Overview of the Amulet User Interface Toolkit

Brad A. Myers,
Rich McDaniel, Alan Ferrency, Rob Miller, Patrick Doane

January 1996

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/~amulet

Abstract

Amulet is a new user interface software environment for C++ on X/11, Microsoft Windows NT or 95, and the Macintosh, which facilitates user interface research by being very open and flexible. In particular, new constraint and undo algorithms can be added, the widgets can modified and replaced, and support is provided for building high-level tools.

Keywords: UI Toolkit

Attending Workshop: Brad A. Myers

Presented at the Human-Computer Interaction Consortium,
Winter Park, CO. February 14-18, 1996

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 includes a number of design and implementation innovations including new constraint, object, input, output and undo models. Amulet was designed with an open architecture, so that user interface researchers can replace and extend parts. For example, Amulet is the first system that supports multiple constraint *solvers* operating at the same time. The undo model also supports new designs. The widgets are implemented in an open fashion using the Amulet intrinsics so that researchers can replace or modify the widgets. The goal is that researchers will only have to implement that parts that they are interested in and use the rest from the library.

Amulet, which stands for Automatic Manufacture of Usable and Learnable Editors and Toolkits, is implemented in C++. Amulet is based on our group's substantial experience with the Garnet user interface development environment [1], which is implemented in Common Lisp. Amulet brings 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 Amulet. Amulet version 1.0 was released in June, 1995, and version 2.0 is in alpha test. Already, a number of research and commercial projects are using Amulet. To get Amulet, see the WWW address on the first page, or FTP the README file from ftp.cs.cmu.edu in directory /usr/anon/project/amulet/amulet/.

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 window manager level, without much support. In contrast, Amulet provides 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 implement, and built-in editing commands, such as cut, copy, paste, to top and bottom, 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 [2], 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 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**. The MASTERMIND project at USC/ISI and Georgia Tech [3] has already started using Amulet as the foundation for their user interface software research, and a number of PhD dissertations at the University of Tennessee and CMU are using Amulet.
- **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 [4] 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. Amulet has been used in three classes at CMU and University of Tennessee, and we are refining the interface based on that experience.
- **General Developers.** We also want Amulet to be useful for general user interface software construction. For this reason, Amulet runs X/11, Microsoft WindowsNT or 95, and the Macintosh. We also provide a high-level of robustness, documentation and performance.

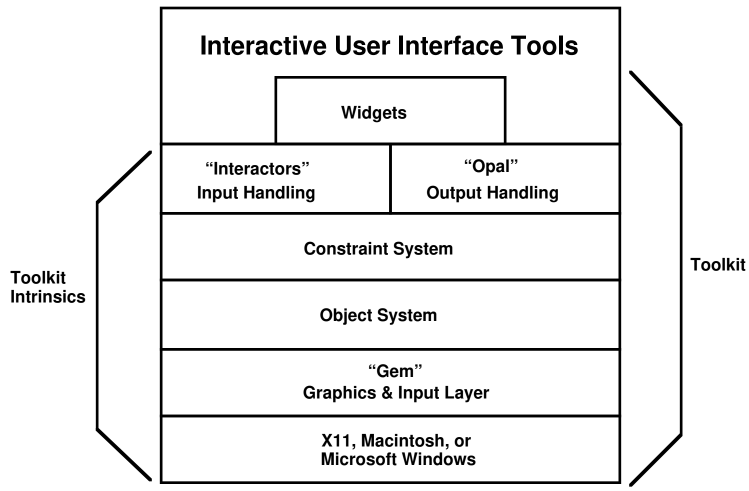

Figure 1: The overall structure of the Amulet system.

3. Design

Amulet contains 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 new models for objects, constraints, input, output and undo. The following sections describe the overall design of each of these.

3.1. Gem: Abstract Interface to the Window Managers

Since we need 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 export the Gem interface for advanced Amulet users. In particular, if the programmer wants to make something very efficient, calling Gem directly may be appropriate.

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. [5-7]). Therefore, in addition to the standard one-way solver described above, Amulet also contains an experimental multi-way, multi-output solver. The "QuickPlan" constraint solver [5] is also being integrated into Amulet.

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. Furthermore, Opal automatically handles object creation and layout when the data can be displayed as lists or tables (graphs and tree layout is coming soon).

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 [8] 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, Amulet contains only five types (select, move/grow, new points, text edit and gesture[1]), 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 will have 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

---

[1]The gesture interactor supports free-hand gestures, such as drawing an “X” over an object to delete it, which will promote experimentation with this promising technology.

two handles [9] 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 (see Figure 2). All other toolkits require programmers to re-implement this standard behavior.

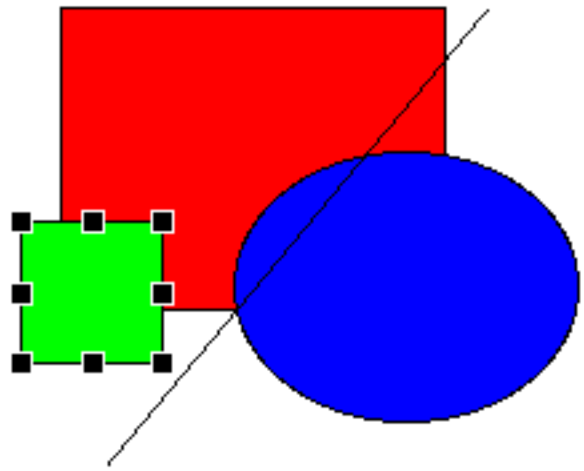

Figure 2: Selection Handles

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. Rather than using a "call-back procedure" as in other toolkits, Amulet allocates a *command object* and calls its "Do" method [10]. Amulet's commands also provide slots and methods to handle undo, selective undo and redo, enabling and disabling the command (graying it out), and help and "balloon help" messages. Thus, unlike MacApp [2], the commands provide a single place for describing a behavior.

