ADEW

A Multimedia Interface Builder for Andrew

Thomas Neuendorffer
Information Technology Center
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, Pa. 15213
(412) 268-6108
tpn@andrew.cmu.edu

Abstract

The Andrew Development Environment Workbench (Adew) is an application builder for the multimedia Andrew Toolkit. It is different than other application builders in that it is built within a systems architecture that support the arbitrary inclusion of objects within other objects, allowing the application to be built within text, spreadsheets, drawings, or other substrates. This paper will give an overview of the Adew system and provide some insight to the problems encountered and their solutions.

Keywords
Andrew
ATK
Interface Builder
Multimedia
Rapid Prototyping
X

From
Proceedings Multi-Media Communications, Applications and Technology Workshop,

1-2 July, 1991, Sydney, Austrailia
With illustrations updated to match Adew release 5.0

1 Introduction

The Andrew Development Environment Workbench (Adew) was designed to simplify the creation of Andrew Toolkit (ATK) applications. It provides an extendable collection of widgets (buttons, switches, thumbwheels, etc), tools for creating multi-media documents, and a code generation program that takes care of the details of writing dynamic objects. It can be used for rapid prototyping, and for allowing non-programmers and programmers alike to design applications. Applications using embedded objects are created by a series of copy-paste operations, with support for editing object attributes. Sample applications include a calculator (fig1), a simple help program (fig 2), and a piano object (figs 3a and 3b). It has also been used to provide interfaces to external devices such as digital scanners (fig 4), and to define interactions between an experimental gesture sensor and a rack of synthesizers (fig 5). The Arbcon control panel for creating multi-media documents (fig 6) is itself a product of these tools.

This paper is an expansion and update of the "Adew: An Overview" paper, originally presented at the Third Annual X Window conference in Boston in the Spring of 1988, prior to time when that conference started publishing proceedings. Adew was probably the first publicly available interface builder under X, and it continues to be a unique example of the genre.

2 Basic Concepts

For those unfamiliar with ATK, the following is the abstract is from *The Andrew Toolkit - An Overview* [pal88].

> The Andrew Toolkit is an object-oriented system designed to provide a foundation on which a large number of diverse user-interface applications can be developed. With the toolkit the programmer can piece together components such as text, buttons, scroll bars, an other graphical controls. to form more complex components. It also allows for the embedding of components inside of other components, such as a table inside of text or a drawing inside of a table. Some of the components included in the toolkit are multi-font text, tables, spreadsheets, drawings, equations, rasters, and simple animations. Using those components we have built a multi-media editor, mail system, and help system. The toolkit is written in C, using a simple preprocessor to provide an object-oriented environment. That environment also provides for the dynamic loading/linking of code. The dynamic loading facility provides a powerful extension mechanism and allows the set of components used by an application to be virtually unlimited. The Andrew Toolkit has been designed to be window system independent. It currently runs on two window systems, including X.11, and can be ported easily to others.

The paper goes on to discuss a key concept of ATK, the notion of *data objects* and *views*. The implementation of the *data object* is responsible for the reading, writing and manipulation of the object's permenent data. For multi-font text, it is the ASCII codes for the text, along with formatting information, such as which characters should appear as **bold**. This data is written in an object specific format refered to as a *datastream*. When objects are embedded in other objects, their datastreams are also embedded, allowing multiple objects to be saved in a single file. The *view* is responsible for displaying a data object on a screen and reacting to user input, such as keystrokes and mouse hits. An ATK application window consists of multiple *views* logically arranged in a tree, where each *view* is the child of some other *view*. This arrangement is called the *view tree.* Andrew allows for multiple views to view a single data object. Thus a table of numbers could be displayed either as a spreadsheet, a bar graph, a pie chart, or an animated flock of birds (fig 7) [ada87] . *Data objects* and *views* usually exist together as an *inset*.

Given that insets can be embedded in other insets, different views can observe a given data object, and data objects can have multiple views, a developer has a lot of possibilities in creating a document or an application. For all the power ATK affords both users and programmers, the flexibility and options create problems of complexity and the need for consistency. For example, what is the proper way to tell an editor to insert a pie chart into a drawing, pointing to the table data object being displayed three pages back in the surrounding text object?

A bigger problem has been how to reduce the learning curve so that the ‘casual’ programmer can create ATK *applications*, without having to learn the intricacies of creating new views, data objects, classes, etc. Especially where *application* is defined as not just a simple stand alone program, but as an extendable object that can be pasted into a paper, sent as part of a mail message or posted on an electronic bulletin board.

