Abstract

Interactive graphics based software tools have been designed for use in rapid prototyping and testing of user interfaces. These tools form an integrated design environment usable by product designers with no computer language or computer graphics background. A tool based architecture and 'visual interactive programming' are used to accomplish this. The design environment consists of multiple windows, with one tool per window. Each tool is used for specifying a phase in the design process. These phases range from specifying a product's functionality and the relations between functions, to drawing and describing moving parts. A design is tested in a 'proof tool' which uses the information specified in the design tools to 'run' the product. Product designs can be easily entered, modified, and evaluated using these rapid prototyping tools. Consumer reactions to a product's interface can be acquired before the product is built. This paper discusses the design environment, existing tools, future plans, and a walk through of a design.

CCCS numbers: C1270, C6115, C7170.

1. Introduction

In designing a tool to specify control for a process' simulation, much consideration must be given to who the intended user will be. In an effort to bring design and simulation capability to people without any computer science background, a design environment for rapid prototyping of various processes has been developed. The specific domain is design and simulation of consumer-to-product interfaces.

How a person interacts with a machine is called the consumer-to-product interface. This includes what type of switches will be used, what they will look
like, and where they are placed. For products with dynamic displays, the consumer-to-product interface includes the hierarchy of commands a consumer uses. In general, this interface also includes the product appearance in terms of colour, shape, and size.

Intended users of the design environment are product designers and marketing personnel. Product designers will make the most use of the building capabilities of the design environment, whereas marketing people will make use of the simulation capability.

The use of product simulation in market analysis studies will allow for better feedback about a product's potential. Current methods of market studies usually do not allow for 'trying' the potential product. Usually, a cardboard mock up of the product is made and a test moderator explains the use of the product to a group of test subjects. They then fill in a questionnaire that attempts to find out about a product's potential. In some cases a hardware prototype is used. For testing multiple possibilities of a product's design, this approach is time consuming, costly, and difficult to change. In the design and simulation environment developed, changes to a product's interface design can be made easily, rapidly, and test subjects can try a product interface under simulation. Subjects will not be able to indicate how a compact disc player sounds, but they will be able to give an opinion on its ease of use.

We call our design environment MVP, which stands for 'multiple view prototyping'. The techniques used in designing and implementing MVP are based on the Unix*) philosophy of tools, and what we call 'visual interactive programming'. MVP runs on a graphics workstation with a high-resolution bit-map graphics display, and input from a pointing device and a keyboard.

To build prototypes using the MVP system, designers pick building tools out of a toolbox. There are three types of tools in the toolbox; building, simulation, and control. When use of a tool is complete it can be put back in the toolbox or multiple tools can be in use simultaneously. Tools can be put away temporarily and when chosen again will reflect the up-to-date state of the design. Fig. 1 shows the MVP toolbox with all tools closed, and again with three tools open and ready for use. Opening and closing tools is accomplished with a mouse.

The next two major sections of this paper can be read in any order depending on interest. Sec. 2 is on the design of MVP and sec. 3 gives an example design of a consumer product interface using MVP. Sec. 4 discusses some future directions and goals.

*) Unix is a trade mark of AT&T Bell Laboratories.
Interactive graphics based software tools for rapid prototyping

Fig. 1. The MVP design environment. (a) All tools closed. (b) State diagram, button, and draw tools open.
2. Design of the MVP environment

Discussion on the MVP environment is broken down into sub-sections on principles, and implementation of the design. The main principles are 'visual interactive programming' (VIP), and the Unix tool philosophy. The implementation section discusses briefly the hardware, software, and data structures used.

2.1. MVP design principles

The design principles of MVP are based on the following goal: To create an environment (MVP) in which people with no computer-science background can design and test via simulation, the interface between consumer products and consumers. To meet this goal, MVP must be easy to use, flexible, easy to learn, and give a capability not previously or readily available.

2.1.1. Visual interactive programming (VIP)

Visual interactive programming refers to the type of design environment a user of MVP is presented with. It uses graphics based input and output as the communication mechanism between the computer and the user. Most of the tools in the MVP toolbox have graphical output, both in terms of commands via icon based menus and application output. Examples are drawing tools, animation tools, and button or icon editors. Input to each tool is usually through a mouse and occasionally through a keyboard. The mouse is used at different times as a locator, stroke, pick, and button device.) The keyboard is used for labeling and file control.