Furthermore, commands promote re-use because commands for such high-level behaviors such as cut, copy, paste, move object, create-object, change-property, and become selected are supplied in a library and can often be used by applications *without change.* The commands in Amulet are *hierarchical*, so that a behavior may be composed of high-level and low-level commands [10]. For example, a scroll bar command might internally use a move-object command. This improves modularity and re-use because each command is limited to its own local actions.

Amulet's commands also support investigation of various undo mechanisms. Currently, Amulet supplies three different undo mechanisms that the developer can choose from: single undo like the Macintosh, multiple undo like Microsoft Word Version 6, and a novel form of undo and repeat, where any previous command, including scrolling and selections, can be selectively undone, repeated, or repeated on a new selection [10]. Alternatively, researchers can create their own undo mechanism.

3.7. Debugging Tools

Debugging interactive applications requires additional mechanisms than supplied with conventional development environments. Amulet provides an interactive *inspector* that displays the properties of objects at a high level, and will trace the execution of interactors, the dependencies of constraints, and the changing values of slots. Furthermore, extensive error checking and helpful messages (no "seg faults") make Amulet applications easy to develop and debug.

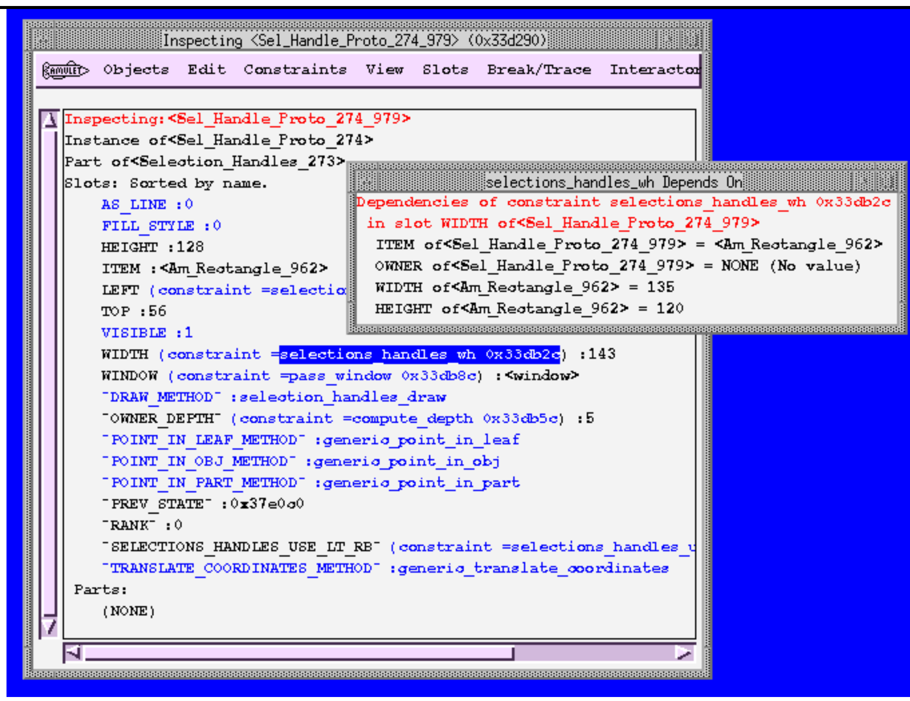

Figure 3: Inspecting a rectangle and its width's constraint

3.8. Future

In the future, we will be investigating techniques to support speech recognition, 3–D, animations, visualizations, world-wide web (WWW) access and editing, and multiple people operating at the same time. We will also be creating interactive tools that will allow much of the interface to be created by direct manipulation.

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. The innovations in Amulet will support the kinds of interfaces that will be investigated and developed in the future.

References

1. Myers, B.A. et.al., *Garnet: Comprehensive Support for Graphical, Highly-Interactive User Interfaces.* IEEE Computer, 1990. **23**(11): p. 71-85.

2. Wilson, D., *Programming with MacApp*. 1990, Reading, MA: Addison-Wesley Publishing Company.

3. Neches, R., *et al. Knowledgable Development Environments Using Shared Design Models*. in *Proceedings of the 1993 International Workshop on Intelligent User Interfaces*. 1993. Orlando, FL.

4. Pausch, R., N.R. II Young, and R. DeLine. *SUIT: The Pascal of User Interface Toolkits*. in *ACM SIGGRAPH Symposium on User Interface Software and Technology*. 1991. Hilton Head, SC.

5. Zanden, B.V., *An Incremental Algorithm for Satisfying Hierarchies of Multi-way, Dataflow Constraints*, . 1995, Computer Science Department, University of Tennessee. p. 55.

6. Hill, R.D. *The Rendezvous Constraint Maintenance System*. in *ACM SIGGRAPH Symposium on User Interface Software and Technology*. 1993. Atlanta, GA.

7. Gleicher, M. *A Graphics Toolkit Based on Differential Constraints*. in *ACM SIGGRAPH Symposium on User Interface Software and Technology*. 1993. Atlanta, GA.

8. Myers, B.A., *A New Model for Handling Input.* ACM Transactions on Information Systems, 1990. **8**(3): p. 289-320.

9. Ahlberg, C., C. Williamson, and B. Shneiderman. *Dynamic Queries for Information Explor-ation: An Implementation and Evaluation*. in *Proceedings ACM CHI'92 Conference*. 1992.

10. Myers, B.A. and D. Kosbie. *Reusable Hierarchical Command Objects*. in *Proceedings CHI'96: Human Factors in Computing Systems*. 1996. Vancouver, BC, Canada.