3 Enter Adew

Adew, the Andrew Development Environment Workbench, was designed to address these problems. It is a system grown, in part, out of earlier work on Glo [neu86], the Graphic Layout Object. Glo was an attempt to aid people creating applications with BE1 [gos84], ATK’s predecessor. Unlike Glo, which was a large monolithic system, Adew is a collection of smaller insets that work together. This approach offers several benefits. It allows the application developer to use only those pieces that are appropriate to the application. Also, by not tying functions into a single complex inset, the developer has the freedom to use whatever insets seems appropriate for a given application.

The Arbcon (fig 6) is a key part of Adew. Displayed in an external window, it gives the application creator access to information and options that will normally be hidden from the final user. At the same time, a knowledgable user can use the Arbcon to customize existing Adew applications to fit their style of interaction. A customized version of the Arbcon itself has already been created and used toward this end.

The Arbcon serves several functions.

1. It is an inset generator that can place an instance of any type of inset into the cut-buffer, where it can then be pasted into any of the ATK insets that support child objects.

2. It is as an editor for inset attributes.
3. It can cut or copy insets from existing Adew applications or templates.
4. It can copy links to existing insets, allowing the user to paste in new views on existing data objects.

The development of Arbcon has been one of stages. The first stage was figuring how to add every desirable function. This resulted in the alpha version with its many switches and places to enter information. The most recent stage has been one of simplification, where the goal has been to pare down the interface to a simple bare minimum, without losing any of the underlying functionality.

Adew also includes a number of user-interaction insets (fig 8). These include switches, sliders, thumbwheels, buttons, bargraphs, single and multiple string entry insets, and a clicklist inset for setting up lists of strings that the user can chose from via mouse clicks. These insets, consisting of values and valueviews, all have Arbcon editable visual attributes, such at the label on a button or the range of values on a slider switch. Valueviews are all views on the *value* data object.. This provides a simple consistent interface for the programmer, who doesn't necessarily have to worry about whether the user entered a number via a thumbwheel, a slider, or by just typing it in from the keyboard. The latest release of Adew includes a Motif-style multiple button valueview, that can be used to replace the button, on-off, and four-way style switches. Other Motif-style interaction objects are currently being planned.

4 Creating Applications

Applications, as defined above, are created using a simple copy-paste paradigm. Enclosed insets may be copied from Adew applications, or from templates supplied by the user or system (fig 8). The Arbcon can also create new insets on the fly. Clicking the mouse button on one of the inset names listed in its top-middle window will place a copy of that inset into the cut-buffer. New inset names may be added at any time.

Creating a new view on an existing object is done by clicking on the desired inset, throwing the Arbcon's Link switch, and (if a different view type is desired) clicking on the desired inset name. Again, the necessary information is placed in the cut buffer so that when the new inset is pasted, it will be a new view on the desired existing object.

Once in the cut-buffer, the inset may be pasted into place. While Adew provides one layout inset (the tiling *lset),* any inset capable of supporting embedded insets (text,tables,drawings,etc.) may be used to create applications. These insets may also be nested in one another to create complex layouts. The page inset can be used to create panels that switch between different sets of objects under application control.

The various insets can all provide useful functions for applications creators. In the piano application, for example, a text inset is used to store the song as it is played on the keyboard. Users can scroll and edit the song as text, and routines to read and write the songs simply call the text read and write routines. In fact, one could have gotten away with no explicit read and write code at all, since the current song automatically gets saved along with the application.

Whenever a new inset is created, its attributes are displayed in the Arbcon, where they may be edited to change how an inset looks or is named. All insets have at least one attribute - a logical name may be used to reference the inset. As noted above, valueviews usually provide other editable attributes. To edit the attributes of an existing view, it is only necessary to click on the desired view to reveal its attributes.

It should be noted here that these applications are created, stored, and read as datastream files by ez, the standard ATK editor, and are treated the same as text or any other ATK inset. In addition to the file specific data, datastreams also contain an indication of what object is needed to read this data. This allows for the creation of editors like ez that can be use to edit any type of object (text, drawings,spreadsheets, etc.), since, in reading the file, it can determine what kind of inset it needs to edit that object. These insets are then dynamically loaded as needed. Similarly, as objects are embedded in one another, their datastreams are embedded when the file containing them is saved, allowing multiple objects and their layouts to be saved together in a single file. The parent objects need only know about dataobjects and views, and not about the specifics on the insets themselves. All of the figures are of files created interactively with ez and the Adew tools. When created like this, they contain editable text and switches that flick on and off, but don't, by themselves, do anything. The missing piece that needs to be added is a *controller.*

5 Creating Controllers

A *controller* is defined as a module that sets up interactions between different insets and ties them together into an application. Like insets, they are dynamically loaded into the application when needed. Unlike insets, controllers take no screen space and are not directly connected to the hierarchy of data objects or views .

