grow, rotate, new points, and text edit), but these are sufficiently parameterized so that they cover the most common behaviors found in direct manipulation interfaces. Unlike widgets, an Interactor does not include any graphics. To make a graphical object respond to input, the programmer simply attaches an instance of the appropriate type of Interactor to the graphics.

3.6. Widgets

Amulet supplies a complete set of widgets, including pull-down menus, buttons, check boxes, radio buttons, text-input fields, scroll bars, etc. Each widget has different drawing routines for the Motif, Microsoft Windows, and the Macintosh looks and feels. Amulet re-implements all the widgets rather than using the built-in widgets from the various toolkits so that we can provide flexibility and control to programmers who want to investigate new behaviors. This is necessary, for example, to create a scroll bar with two handles [Ahlberg 92] or to support multiple people operating with a widget at the same time for a CSCW application.

In addition, Amulet supplies other widgets for the *insides* of application programs. For example, the selections-handles widget implements the familiar squares around the edges of graphical objects that shows what is selected and allows them to be re-sized. All other toolkits require programmers to re-implement this standard behavior.

3.7. Commands

Often, the Interactors and widgets operate simply by setting the appropriate slots of objects and having the values propagated by constraints. However, sometimes other actions are required. In conventional toolkits, this would be implemented using call-back procedures, but Amulet takes a different approach, based on the design of MacApp

[Wilson 90]. At the completion of each interactor or widget, a command object is generated, and its “do” method is called. The command object is then stored in case the user wants to perform “undo.” Kosbie introduced the idea of making the command objects hierarchical [Kosbie 93], which means that a low-level command object such as “move-object” might be part of a higher-level command object such as “adjust-scroll-bar-indicator.” This gives the programmer significant flexibility while still providing built-in undo. Many standard command objects are provided in the Amulet library and can often be used without change. Furthermore, two different undo mechanisms are provided by the library, and researchers are encouraged to develop new ones.

3.8. Future interactive tools

As stated above, an important goal of Amulet is to provide innovative interactive tools that will allow much of the interface to be created by direct manipulation. The centerpiece of the student version will be an interactive tool. We also plan to provide interactive tools for creating innovative interfaces, such as those that use gesture and speech recognition, 3-D and CSCW. One of the initial interactive tools will be a browser and interactive WYSIWYG editor for HTML (WWW) pages. This will be easy to build given Amulet’s support for multi-font editing with embedded graphical objects.

An important issue is how to allow the interactive tools to execute an interface as it is being created. In Garnet, we could simply use the built-in Lisp interpreter. Some commercial tools, such as UIMX incorporate a C interpreter. The success of tcl/tk [Ousterhout 91] shows that people are anxious to use an interpretive environment for user interface development. We will clearly need to support some kind of interpreter, but we have not decided whether to interface to an interpretive language such as Scheme, tcl, Python or Visual Basic, or whether to use an interpreter for a subset of C++.

4. Conclusions

We are very excited about the potential for Amulet to be a useful and efficient platform on which to do user interface research. We hope it will also be popular for user interface education and for the implementation of real systems. By moving the best features of Garnet to C++, we hope to bring Garnet’s proven effectiveness and ease-of-use to a wider audience. The additional innovations in Amulet will support the kinds of interfaces that will be investigated and developed in the future.

References

[Ahlberg 92] C. Ahlberg, C. Williamson and B. Shneiderman. “Dynamic Queries for Information Exploration: An Implementation and Evaluation,” Proceedings ACM CHI’92 Conference, 1992. pp. 619-626.

[Gleicher 93] Michael Gleicher, “A Graphics Toolkit Based on Differential Constraints,” *Proceedings UIST'93: ACM Symposium on User Interface Software and Technology*, Nov, 1993, Atlanta, GA, pp. 109-120

[Hill 93] Ralph D. Hill, “The Rendezvous Constraint Maintenance System,” *Proceedings UIST'93: ACM Symposium on User Interface Software and Technology*, Nov, 1993, Atlanta, GA, pp. 225-234

[Kosbie 93] David S. Kosbie and Brad A. Myers, “A System-Wide Macro Facility Based on Aggregate Events: A Proposal,” *Watch What I Do: Programming by Demonstration*. Allen Cypher, ed. Cambridge, MA: MIT Press, 1993, pp. 433-444

[Myers 90a] Brad A. Myers, Dario A. Giuse, Roger B. Dannenberg, Brad Vander Zanden, David S. Kosbie, Ed Pervin, Andrew Mickish, and Philippe Marchal. ``Garnet; Comprehensive Support for Graphical, Highly-Interactive User Interfaces," *IEEE Computer*. vol. 23, no. 11. November, 1990. pp. 71-85.

[Myers 90b] Brad A. Myers. “A New Model for Handling Input,"“ *ACM Transactions on Information Systems*. vol. 8, no. 3. July, 1990. pp. 289-320.

[Neches 93] R.Neches, J. Foley, P. Szekely, P. Sukaviriya, P. Luo, S. Kovacevic, and S. Hudson, “Knowledgable Development Environments Using Shared Design Models,” *Proceedings of the 1993 ACM International Workshop on Intelligent User Interfaces*, Orlando, FL, Jan,1993, pp. 63-70

[Ousterhout 91] John K. Ousterhout, “An X11 Toolkit Based on the Tcl Language,” USENIX Winter Conference, 1991, pp. 105-115

[Pausch 91] Randy Pausch, Nathaniel R. Young II, and Robert DeLine, “SUIT: The Pascal of User Interface Toolkits,” *Proceedings UIST'91: ACM SIGGRAPH Symposium on User Interface Software and Technology*. Nov, 1991, Hilton Head, SC, pp. 117-125

[Sannella 94] Michael Sannella, “SkyBlue: A Multi-Way Local Propagation Constraint Solver for User Interface Construction,” *Proceedings UIST'94: ACM SIGGRAPH Symposium on User Interface Software and Technology*. Nov, 1994, Marina del Rey, CA, pp. 137-146

[Vander Zanden 92] Brad Vander Zanden, “An Active-Value-Spreadsheet Model for Interactive Languages,” in *Languages for Developing User Interfaces*, Brad A. Myers, ed. Boston, MA: Jones and Bartlett, 1992. pp. 183-209

[Wilson 90] David Wilson. *Programming with MacApp.* Reading, MA: Addison-Wesley Publishing Company, 1990