Visual interactive programming subscribes to the 'what you see is what you get' philosophy (WYSIWYG). All input to tools is immediately processed, and the results are displayed in the current tool and in all related tools. (This will be discussed further in sec. 2.2.) Thus, the displays shown are always the current up-to-date state of a user's design.

'Programming' in VIP means setting up, via graphics based interactive tools, descriptions of how the consumer product is going to function and what it is going to look like. This information is saved and used by the 'proof tool', which is the window where simulation and testing of the interface design takes place.

2.1.2. Tool based

MVP is a tool based system.). As in the original design philosophy of the Unix operating system, each tool in MVP is designed to 'do one thing, and do it well'). The power of MVP, as in Unix, comes from the tools being able to be used in conjunction with each other. As in object oriented programming,
Interactive graphics based software tools for rapid prototyping

each tool runs as a separate process and communicates with the other tools via specific data paths and protocols\(^4\). Each tool runs in its own window. The window is either open / in use, or closed. In the closed state the tool is shown as an icon inside the MVP toolbox. In either state the tool is always running; therefore when a closed tool is opened it is up-to-date.

Many tools will be needed to implement all the different types of interfaces for consumer products. Most of the tools in the MVP tool box are not novel or unique, and at present only a minimum implementation of each tool exists\(^5\-8\)). It is the integration and simplicity of these tools that is critical to the goal of this work. Each MVP tool can be developed and improved independently without affecting other MVP tools. Discussed in following subsections are four of the more important building tools that will be used by most applications.

2.1.2.1. **Functionality editor**

The tool to edit a functionality diagram is the heart of setting up a simulation. The proof tool gets most of its information to run the simulation, from the data structure that the functionality editing tool sets up. To the user of MVP the functionality editor tool is for entering the various functions on the prototype that a consumer will interface to, and the relations between the functions. Fig. 1b shows the functionality editor ready for use in the upper left-hand corner.

Currently, the commands to the functionality editor are for adding, deleting, connecting, and disconnecting functions. The name of a function to be added can be entered via the keyboard. All other input to this tool is through the mouse. The functionality editor highlights the 'current' function, which is the function to which all commands refer. When a function is added, it is automatically displayed with a connection from the current function. 'Connect' links the current function with any other legal function that is picked with the mouse. 'Delete' removes the current function and any other function that would be left 'dangling' (inaccessible) by the deletion. 'Disconnect' is for taking away a link between two functions, and removing any dangling function.

2.1.2.2. **Drawing/animation editor**

The drawing tool was written to allow for realism and user feedback during simulation of a potential interface for a product. It serves two related needs. The first function is 'drawing' with the mouse. This allows for entering a drawing of the prototype product or the 'looks' of the machine. Only static parts of the prototype are entered at this point in the drawing tool. The draw-
ing tool, ready for use, is shown in the lower left-hand corner of fig. 1b. Other
tools and the animation part of the drawing tool will be used for drawing
dynamic displays, function buttons, and moving parts.

When selecting a function would cause some part to start moving, an
animation of that movement should be shown during simulation. Without it,
a person using the simulation to test or evaluate the prototype would not get
proper feedback when trying different control functions. To specify what the
motion or movement to take place looks like, the animation editor in the
drawing tool is used. Using the draw primitives in the drawing tool, the de-
signer specifies what moving parts look like before movement starts and after
movement stops. The animation program then automatically generates ‘in-be-
tween’ frames from the two keyframes provided. The images of these in-be-
tween frames are stored for rapid retrieval. During simulation, rapidly dis-
playing them gives the effect of movement.

2.1.2.3. Button or icon editor

The button editor is similar to the drawing editor except that it is for drawing
more finely detailed objects, and that each button drawn is associated with a
function in the functionality diagram. Drawing in the button editor is done on a
5X magnification scale. An actual size image is also displayed simultaneously.
An open version of the button editor tool is shown in the upper right hand
corner of fig. 1b. Whenever a different function is highlighted in the func-
tionality editor, the corresponding button for that function comes up in the button
editor. That button can then be edited in the button editor. When the button
tool is edited, by the cursor tracking the mouse leaving the window for the tool,
the edited picture of the button is automatically sent to the proof tool, which is
where simulation takes place. This method assures that all tools are consistent
and that the simulation is always that of the most up-to-date specifications.

2.1.2.4. Proof tool

The tool for simulation of a prototype product is the proof tool. It also has
one other function, which is to position the buttons specified in the button
drawing tool on the product. This is done by using the mouse to ‘drag’ the
button around in the proof tool window until its placement is correct. Before
being positioned, the buttons for the current state of simulation appear along
the top edge of the proof tool window from left to right.