Createcon is a program, callable from Arbcon, that reads over a datastream describing the application layout and generates the files needed for a controller object. These consist of a `.c` file, a `.ch` file, an X11 `Imakefile`, and a shell script to create the makefile and compile the controller. The `.ch` file contains the necessary definitions for creating a dynamic object using the ATK Class system. When compiled, these files produce a dynamic object with pointers to all the data objects and views contained within the application, and call-back procedures set up to be called whenever a change is noted in a value object. This controller, while it does nothing by itself, contains comments indicating where a programmer can insert code. By automating all of the 'housekeeping details' of creating a new controller, and tailoring its contents to a specific set of insets, createcon allows applications to be created with very few lines of developer written code. The classic calculator example (fig1) implements a desk calculator in under a hundred lines of developer written C code. And that includes 35 lines of string parsing code so that data can be entered by typing in to the text entry area.

If a developer decides to change the interface of an existing Adew application, adding new buttons or changing functionality, createcon will merge the developer-written code of the old version into the new version. Because of its basis on ATK, applications built with Adew are portable between machines and window systems. Because Adew applications consist largely of machine generated code, upgrading old Adew applications to new versions of ATK and Adew have been automatic. Application developers who worked with the Alpha Adew and X11 release 3 were able to automatically update their code to the X11 Release 4 system by processing it with the Release 4 createcon.

The separation of the look of the application from its functionality provides additional advantages. It means that non-programmer designers can piece together the way they think an application should look, before turning it over to a programmer for coding. In this way, Adew can also be thought of as a rapid prototyping system. Unlike most such systems however, the prototype actually helps generate the working code, and then becomes the application itself.

6 Problems and Solutions

The separation of the controller code from the layout of the included insets provided many benefits, however it also presented problems. One was how to link the application insets into the controller code. Since the application consisted of both views and data-objects, and could itself be split into multiple windows, it was necessary to create the system where each new window would have a separate instantiation of the controller object controlling the separate views. Since ATK did not provide a logical naming scheme for insets, or any general method for locating dataobjects in an application, or views in a window, it was necessary for Adew to find a way of providing a naming scheme that would not require anything new or special from the insets themselves. This was accomplished by two new insets, the *cel* and the *arbiter*.

6.1 Cels

The *cel* inset can contain any other inset in the system and provides a logical name with which to access the inset. To create new objects, the Arbcon actually places a cel's datastream into the cut buffer. When pasted into a parent, such as text or lset, the cel creates the desired child object and inserts it in such a way that the cel inset itself is transparent to the user. The cel view is able to locate and negotiate with other cel views within the view tree in which it is placed. The registered cels are reported back to the Arbcon, which then provides the user interface for manipulating the cel's data, which includes the cel's logical name, and various attributes that the cel passes on to its child object.

6.2 Arbiters

To prevent namespace collisions, and to provide this link whereby the Arbcon and the cels can interact, it was necessary to create to *arbiter* namespace object. When a cel registers itself, a call is made up the view tree to find the arbiter that is supporting that namespace. If none is found, a phantom one is automatically created at the top of the view tree. Arbiters are subclasses of cels and may be explicitly included anywhere in the view tree, providing a hierarchy within a normally flat namespace. This can be useful when including multiple instances of an Adew application in the framework of a larger document or application, since it will prevent namespace collisions. Early versions of Adew required these arbiters and cels to be explicitly included in the application, however this is no longer necessary. In fact, most Adew application creators don’t have to know anything about them, since the interface that uses them is part of the code automatically created by createcon.

6.3 Connecting Controllers with Application Layouts

Arbiters and cels provide a framework where any inset can identify and locate any other inset within the viewtree. The next problem involved the linking of a controller object to a specific view tree. The initial approach to this problem involved the use of a *control-button* inset. One of the *control-button* attributes that could be set from the arbcon was the name of a procedure in an object that would be called when the button was clicked. By specifying the name of a controller object, that controller was able to receive a pointer to the *control-button*. This provides the controller a handle on the view tree from which pointers to all of the enclosed objects and views can be derived. Later versions of Adew also provided an application called *runAdew*, that would take the application datastream and controller class as arguments and would provide the link directly. The advantages to runAdew included the ability to pass command line arguments to the controller class and the elimination of the need for the *control-button*. It allowed the application to run in a more traditional single-window stand-alone mode, but, if the application creator did not include the *control-button*, the application could not function as an embedded inset. The latest version of Adew (available with Andrew patch level 10) provides another option for solving this problem, the ability to use Adew to create true insets.

6.4 Creating Insets with Adew

In addition to the need for a control button, there are other drawbacks to the controller/datastream model on an application that Adew has supported. While Adew applications could be embedded in other objects, the model was different than ATK users were accustomed to (instead of embedding an inset, one inserted a file). The application datastream would be saved along with every application that included it, using unnecessary disk space. There was also a problem that controllers were not able to get a handle on embedded dataobjects until a view on that object was inserted into the view tree, an action that might not occur until late in the running of an application, when the user scrolled embedding text down to that object’s location. Another problem was that there was no direct way for Adew application to save data associated with the application, except to include the data within one or more of the objects that were part of the application.

These and other problems have recently been solved in the latest version of Adew, which is capable of producing true insets. The new createcon can be set to create code for a dataobject and a view instead of a controller. A representation of the datastream containing the object layout information is automatically specified in an include file and stored as part of the dataobject. When the object is instantiated, it will read this representation by default if there is no existing datastream to be read. This allows the application to either save the datastream when it is saved, supporting applications like the piano, or never saving it, reducing the amount of disk space whenever the application is saved. The object that is created is a subclass of the arbiter and therefor acts as its own namespace.

6.5 Improving Availability of Dataobjects

Regardless of where the application datastream is stored, the creation of child objects is instigated by the reading of a datastream. The new Adew takes advantage of this fact by having the application declare itself as the current ‘master’ before it calls its child’s read routine. The subsequently read cel objects announce their existence to the current master as they are read, allowing the application to build up a list of child objects at the time they are created. The user’s application code is provided with methods for accessing these dataobjects at any time, without having to wait for the view on that object to be established and placed in the view tree When an Adew application contains another Adew application, the read process will push the current master on a stack, so children are always declared to the proper master. This whole process is transparent to both the embedded objects and the application itself.

6.6 Support for Storing Application-Specific Data

Another new feature of Adew is the is the capability of saving application-specific data. While Adew applications were able to save information in the datastreams of it child objects, there was no provision for the Adew application itself to maintain its own datastream of application-specific data. In keeping with Adew policy of making things as automatic as possible, the new Adew allows the application creator to declare variables that will then be automatically saved and restored along with the application itself. Provision are made for new variables to be later added in a backward compatible fashion. Thus, if an application developer desires an array of 50 long integers is important to the application, that array only has to be declared. Adew will write the code for declaring, initializing, saving and restoring that array in a legal ATK datastream that will accompany the application.

6.7 Backward Compatibility

Despite all of these changes to the Adew model of applications, old Adew applications are still supported. Not only does the controller model still work, but createcon supports flags for automatically upgrading existing applications into insets. It does this by migrating the existing controller code into the newly created view code, where the same callbacks, child object pointers and application methods are supported.

7 NESS

An alternate way of creating controllers is provided by NESS, an interpreted extension language being distributed with Andrew. This language serves essentially the same function for the ATK as Hypertalk serves for buttons and fields in Hypercard. The innovative portions of the language are its string processing facilities, and its techniques for extending insets with scripts. It allows one to create self-contained applications that don't require a separately loaded control module. See the Hansen paper [han90] for more details.

8 Uses of Adew

Adew isn't designed to replace all ATK programming, it is for mainly for applications that can be produced in large part by combining existing insets. But while Adew may not be useful for writing the next WYSIWYG graphics editor, it has proven its usefulness in prototyping interfaces, writing test programs for new insets. creating control panels for external devices, interfaces to existing non-windowing programs, customizing interfaces to existing insets, creating demo programs, etc.. Its use in providing a video codec was demonstrated by Beadle [bea90], and its use in producing document management systems has been described by Beer [bee90]. It also serves as good way to learn ATK programming, and may be just the thing for professors and others with specific applications in mind, but without the time or desire to learn all of the intricacies of creating an inset. As other insets are developed, the applications that can be developed with Adew will only increase.

9 Summary

From a user's perspective, Adew consists of the following:

1. The Arbcon, an editing tool that sits in a separate window to aid in the creation and editing of groups of insets.
2. The Value insets, a set of buttons, switches, and string entry objects for user interaction.
3. The Lset, an inset designed to lay out embedded insets in a tiled fashion.
4. Createcon, a program that takes a group of insets and writes a controller object for them that makes it easy for programmers to define how the insets should interact for a specific application.

Together, they provide an environment for:

1. Collecting groups of insets into prototype applications.
2. Turning those prototypes into functioning applications.

10 Conclusion

ATK takes advantage of an old Unix® idea, that complex applications can be often formed by the combination of multi-purpose objects. Adew attempts to expand on this idea in a object oriented windowing environment. Where Unix provided shell scripts to combine many components into one, Adew provides the Arbcon . Where Unix provides pipes to connect the components, Adew creates a controller where the variety of interactions can be described. The untimate goal of Adew is to allow the application builder to create a whole that is greater than the sum of its parts.