The design shown in the proof window operates in the way specified by the
other construction tools. The design can be tested by ‘picking’ (with the
mouse) function buttons on the design. Any action or change of state asso-
ciated with that button will then take place.
2.1.2.5. Other tools

Other tools currently in the MVP toolbox but not discussed below are a library, ‘C-shell’, print, help, and quit tools. The library tool is not yet implemented but will be for retrieving parts of other designs that can be used in the current project. The ‘C-shell’ window currently exists as a way of getting into the Unix ‘csh’ environment without having to exit MVP. Printing a hard copy to a laser printer of the current screen image is done by picking the print tool icon. When this tool is selected the print tool icon is displayed in reverse video and all i/o to the screen is frozen until the scan of the screen is complete. Help will print out a description on a given tools use. Selecting the quit icon puts up a menu which allows for exiting MVP without updating, exiting MVP and making all changes, and returning to MVP.

2.2. MVP design implementation

2.2.1. Hardware

MVP runs on a Sun-2/120, a 68010 based workstation from Sun Microsystems Inc. Our configuration includes two megabytes of main memory, 42 megabytes of formatted disk storage, optical mouse, keyboard, four serial lines, ethernet interface, and an 1152×900 black and white display. Because of the large size of the Unix kernel and all the bit mapped graphics images that MVP uses, two megabytes is the minimum memory requirement to avoid excessive paging.

2.2.2. Software

Sun Workstations run a version of the Unix 4.2 BSD operating system. Sun Microsystems provides an environment called SunWindows, in which MVP runs9). In SunWindows, individual processes, each running in a separate window can be started; these processes are called tools. Each tool can be further broken down into subwindows. During initialization each tool and each subwindow within a tool sets up what input events it is to be notified of, and what routines will be executed upon notification. This is done via a combination of system calls and by changing values of pointers in a system data structure associated with each tool and subwindow. Some of the input events used in MVP are clicking of the mouse buttons, movement of the mouse, the mouse entering or exiting the window associated with a tool, and activity on a specified communication channel. There are some special purpose subwindows provided. The ones used in MVP are message subwindows for writing messages to a user and option subwindows for creating and reading menus. In general, tool subwindows are written to by using pixwin and pixrect graphics routines supplied by Sun Microsystems.
The code of MVP is written in the proprietary programming language 'e'. Code written in 'e' is run through a preprocessor and translated into 'C', then run through the normal system 'C' compiler. The main difference between 'C' and 'e' is that 'e' is indentation sensitive, thereby alleviating the need for curly brackets. Other syntactical differences from 'C' exist, but the semantics are identical.

2.2.3. Tool communication

The MVP environment is an integrated set of interactive graphics based tools. Since each tool runs in a separate window as a separate process, a communication mechanism must exist between them. This is accomplished through the Unix pipe facility, shared files, and the asynchronous i/o facilities of tools and their subwindows.

When MVP is first started the user is prompted for a project name. This name is sent as a command line argument to each of the tools along with the file descriptors of any pipes that that tool may use. The pipe connections for the tools discussed above are shown in fig. 2. The pipes are used to send commands and small amounts of data between tools. Intermediate files are used to pass larger amounts of data between tools such as raster images and display lists.

A shared memory scheme was originally planned for keeping all tools up-to-date and using the same data structures and images. This would have led to faster synchronization of tools. The usual problem found with using shared memory is arbitration between multiple processes trying to read and write to the same memory location simultaneously. This would not have been a consideration since in the MVP environment each tool is 'triggered' by a user event and no two processes would be running simultaneously. Because imple-
menting shared memory is non-trivial and must be configured into the kernel under 4.2 BSD Unix, a command passing scheme was used. Each tool replicates copies of necessary data structures and images. When a tool makes a change to a data structure or image, a command is sent out to other tools that use a similar data structure or image. These tools in turn update their private copy.

2.2.4. Data structures

Information that each tool gets about building an application is forwarded over the communication pipes to the proof tool, which is where the simulation takes place. This information must be readily accessible to be interpreted during simulation. The primary data structure used is a graph with each node representing a possible state of the prototype. Each node is doubly linked to adjacent nodes for easier movement about the graph. For the most part, the state diagram representing the functionality of a device follows a tree structure, therefore the nomenclature associated with trees is used for describing the state diagram representation. The structures are not true

![Diagram of 'state' data structure](image-url)
trees however, because cycles can exist. The data structures used in simulation are graphically depicted in fig. 3.