11 References

[ada87] Douglas Adams, *Dirk Gently's Holistic Detective Agency*, Simon and Schuster, New York 1987

[bea90] Peter Beadle, Gregory J. Wilkins, John Judge, Murray Macpherson *A Control Methodology for a Variable Bit Rate Video Codec*, Proceedings Australian Video Communications Workshop. Melbourne, Australia, July, 1990

[bee90] Martin D. Beer, *Developing Document Management Systems Using the ANDREW Toolkit.*,Proceedings of UKUUG Summer 1990 Conference. London, July 1990

[gos84] James Gosling and David Rosenthal, *The User Interface Toolkit*, Proceedings PROTEXT 1 Conference, 1984

[han90] W. J. Hansen, *Enhancing documents with embedded programs: How Ness extends insets in the Andrew ToolKit*, IEEE Computer Society International Conference on Computer Languages, 1990

[neu86] Thomas Neuendorffer, *Glo - A Tool for Developing Window-Based Programs*, Proceedings USENIX Technical Conference, Winter 1986

[pal88] Andrew J. Palay, Fred Hansen, Mike Kazar, Mark Sherman, Maria Wadlow, Thomas Neuendorffer, Zalman Stern, Miles Bader, Thom Peters, *The Andrew Toolkit - An Overview*, Proceedings USENIX Technical Conference, Winter, 1988

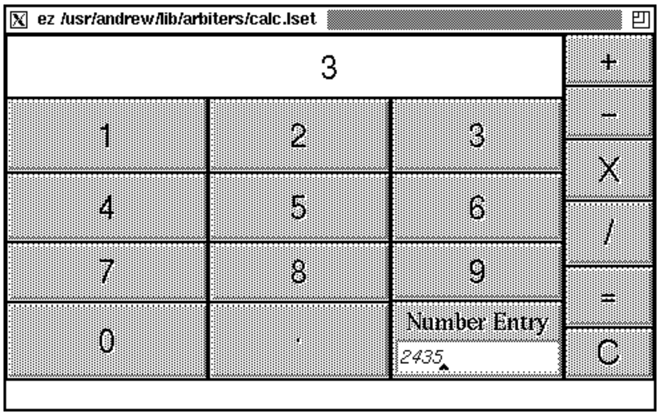


Fig. 1. A simple calculator program. The controller program required only about ˜100 lines of user-written code (or ˜65 lines without the parser that allows textual entry of values and operations).

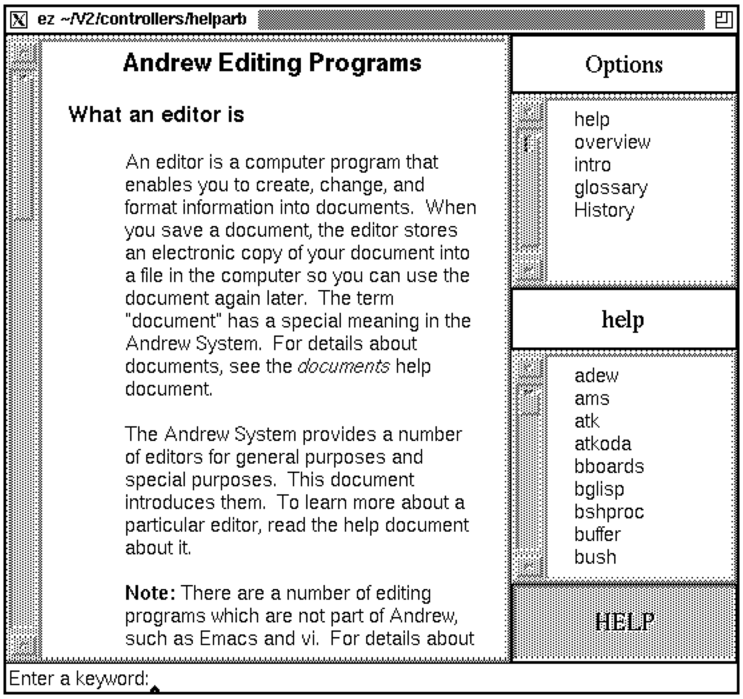


Fig 2. A prototype help program.

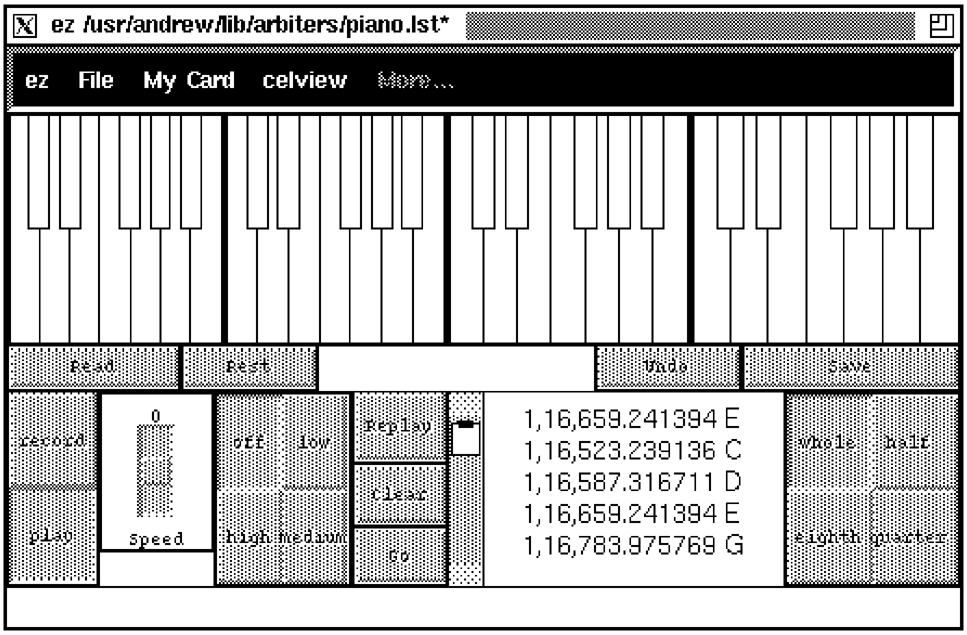


Fig 3a. An playable electronic piano. The song, as it is played on the keyboard, is stored in the scrollable textview area as text, and thus takes advantage of the text editing, reading and storing code of the standard ATK text object. As with other applications, it can be cut, pasted and sent as electronic mail.

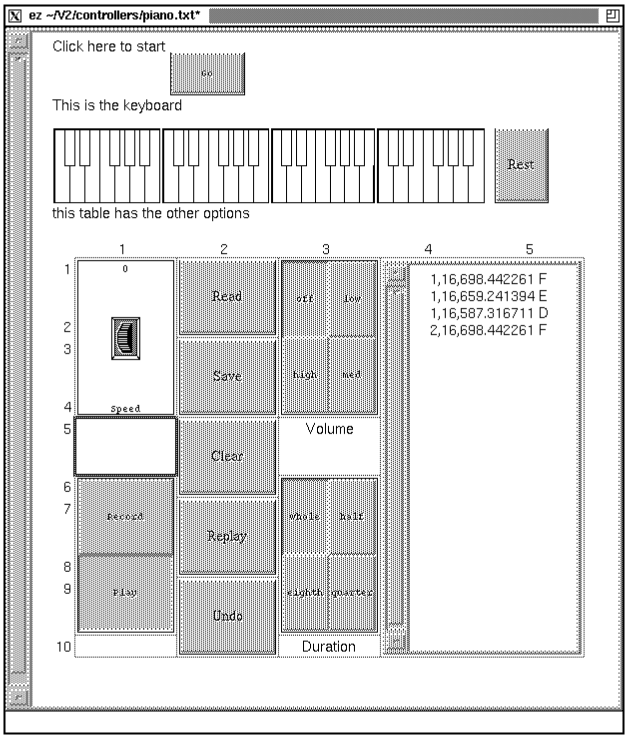


Fig 3b. A variation on the piano in figure 3a, this piano was created by cuting and pasting the objects from the first piano into a scrollable text containing a spreadsheet, making some changes to the views involved. Also shown are the results of using the new Motif-style options. The code

for controling this piano is identical to the code for the piano above.

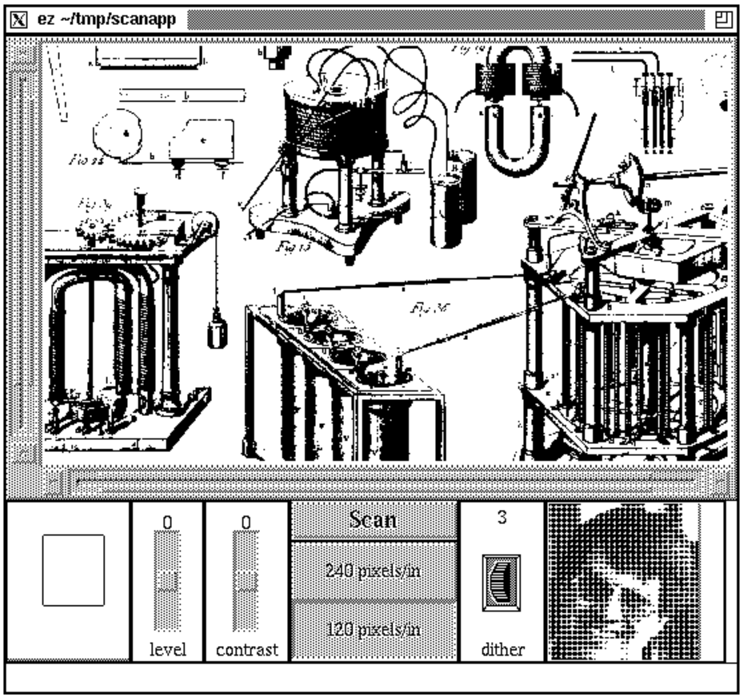


Fig 4. A scanner controller. When the Scan button is pressed, an independently developed scan program is called to operate a scanner, with the program flags set according to the switch settings. The program output is piped directly to the read routine of raster object in the top half of the window. This graphic interface required less than 50 lines of user written C code.