The data structure labeled ‘state’ in fig. 3 consists of two types of information. Information is stored for getting around the functionality diagram that was set up in the state tool. Also pointers are kept to structures that are set up by each of the other tools. The structures set up for the pictures and positioning of buttons, and animation sequences are shown.

3. A design example using MVP

The design of a small audio cassette player is now illustrated. The technology visualized in this example design is that the cassette player will have a flat surface pixel addressable display with a touch sensitive surface. This will allow for dynamic changing of buttons on the display, where only buttons that have significance at the current point in time are shown. For example a fast forward button is not show while the cassette is rewinding.

The order in which a design is built up is somewhat flexible. In this design the functionality editor is the first tool chosen from the MVP toolbox. Fig. 4 shows the functions and their relations for the example cassette player. The

![Fig. 4. State diagram of example cassette player.](image)

‘top’ function always exists as a starting point. The first user defined function is ‘on’. Once the ‘on’ function is selected, the functionality diagram shows that the functions ‘off’, ‘play’, ‘forward’, ‘reverse’, or ‘eject’ can be chosen. Each of these will cause a change to a new set of options or cycle back to a previous set. The ‘tape in’ function is really a hardware triggered function,
Interactive graphics based software tools for rapid prototyping

not a function triggered by the user pushing a button. Since in the simulation there is no way for sensing hardware events, the function will be 'hand triggered' by associating it with a button. This 'pseudo' button most likely would not be shown on the product being simulated, but possibly off to the side.

Next, a picture of the cassette player is drawn using the draw tool. Fig. 5 shows the drawing. Upon exiting the window for the drawing tool the image is sent to the proof tool where simulation will take place.

The button drawing tool is chosen next out of the MVP toolbox. This tool must be used in conjunction with the functionality editor. When a function is chosen in the functionality editor, the corresponding button is shown in the button editor. Fig. 6 shows the drawing of the button for the 'stop' function.

Finally, the proof tool is taken from the MVP toolbox. As mentioned earlier, the proof tool provides two main functions; button placement and simulation. Fig. 7 shows screens of a simulation for the cassette player just built.

Fig. 8 shows all building tools and the simulation tool open. Note that all tools are synchronized. The button editor shows the button for the state that the functionality editor is showing and the buttons shown in the proof window are associated with the 'children' of that function.

The design of the cassette player is not complete. For instance it is lacking volume control. Volume control is really only necessary while the tape is play-
Fig. 6. Drawing of fast forward button.

Fig. 7. Proof window during simulation, (a) initial machine state; (b) after pushing on button; (c) after pushing play, forward, or reverse; (d) after pushing eject.
Interactive graphics based software tools for rapid prototyping

Fig. 8. MVP design environment.

ing, but it is probable that people are used to having it available at all times. This would be a good question for a market study. Using the MVP environment ‘volume control’ could easily be added as a function under all states or as a function just under ‘play’. Using MVP these changes are quick and easy. Using the more established method of building hardware prototypes to test ideas like these is more expensive, slower, and less flexible.

4. Future goals

Once the current goals of MVP are met, there are other areas of work that may supplement and/or complement MVP. Two such areas are code generation and design checking.

Currently the MVP tools are used to build a product interface simulation. If a description of the devices that will be used to build the interfaces in the product being developed are available to MVP, and are programmable, control code for this hardware can be generated. This code could be downloaded to a prom burner and then tried in the target machine. This would give faster and easier turn around for testing and modifying a hardware prototype.

There exists a wealth of knowledge in the literature on the design of user interfaces in terms of ergonometics\(^2\)). This knowledge is an ideal candidate
to be put into an expert system. If such an expert system were incorporated into MVP, then design evaluation could be performed.

5. Conclusion

The main goal of MVP is to provide an easy to use system for setting up and simulating interfaces to consumer products by people without any computer science background. This has been accomplished using the Unix tool concept and visual interactive programming (VIP). The areas most likely to benefit from the MVP design environment are product interface designing and market analysis of a product. Although much work remains in making MVP a truly universal design tool of production quality, the concepts behind it appear to be solid. Current efforts are in designing new tools for the MVP toolbox and in refining existing tools. If MVP proves successful in designing consumer product interfaces, the concepts can be easily expanded to other application areas.

REFERENCES

4) A. Goldberg and D. Robson, Smalltalk-80, Addison-Wesley, Reading, Massachusetts (1983).
6) Suntools, Icon Tool Section, Unix Manual Page.