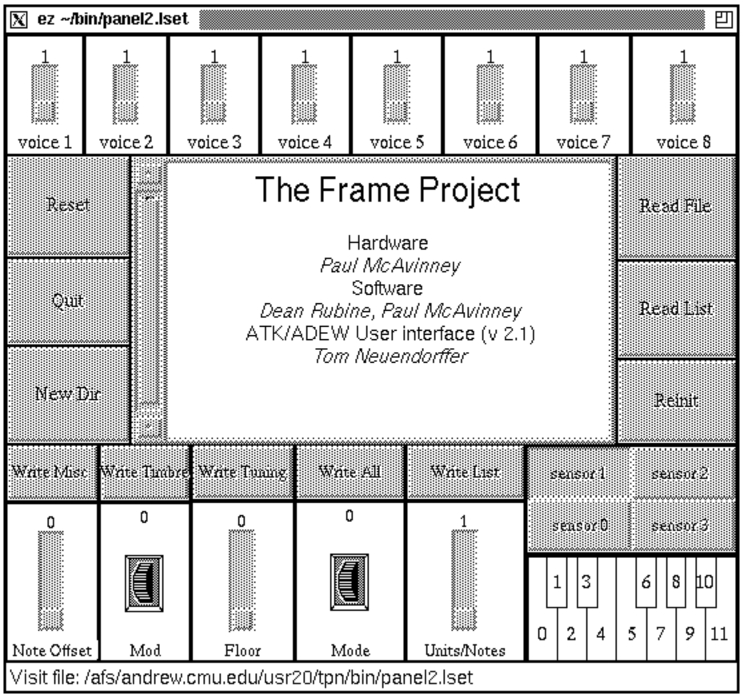


Fig 5. Like the scanner above, this was written as a graphic user interface to an existing vanilla C program. It controls the interactions between an experimental gesture sensing device (the Frame), and a rack of musical synthesizers. It can set up and store various types of musical configurations.

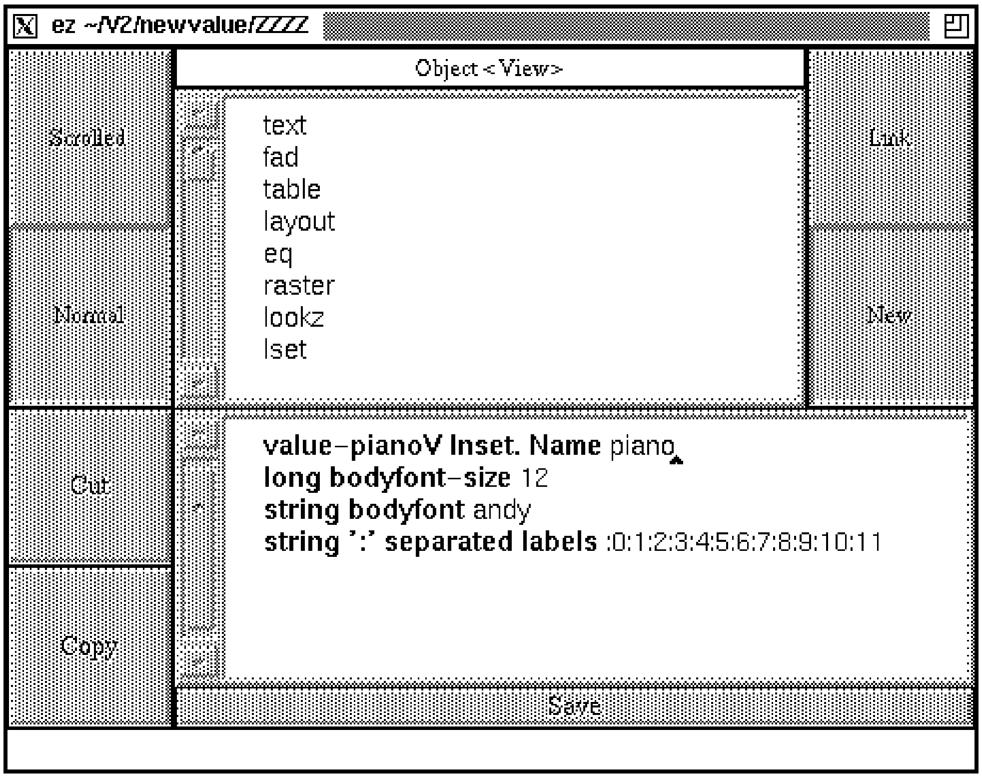
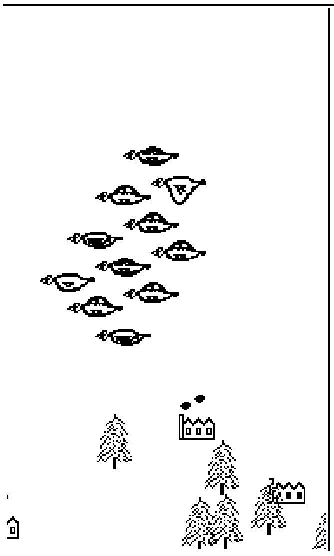


Fig 6. The Arbcon, a controller that is both a part of, and a result of the Adew package.

ez ~/itc/be2/flock/foo.lset dublin



	1	2	3	4
1	23	140	205	304
2	350	100	143	324
3	543	423	396	
4				
5				
6				
7				
8				
9				
10				

Fig. 7 A table of numbers with two views. On the right, the table is viewed and can be edited as a spreadsheet. On the left, the table is being displayed as an animated flock of birds, with each bird representing a number, and the relative size of the numbers being indicated by the speed that its bird is flapping its wings.

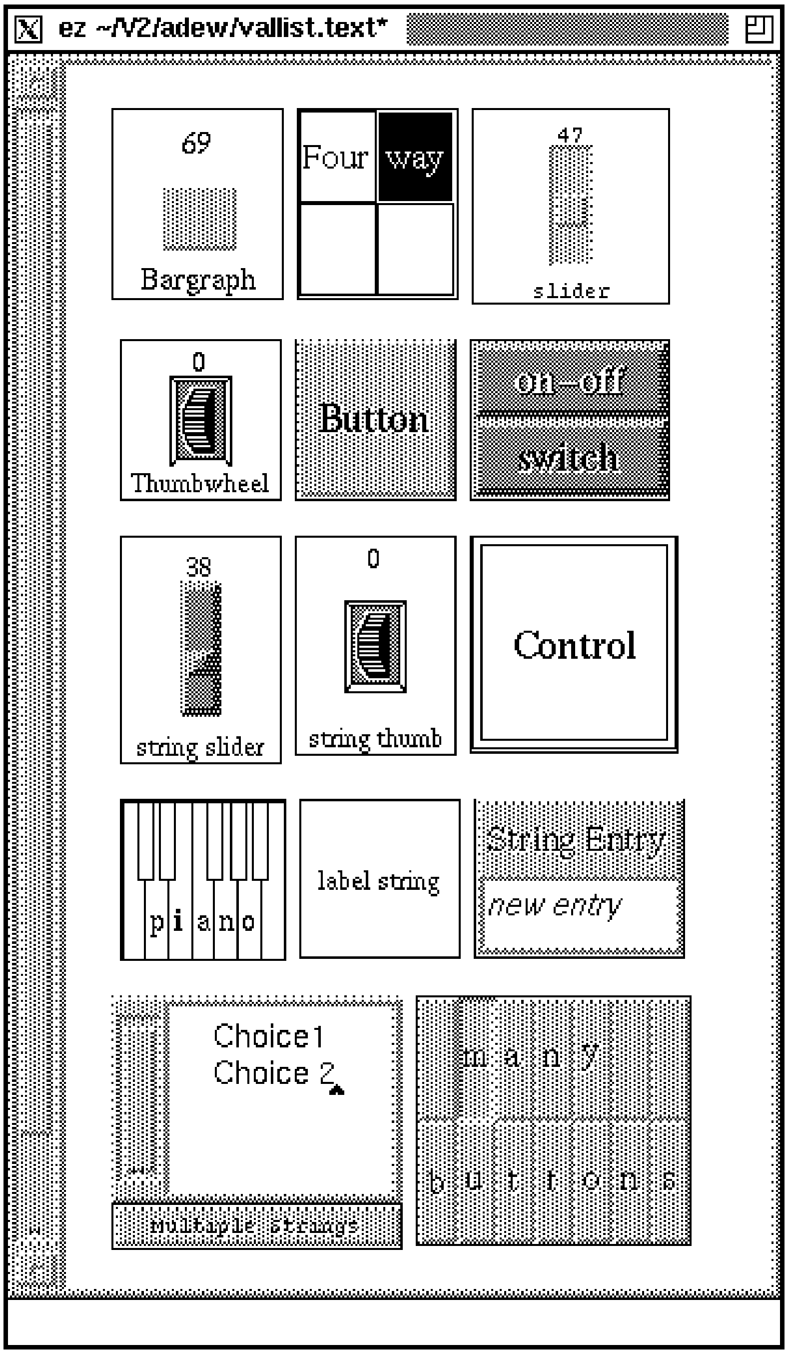


Fig 8. A template of some currently available interaction objects available for copying and pasting into an application. There is nothing special about this file, it is just a collection of objects in a textview. Developers can easily custom make their own, as any Adew file can serve as a template.